Redbooks

ibm.com/redbooks

IBM Db2 12 for z/OS Performance Topics

Akiko Hoshikawa
Bart Steegmans
Bharat Verma
Brian Baggett
Chang Kim
Chongchen Xu
Chung Wu
Cristian Molaro
Jasmi Thekveli
Kaitlin E. Murray
Liang Zhang
Lingyun Wang
Mai Nguyen
Neena Cherian
Nguyen Dao
Ou Jin
Peng Huang
Ping Liang
Robert Boon
Todd Munk
Xue Lian Zhang

Analytics

Information Management

IBM

Redbooks

International Technical Support Organization

IBM Db2 12 for z/OS Performance Topics

September 2017

SG24-8404-00

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (September 2017)

Contents

Notices

Trademarks

Preface

Db2 12 for IBM z/OS® delivers several significant business values, such as cost reductions, scalability improvement, and improved application enablement for cloud and mobile environment.

Db2 12 also introduces a foundation of continuous delivery. In continuous delivery, new capabilities and enhancements, including performance features, are delivered through function levels without waiting for a new release.

Function levels 100 and 500 activate the initial set of enhancements and capabilities in Db2 12. For migration from Db2 11, most new capabilities become available in Db2 12 only after the activation of function level 500 or higher. In this IBM® Redbooks® publication, we focus on the enhancements that are available in function levels 100 and 500.

Db2 12 for z/OS continues the trend by focusing on further CPU savings and performance improvement through innovations within Db2 and the synergy with IBM System z® hardware and software. While most of CPU reductions in analytical queries are built directly in Db2, other CPU and cost reductions are realized by using more real memory. Enhancements often take effect after the REBIND in the case of static applications, while some enhancements require user actions, such as activating function level, Data Definition Language (DDL) updates, or providing more memory for Db2 to use.

In this book, we describe the performance of Db2 12 for z/OS and the possible performance affects of the most important functions when migrating from Db2 11. We include performance measurements that were made in the Silicon Valley Laboratory and provide considerations and high-level estimates. Results are likely to vary because the conditions and work differ.

For the purposes of this publication, we assume that you are somewhat familiar with Db2 12 for z/OS. For more information about the new functions, see *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

Authors

This book was produced by a team of performance specialists working around the world.

Akiko Hoshikawa is a Senior Technical Staff Member with the IBM Silicon Valley Lab, where she leads the performance engineers for Db2 for z/OS development. She is responsible for driving performance-initiated development in each Db2 release, working closely with Db2 developers. Her areas of expertise include system level performance tuning, Sysplex, and XML. She has been involved with or written IBM Redbooks publications about Db2 for z/OS Performance Topics since 1998.

Bart Steegmans is a Consulting Db2 Product Support Specialist from IBM Belgium, currently working remotely for the Silicon Valley Laboratory in San Jose and providing technical support for Db2 for z/OS performance problems. He has over 29 years of experience in Db2. Before joining IBM in 1997, Bart worked as a Db2 system administrator at a banking and insurance group. His areas of expertise include Db2 performance, database administration, and backup and recovery. Bart is co-author of numerous IBM Redbooks publications about Db2 for z/OS.

Cristian Molaro is an IBM Gold Consultant, an independent Db2 specialist, and a Db2 instructor based in Belgium. He has been recognized by IBM as an IBM Champion for Information Management in 2009 to 2013. His main focus is on Db2 for z/OS administration and performance. Cristian is the co-author of several IBM Redbooks publications that are related to Db2. He holds a Chemical Engineering degree and a Master's degree in Management Sciences. Cristian was recognized by IBM as TOP EMEA Consultant at the IDUG EMEA Db2 Tech Conference Prague in 2011.

Kaitlin E. Murray is a Software Developer in the United States, based at the Silicon Valley Lab. She has a year of experience in Db2 performance in Db2 for z/OS development. She holds a degree in computer science from University of California, Davis. Her areas of expertise include query performance and analysis.

This publication also was written by the following authors:

- Bharat Verma
- Brian Baggett
- Chang Kim
- Chongchen Xu
- Chung Wu
- Jasmi Thekveli
- Liang Zhang
- Lingyun Wang
- Mai Nguyen
- Neena Cherian
- Nguyen Dao
- Ou Jin
- Peng Huang
- Ping Liang
- Robert Boon
- Todd Munk
- Xue Lian Zhang

Special thanks to the following people for their contributions to this project, and for their consultations and performance analysis:

- Terry Purcell
- Fenghui Li
- Jie Ling
- David Dossantos

Thanks to the following people for their contributions to this project:

- Jeremy B Jones
- Debbie Yu
- Steve Turnbaugh
- Emily Prakash
- Mike Schadduck
- Keiko Fukushima
- Ira Sheftman
- Richard Vivenza
- Ka Chun
- Hugh Smith
- Kevin Hite
- Jerry Stevens
- Brenda Beans
- Michael Woo
- Eric Katayama

This book was produced by the IBM Redbooks publication team with contributions from Martin Keen, IBM Redbooks Project Leader.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction

As with prior releases, performance and scalability improvements are major themes in IBM Db2 12 for z/OS. Performance optimization in Db2 12 for z/OS focuses on real-time analytics and using in-memory optimizations.

We describe the performance expectations of Db2 12 for different types of workloads in this chapter. Many performance improvements offer potential CPU and cost reductions by migrating to Db2 12 and performing rebind, while other new enhancements might require user actions or extra memory allocation to use them.

In this chapter, we also provide an overview of the performance improvements and expectations that are based on laboratory measurements and experiences from customers who participated in the early support program for Db2 12.

This chapter includes the following topics:

1.1 Overview

Db2 12 for z/OS delivers several significant business values such as cost reductions, scalability improvement and ease of data management. These values can be achieved through simple migration and rebind or by taking user actions that are needed to use certain new features. Upgrading to Db2 12 includes the following benefits:

- CPU reductions and scalability improvements
- Simplified and highly available system management
- Ease of application enablement for cloud and mobile environments
- Continuous delivery for speedy adoption of new features

As with older releases, Db2 12 for z/OS provides strong performance improvements that drive great value to Db2 users. Reduced CPU use in Db2 12 can deliver significant savings in total cost of ownership for running applications on Db2 for z/OS.

Most of the Db2 12 CPU reductions and scalability improvements are built directly into the Db2 engine and require no application changes. Optimizer enhancements often take effect after a REBIND in the case of static applications or a PREPARE for dynamic SQL. System level enhancements can require more real memory, which is usually available in the latest IBM z® Systems processors.

In this IBM Redbooks publication, we provide performance information for Db2 12 for z/OS, including the possible performance effects and resource requirements when migrating from Db2 11. This book also includes the latest performance information for IBM Db2 Analytics Accelerator V4, V5, and V6 and IBM z Systems® synergy information.

We use performance measurements that were made in the Silicon Valley Laboratory to illustrate the performance characteristics and usage considerations of Db2 12. The high-level estimates that we provide in this publication are based on these performance measurement results. As with any performance discussions, readers must keep in mind that performance results depend on many variables, including measurement environments, conditions, and workloads that are used. Our measured results provide a point of reference for comparison and for setting reasonable expectations. As always, your results can vary.

1.1.1 Performance

Db2 12 provides many opportunities for reduced CPU use in simple online transaction workloads and in more complex real-time analytics queries. To address the need for importing more data into Db2, major bottlenecks are addressed to improve scalability of applications that are concurrently inserting records into Db2 tables.

As with Db2 11, Db2 12 focuses on business resiliency improvements. By introducing enhancements in areas, such as access path stability, lock avoidance in data sharing, and reducing the cost of using system level CF duplexing to maintain high availability, Db2 12 removes many business resiliency inhibitors to provide consistent performance.

CPU savings

In this publication, CPU savings in Db2 12 are analyzed by comparing measurement results for the same workloads that are run on Db2 12 and Db2 11 New Function Mode.

CPU reduction in online transactions

Most of the system performance improvements are designed to use larger and relatively less expensive real storage, which is now available in the latest System z processors. Db2 online transactional workloads can expect CPU savings ranging from a few percent up to 10%, depending on the workload characteristics and memory allocation. Simple online transactions with large singleton selects through a unique index can see more CPU reduction because of index memory optimizations and other new internal optimizations.

Db2 subsystems that are on z/OS LPARs with large amounts of available memory, or Db2 subsystems that are caching frequently accessed indexes or table spaces can use improved in-memory buffer pool management techniques in Db2 12. With these new features, CPU savings can result from avoiding I/O costs and reducing the overhead of managing buffer pools if the entire Db2 user objects can fit within the buffer pools.

CPU reduction in real-time analytics queries

Db2 12 delivers significant CPU reductions for modern real-time reporting and analytics queries.

Unlike traditional online transactional SQL where only one or a few records are retrieved by using a direct index lookup, many modern applications are increasingly using more complex SQL functions. They tend to involve more join processing through views or history tables, and more sorting is needed to manipulate the data.

Db2 12 specifically targets improving the performance of these types of more complex query workloads, which often require short response times (less than a few seconds) and sometimes require higher concurrency than IDAA can provide.

After PREPARE/REBIND in Db2 12, we observed a wide range of CPU and elapsed time savings. Depending on the workload, we saw a range of savings from a few percent for simple traditional query workloads, all the way up to 50% or more for modern complex query workloads compared to Db2 11. The workloads that benefited most include operations, such as complex outer joins, use of User Defined Functions, and UNION ALL constructs within views or with history tables. These operations have specific optimizations within Db2 12 to improve their performance.

Cost reduction by using more zIIPs

Db2 continues to emphasize the use of zIIPs to reduce the total cost of ownership. Db2 12 increases the zIIP eligibility of child tasks under query CPU parallelism from up to 80% to up to 100%. This ability simplifies capacity management for zIIP processors when estimating zIIP capacity. In addition, the reload phase of the REORG and LOAD utilities is now marked as zIIP eligible.

High resiliency and consistent performance

One critical aspect of performance management is consistency. Unstable access path selection for critical queries can have a significant effect on system level performance. Db2 10 provided static plan stability and reuse capability for access paths (APREUSE) for static packages. Db2 11 provided further enhancements in APREUSE.

Db2 12 takes another major step in our access path stability functionality by providing dynamic statement stability. Dynamic statement stability gives users control over when their dynamic SQL applications are eligible for new access paths and when they are not, similar to static plan stability for static SQL. Among other benefits, this new feature can be used to ensure that consistent access paths after the REORG/RUNSTATS utilities are run by storing the stable access path into Db2 catalog tables.

In addition to access path stability, Db2 12 adds resource limit facility support for static SQL. Users can now use RLF to govern the resource usage of static and dynamic statements with reduced RLF overhead.

In a data sharing environment, one or a few long running units of recovery can cause Db2 lock avoidance techniques to fail. This failure leads to more locks and lock contention, which in turn can cause an increase in CPU consumption. An innovative lock avoidance technique is introduced in Db2 12 to ensure consistent transactional lock avoidance for GBP-dependent objects.

Moreover, with Db2 12, system-managed lock duplexing can be accomplished with less system affect by using asynchronous lock duplexing, which is available in system IBM z13® (CFCC 21). With this feature enabled, the communication between primary and secondary lock structures is done asynchronously for Db2 lock requests, and the efficiency of duplexed lock requests is greatly improved. As a result, the overhead of lock duplexing to achieve high resiliency is reduced in two CEC configurations.

Scalability improvement

To address the demand for high volume ingestions from large numbers of concurrent clients, Db2 12 introduces a new insert algorithm for table spaces that do not require clustering (do not require APPEND YES). The new insert algorithm avoids table space search contention effectively when many concurrent users insert at the end of table spaces or into the same partitions.

The new insert algorithm and another scalability improvement in log writes made it possible during an IBM-internal benchmark to reach 11.7 million inserts per second for a sustainable period in a 12-way data sharing configuration.

1.1.2 High-level expectations

The degree of CPU savings that are realized when you are migrating to Db2 12 depends on a specific workload’s characteristics.

Most of the query improvements are available under co-existence mode (Function Level 100) but do require users to perform a REBIND on Db2 12. Regardless of whether you plan to take advantage of Db2 12’s access path changes, a REBIND should be planned at some point after the migration to Db2 12. This process reactivates select procedure optimizations and avoid the internal structure conversion overhead that is incurred when packages that are bound in Db2 11 on Db2 12 are run.

Simple online transactions with many random primary key index accesses can benefit from index memory optimizations in Db2 12 starting in co-existence mode (Function Level 100) in a non-data sharing or single member data sharing environment. Activation of Function Level 500 or greater is required to get the full benefit of these optimizations in a data sharing environment with more than one Db2 member.

Insert-intensive applications against table spaces that are defined with MEMBER CLUSTER can see a throughput improvement and a CPU reduction by avoiding contention through a new insert algorithm. This feature requires Function Level 500 or greater.

The range of performance improvements that are expected in Db2 12 Function Level 500 when compared to Db2 11 running in New Function Mode is shown in Figure 1-1. The results, averaged by workload type, as evaluated by using data from multiple sets of measurements also are shown in Figure 1-1.

Compared to the same workload running on Db2 11, you might see the following improvements:

- A CPU reduction of 0% - 6% for online transaction processing (OLTP) workloads that are not using index in-memory optimizations.
- A CPU reduction of 0% - 10% for online transaction processing (OLTP) workloads that use index in-memory optimizations.
- A CPU reduction of 0% - 12% in concurrent insert workloads by using MEMBER CLUSTER.
- A CPU reduction of 0% - 50% in queries after REBIND.

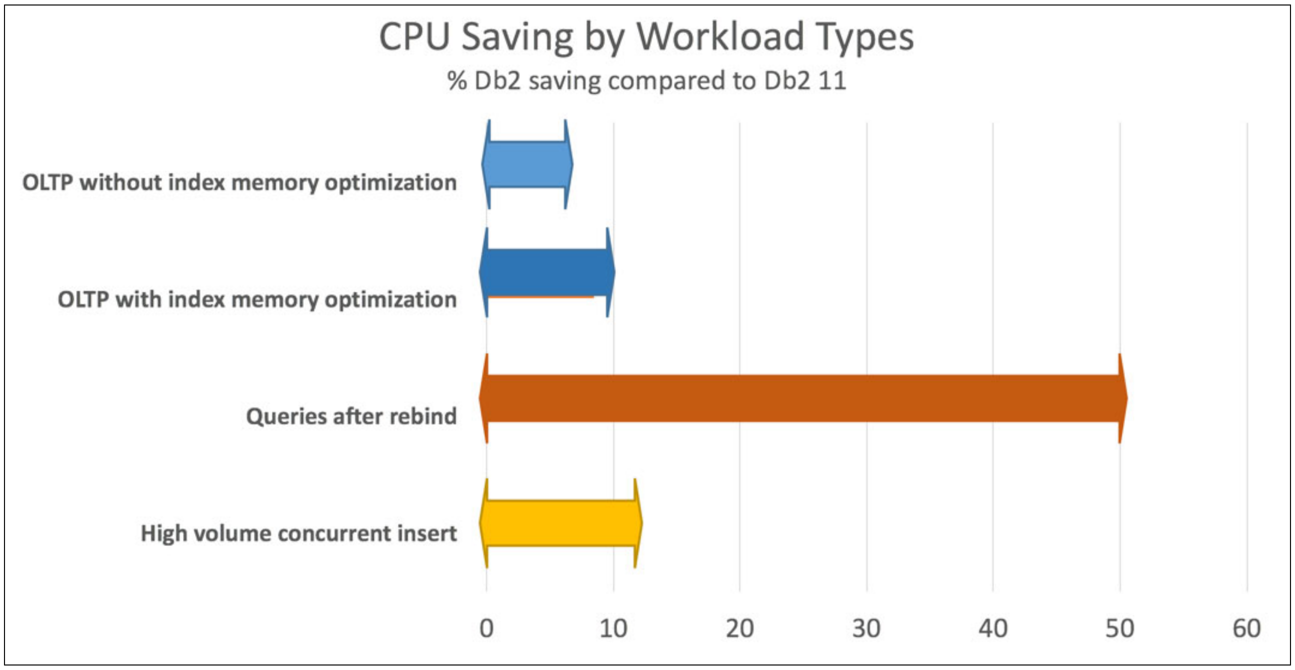


Figure 1-1 CPU savings by workload types

1.2 Db2 12 for z/OS performance features

In this section, we describe key performance improvements in Db2 12. We also describe performance measurement data that was obtained by the Db2 performance team by using public benchmark workloads and customer workloads.

1.2.1 System-level performance

The maximum amount of memory that can be installed in an IBM z Systems® server increased significantly with the advent of the z13 in 2015, as shown in Figure 1-2. Today, a fully configured z13 server can have up to 10 TB of memory installed, with each logical partition (LPAR) allocating up to 4 TB of memory.

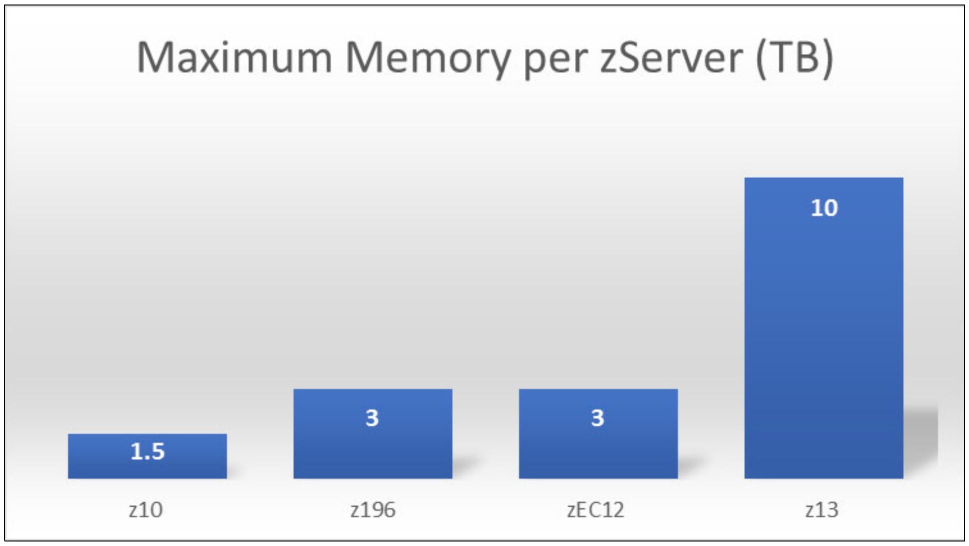


Figure 1-2 System z Maximum Memory per server

At the same time, the cost per GB for mainframe real storage dropped sharply in recent iterations of IBM's z Systems servers, particularly with the latest IBM z13.

Because of these changes, today's typical z Systems server configurations feature significantly more memory than before. With easier and cheaper access to large amounts of memory, many new Db2 12 features take advantage of in-memory optimizations by caching more data or control information in memory.

Index in memory optimization (Fast Traverse Blocks)

The fastest way to access a record in a table is to use direct index lookups. Many Online Transaction Processing (OLTP) applications make extensive use of direct index lookups, where the application provides a fully qualified key that uniquely identifies a single row in the target table. The access pattern for such key-based lookups is often random, as these keys are usually based on customer-id, sales-record, or card number.

As the volume of data grows, the size and levels of the index grows, as does the cost of traversing the index tree structure. Index traversal here means accessing the index from top down, from the root, to non-leaf pages, and then to the leaf page, as shown in Figure 1-3 on page 7.

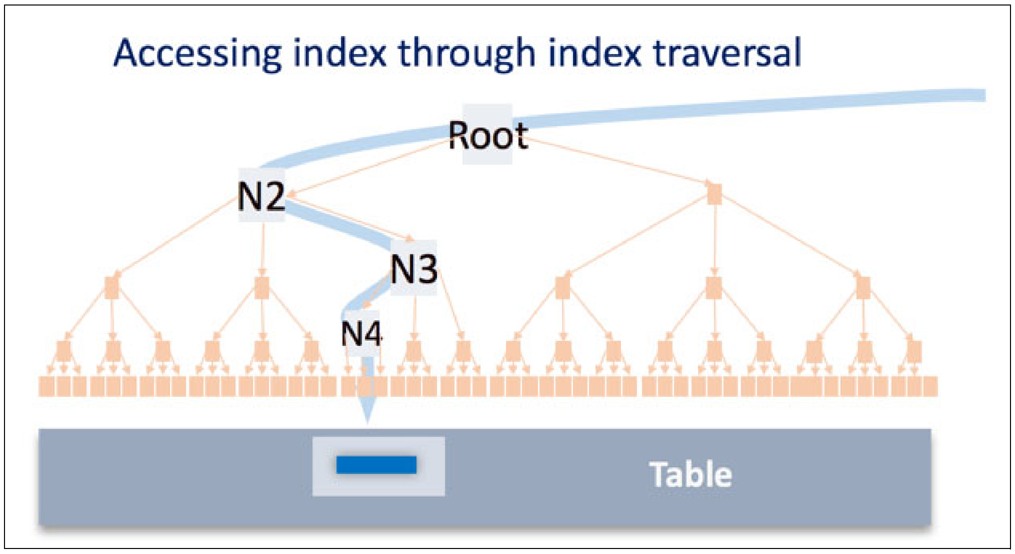
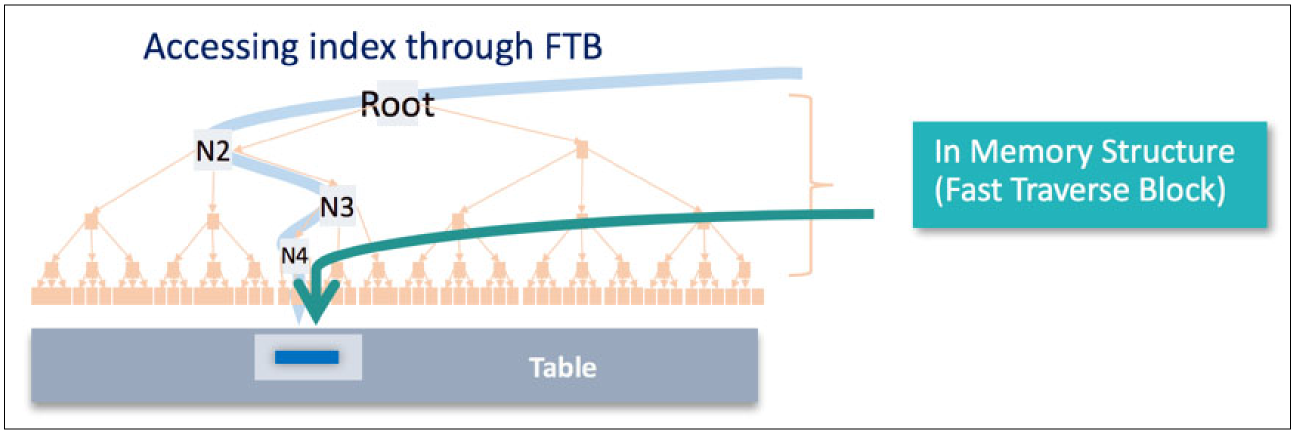


Figure 1-3 Index traversal access

Db2 10 for z/OS introduced hash access capability to store tables into hash-based table spaces and eliminated the need for indexes, which works well if the access to the table is always random. However, the tables are often accessed randomly by using unique keys and sequentially through batch accesses. The performance affects on sequential access with hash-based table spaces can negate the performance benefit for random access.

Db2 12 for z/OS takes a new approach by storing index information into a memory structure to reduce the index traversal cost for random index access. This memory structure, internally called Fast Traversal Blocks (FTBs), holds the root and non-leaf pages of the indexes. The Fast Traversal Block structure is designed to be processor cache friendly for traversing index keys and the index tree traversal operations can be bypassed, as shown in Figure 1-4. When the index is sequentially accessed, its leaf pages (in the buffer pools or on disk) can be accessed as before.



Accessing index through FTB
Root
N2
N3
N4
In Memory Structure (Fast Traverse Block)
Table

Figure 1-4 Index traverse with FTB

Db2 monitors the use of indexes and automatically builds a Fast Traverse Block for an index with many random traversal accesses. It is also avoiding building FTBs or effectively removing FTBs if the indexes are primarily used for sequential access, or if the indexes frequently go through index structure modifications.

A new system parameter INDEX_MEMORY_CONTROL is introduced to control FTB usage at the subsystem level so that the user can disable the feature completely, or enable it but define the upper limit of memory usage for FTBs. The default value is AUTO, which enables index in-memory optimization and allows storage of up to 20% of the sum of allocated buffer pools to be used for FTBs. Similar to the index pseudo-delete clean-up process, a new Db2 catalog table, SYSIBM.SYSINDEXCONTROL, is added for users to influence the behavior at the individual index level.

A few IFCIDs and a Db2 display command are provided to monitor FTB behavior, as described in Chapter 12, “Monitoring and instrumentation” on page 341.

Performance improvement with index in-memory optimization depends on the access pattern, depth of index level (determined by key size and numbers of index entries), and commit frequency. When we ran a loop of singleton select statements through batch jobs with infrequent commits, we observed up to a 20% reduction in CPU time. The measured Db2 CPU time savings by using indexes with different numbers of index levels are shown in Figure 1-5. For example, when FTBs are used on a 5-level index, the number of index getpages per statement was reduced from 5 getpages to 1.

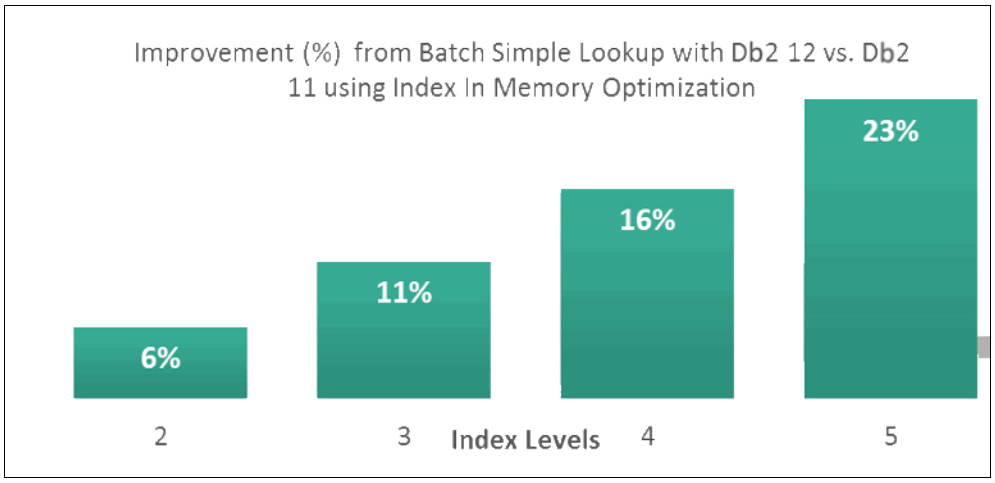


Figure 1-5 Index in memory optimization in batch select with different index level

The performance improvements with index in-memory optimization in Online Transactional Processing workloads where commits are issued more frequently demonstrate a more realistic usage. As shown in Figure 1-6 on page 9, the index in-memory optimization results in a few other percent in overall Db2 12 CPU improvement.

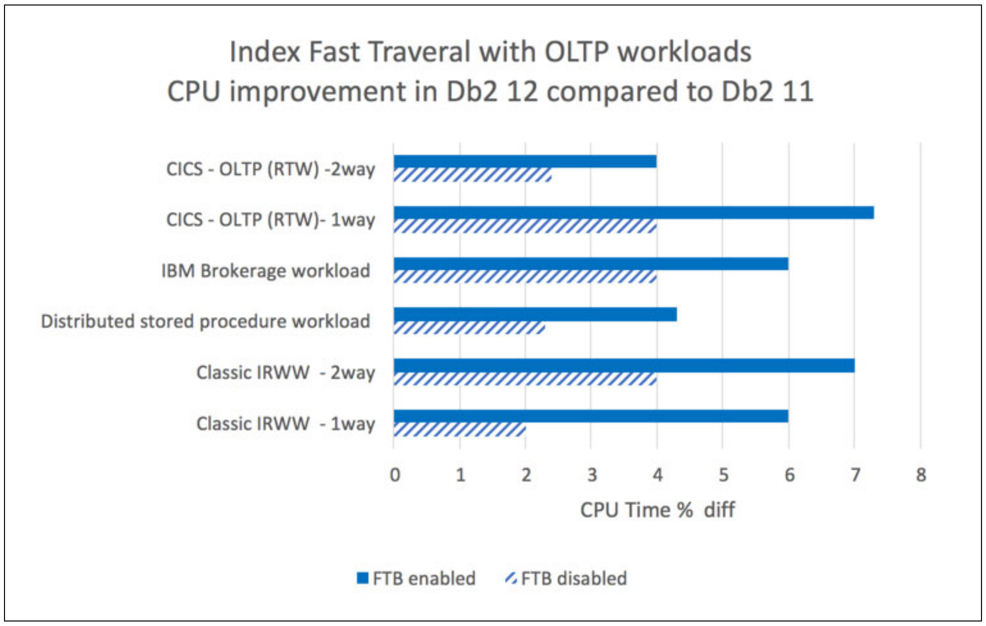


Figure 1-6 Db2 12 improvement over Db2 11 with and without using index in-memory optimization

Contiguous buffer pools

Db2 12 adds another key performance feature that is called contiguous buffer pools, which use the larger memory in System z servers. This feature is an enhancement to the PGSTEAL(NONE) feature that was introduced in Db2 10.

Db2 uses least recently used (LRU) as the default page steal option for buffer pools. With PGSTEAL(LRU), Db2 manages the pages in the buffer pools to keep the frequently used pages in the buffers and steal the pages that are least recently used when new pages must be brought into the buffer pools from disk. This buffer pool management process to determine which pages can stay longer in a buffer pool was important when the buffer pool size was limited to less than 2 GB because of the 31-bit memory constraint. Even when the buffer pools were moved above the 2 GB bar in Db2 V8, this buffer pool management process was still critical because, at that time, the real storage available in the system was still limited.

With large enough memory available with later z processors to fit mission critical indexes and tables in the buffer pools, Db2 introduced PGSTEAL(NONE) to be used for such buffer pools to ensure that important objects do not suffer disk I/O waits. With PGSTEAL(NONE), Db2 prefetches the entire page set or partition into the buffer pools at first access to the page set so that subsequent accesses to the page set do not require I/O waits. However, page management is still needed with PGSTEAL(NONE) buffer pools in Db2 10 and 11, as the pages are laid out in random order in the buffer pools.

When the PGSTEAL(NONE) attribute is used in Db2 12, the buffer pool control blocks that are used to manage buffers are physically laid out in memory in the same order that the page sets are on disk. Because this configuration is used, buffer pools with this attribute are also known as *contiguous buffer pools*. Contiguous buffer pools allow for Db2 to skip pool management processes while accessing the pages in the buffer pools. The larger the buffer pools are, the bigger the cost of these page management processes as LRU and hash chain management suffers more processor cache misses.

In one of our OLTP workloads that simulates online brokerage transactions, the use of PGSTEAL(NONE) as opposed to PGSTEAL(LRU) demonstrated an 8% CPU improvement.

The CPU improvement is not from I/O reductions in this case, as I/Os were avoided with PGSTEAL(LRU) pools because the pool size is large enough.

The CPU reduction is attributed to fewer processor cache misses while getting and releasing pages. The degree of improvement depends on the ratio of getpages from the buffer pools with PGSTEAL(NONE) versus getpages from PGSTEAL(LRU) buffers. In this workload, a total of 70 GB of buffer pools are allocated with 10 GB of buffer pools specified to use PGSTEAL(NONE). However, because more than 80% of getpages are from the buffer pools with the PGSTEAL(NONE) attribute, the CPU savings is high.

Db2 12 PGSTEAL(NONE) is recommended for objects that are accessed frequently with no growth, or with a stable growth rate. It is recommended for frequently accessed objects because better improvement can be expected for such objects (as opposed to objects infrequently accessed), if the objects fit in the buffer pools.

If PGSTEAL(NONE) is used and the page sets do not fit in the contiguous buffer pool, Db2 uses overflow buffers, which are unique to Db2 12 PGSTEAL(NONE) pools, for the pages that do not fit in the contiguous buffer pool. Up to 6400 buffers are set aside by Db2 for the overflow area for each buffer pool with PGSTEAL(NONE) attribute to keep reasonable performance, even in the case of unexpected growth.

Db2 messages and statistics are updated to monitor the usage of the overflow area. To identify candidate objects for contiguous buffer pools, a new column “GETPAGE” is added in Db2 real-time statistics (SYSIBM.SYSTABLESPACESTATS, and SYSIBM.SYSINDEXSPACESTATS). Generally, the higher the number of getpages is, the greater the benefit from using contiguous buffer pools for the object. For more information about choosing candidate objects, see 2.2.3, “Managing a contiguous buffer pool” on page 42.

In addition to contiguous buffer pools, Db2 12 extended the maximum size of buffer pools from 1 TB to 16 TB. Although system z13 supports up to 10 TB, z/OS supports only up to 4 TB of memory per LPAR as of this writing. With this z/OS memory limitation, the realistic maximum Db2 buffer pool size today is less than 4 TB.

Buffer pool simulation

Having larger buffer pools could improve the elapsed and CPU time of applications with large sync I/O waits. However, the actual improvement is difficult to estimate in complex workloads because the benefit from larger buffer pools depends on the size of the active working set of data and the access pattern.

For example, if you have a workload with mostly random access to the data that is repeatedly referenced, increasing the buffer pool likely is beneficial because there is a good chance that the same pages are being referenced multiple times. However, if your application reads and processes only the data sequentially once and does not reference the pages again for a long time, having a large buffer pool might not help much.

The buffer pool simulation feature in Db2 12 helps users to estimate the benefit of having larger buffer pools without allocating large real storage. This feature introduces the concept of simulation pools where you can specify the extra pool size you want to run during the simulation. With simulation pool size (SPSIZE) defined, Db2 allocates the buffer pool control blocks to cover simulation pools and starts simulation by using the LRU page steal algorithm as though there are other pools allocated. By using the LRU information in the control blocks for the simulated pools, Db2 can determine whether the pages that are missed in the actual virtual buffer pool are in the simulated pools.

Because Db2 applies the actual LRU algorithm, the simulation is accurate if the workload is consistent. The results from the simulation are collected in IFCID 2 buffer pool statistics and can be displayed by using the **`-DISPLAY BUFFERPOOL`** command. The information in the buffer pool statistics includes avoidable synchronous I/Os, asynchronous I/Os, avoidable GBP accesses, and elapsed time savings.

The simulation requires more memory to accommodate the control blocks that are allocated for the simulated buffers. The memory allocation is approximately 2% of the simulated size for 4 K page size buffer pools. Our measurements indicate that there is approximately 1% CPU overhead when the simulation is run with many buffer pool misses. (The feature is retrofitted to Db2 11 by using APAR PI22091.)

Other memory optimizations

In addition to the optimizations that we described, Db2 12 introduces various internal optimizations to reduce CPU usage by using more in-memory processing and optimizing memory management. These internal optimizations are described next.

EDM pools

EDM pool management is updated to avoid micro managing storage usage. It also reduces the scalability inhibitors, such as latch contentions when many statements are cached in the dynamic statement cache or when many packages are held in the skeleton pools.

UDF result sets

User Defined Functions, which are defined as DETERMINISTIC and NO EXTERNAL ACTION, can use result set caching within the scope of the UDF. This optimization is an effective use of memory because many UDF calls can be avoided by using result set caching.

DGTT

Db2 11 for z/OS added a significant performance improvement in the use of Declared Global Temporal Tables (DGTTs) by providing a NOT LOGGED option. It is especially effective when there are many INSERT, UPDATE, or DELETE operations against the DGTT.

Db2 12 further improves DGTT performance at declaration time. When a DGTT is declared, Db2 creates a virtual catalog within the DGTT. Db2 12 caches the virtual catalog information in-memory to speed up the declaration process. The benefits of this enhancement are noticeable in applications that frequently create DGTTs, but insert only small numbers of records before dropping the DGTT.

RLF

Db2 12 for z/OS adds Resource Limit Facility support for static SQL in addition to the existing support for dynamic SQL. When RLF is enabled, Db2 is required to check the resource limit tables for every SQL access. When the tables are small, this cost is trivial. However, the overhead becomes noticeable when the resource limit tables contain many rows. Db2 12 caches the contents of resource limit tables in memory to minimize the overhead of RLF lookups. With this optimization, Db2 12 can support RLF for static statements without noticeable overhead, and provide improvement in the existing RLF process for dynamic statements.

RDS sort

Db2 12 continues to improve in-memory RDS sort processing. It increases the number of cases when Db2 can avoid writing intermediate sort results to work files so the results can be fetched directly from memory instead of from work files. Db2 12 also expands the maximum number of nodes in a sort tree, from 32,000 to 512,000 for non-parallel sorts or 128,000 for parallel sorts under child tasks. These enhancements might require more memory to be allocated to the thread for sort activities, but can result in a significant CPU reduction.

High volume data ingestion

With the prevalence of mobile computing, the demand for high volume data ingestion continues to grow. Db2 12 addresses this demand through the new enhancements that are described next.

Insert Algorithm 2

Typical examples that require a high insert rate can be found in application audit or journaling applications, in which many concurrent transactions must insert one or more rows into a table.

Traditionally, the MEMBER CLUSTER option is used for such applications to speed up the table space search processing. The MEMBER CLUSTER option provides good performance by assigning a space map and its associated data pages for a specific member's private use, ignoring the data clustering defined through the clustering index. However, as data sharing member consolidation continues, the number of concurrent clients from the same member increases and so does the contention on the space map and data pages at the end of the table.

Db2 12 introduces a new insert algorithm (Insert Algorithm 2). This algorithm is used by default for Universal Table Spaces defined with the MEMBER CLUSTER option after the activation of Function Level 500 or greater. With Insert Algorithm 2, each thread is assigned a separate page to insert into, which eliminates data page contention. It uses a background system task to format the new pages before the user thread requests a page. The information about the candidate pages is stored in a new in-memory structure, which is called the "insert pipe". The old method of insert space searching instead consults a space map page, which was the major bottleneck in the old insert algorithm.

With the new insert algorithm, a significant throughput increase in concurrent insert applications can be expected when there was a bottleneck because of insert space searching. In IBM measurements that use 200 or more threads, we saw class 2 elapsed time savings of up to 80% that used Db2 12 for cases with insert space search bottlenecks in Db2 11. However, if the current insert bottleneck is somewhere else, such as log write I/O performance or index split processing, little to no improvement from Insert Algorithm 2 is realized.

Also, Algorithm 2 might push the bottleneck from Db2 to somewhere else. One of our Java applications showed significant improvement in Db2 processing. However, it pushed the bottleneck to the network and the benefit of the new insert algorithm did not translate to a throughput improvement until we eliminated the network bottleneck. Careful examination of accounting reports is recommended to understand the effect of Insert Algorithm 2.

Active log improvement

For high volume insert or update applications, Db2 active log writes tend to become a bottleneck. One of the contributing factors to log write bottlenecks is the serialization processing that is required when writing the logical logs, which is known as *log latch contention* (latch class 19). Db2 10 introduced some relief to log latch contention and Db2 12 virtually eliminates the bottleneck that is associated with log latch contention when many concurrent update transactions are run.

With intensive update applications, the frequent switching of Db2 active log data sets can occur even when 4 GB active log data sets are used. The frequent log switching causes a performance effect and a possible delay in recovery because the log read can occur from archive logs. Db2 12 increases the maximum size of active log data sets from 4 GB to 768 GB, which helps to reduce the frequency of log switches and their effect.

11.7 million inserts per second with Db2 12

With these scalability improvements, the Db2 performance team could measure up to 11.7 million inserts per second into a single Db2 table, by using a 12-way data sharing group. The active log write rate reached 2 GB per second. For more information about the benchmark configuration that was used, see 5.1, "High volume INSERT benchmark " on page 156.

DRDA Fast Load

Many Db2 customers expressed the need for faster data load into Db2 for z/OS from distributed clients. One example is SAP applications, where the source data is on the SAP application server on UNIX platforms. Db2 12 introduces a new DRDA fast load feature that enables the customers to start the LOAD utility directly from the client. This feature supports APIs, including Db2 call level interface (CLI), command line processor (CLP), and Java applications.

In IBM measurements, DRDA fast load can load the data faster than the combined elapsed time of file transfer (FTP) and running the LOAD utility. This feature requires IBM data server driver updates.

Data sharing improvement

Data sharing configuration continues to be a key to high availability but incurs more overhead to maintain the data integrity across the data sharing group. Db2 12 features a major improvement to reduce the overhead by improving lock avoidance, even with long-running URs and page registrations from utilities.

Improved lock avoidance

Before Db2 12, a single long-running unit of recovery (UR) against a GBP-dependent object caused an increase in locks and contentions for non-related applications in all the data sharing members. This increase occurred because Db2 maintains a single commit LRSN for a data sharing group. Db2 uses the commit LRSN to evaluate the possibility of lock avoidance for all GBP-dependent objects for all data sharing members. If lock avoidance does not work, more select locks can occur, which can increase CPU time and possibly contentions.

Because of their possible effects, Db2 provides warning messages for those long-running URs to notify users when they occur. Although the best thing to do with these long-running URs is to change the applications to commit frequently, it is not always easy for customers to update the applications in a short time.

Db2 12 maintains more granular global commit LRSNs for long-running URs in the SCA structure. For the objects that are not targets for the long-running URs, Db2 can perform better lock avoidance and the critical transactional effect from infrequently committing applications can be reduced.

Page registration by Db2 utilities

UNLOAD and RUNSTATS utilities that sequentially read many pages can result in a large volume of register page list operations to the coupling facilities if the objects are GBP-dependent. The Register Page List (RPL) operation is done to register a set of pages in a GBP structure.

Although the list operation is much more efficient than doing multiple single page registrations, it is an expensive and long-running CF command. If large numbers of RPLs are requested at the same time, it can cause high CF CPU utilization and possibly affect other structures' response time.

To avoid CF-wide effect by RPL operations, Db2 12 adds a REGISTER option to UNLOAD and RUNSTATS utilities to avoid page registration if the utilities are running with ISOLATION UR. The new keyword REGISTER YES/NO is available after the migration of Db2 12 (Function Level 100). The default REGISTER behavior for the UNLOAD utility is NO; the default for the RUNSTATS utility is YES.

System z Synergy

Db2 continues to work closely with the System z hardware and software teams for Db2 to take better advantage of new System z technologies. These collaborations resulted in several new features in Db2 12, which are described next.

Asynchronous CF lock structure duplexing

The asynchronous CF duplexing feature became available with IBM System z13 with CFCC 21 level starting October 2016. This feature reduces the latency of XES lock requests in system-managed duplexing compared to system duplexing managed synchronously. Db2 12 and IRLM for Db2 12 use the feature to reduce the latency and cost of lock structure duplexing. The benefit of using this feature varies depending on configuration, such as CF distance, CF link types, and commit frequency of the applications. Preliminary IBM measurements demonstrate promising results of up to 2 times cost reduction by using asynchronous lock duplexing, compared to today's synchronous duplexing.

LOB compression with zEDC

IBM System zEnterprise® Data Compression Express (zEDC Express) is available in system zEC12 and later processors and offers a compression acceleration that is designed for high-performance, low-latency compression with little extra overhead. z/OS supports compression of SMF records and BSAM/QSAM files through zEDC.

Db2 12 uses zEDC Express to compress the LOB data type. The compression is available for decompressed LOB data and LOB data in a Db2 directory. When compressed well, this feature provides disk usage savings and application throughput improvement for processing large objects.

zHyperWrite

zHyperWrite can speed up Db2 log write time in a configuration where synchronous disk replication, such as IBM Metro Mirror, is used. This feature requires DS8K storage subsystems with IBM GDPS® HyperSwap and z/OS DFSMS changes. When enabled, Db2 can request a log write to the secondary location at the same time as the primary location instead of waiting for the disk subsystem to do the replication.

This feature reduces the wait time for log write or update commit time. Evaluations by customers and the IBM lab show up to 40% reduction in log write wait time. The feature was developed in Db2 12 and is available as a retrofit by using APARs in Db2 10 and Db2 11.

OLTP workload measurements

IBM measurements from various online transactions mostly show the range of a few percent to up to 10% of Db2 CPU reduction by using Db2 12 compared to Db2 11 benefiting from the features shown in Figure 1-7. For more information about the measurements and workload, see Chapter 5, “Workload level measurements” on page 155.

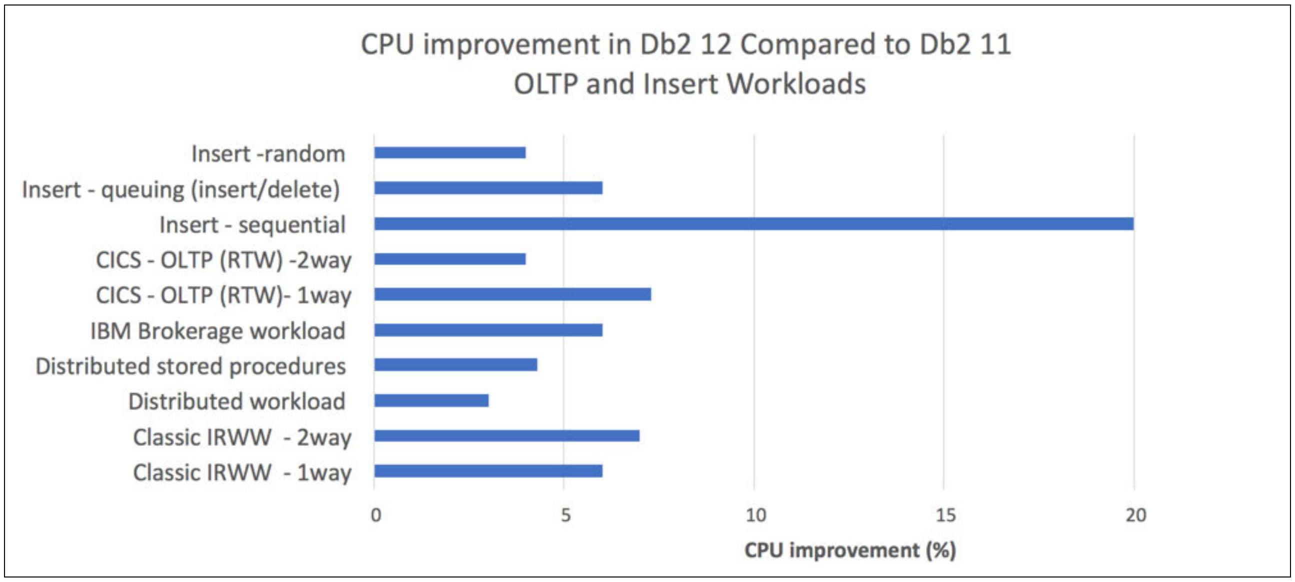


Figure 1-7 OLTP and Insert workload improvement in Db2 12 compared to Db2 11

1.2.2 Query performance improvement

Db2 12 introduces optimizations that specifically target modern applications that use UNION ALL processing through views or with history tables. Db2 12 also includes enhancements for applications with large sorts and complex queries that require a short response time of less than a few seconds, or with higher concurrent execution than IBM Db2 Analytics Accelerator for z/OS (IDAA) can provide. Significant CPU savings are observed in IBM measurements, as shown in Figure 1-8.

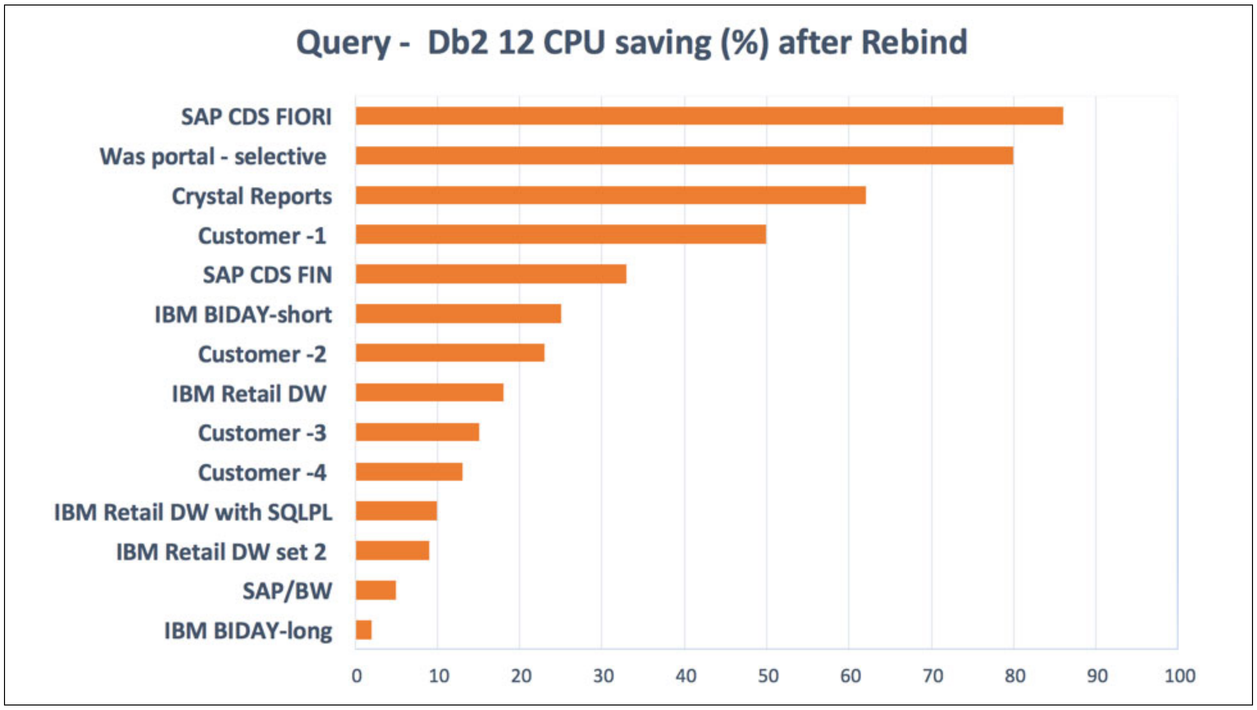


Figure 1-8 Db2 12 query workload CPU saving after REBIND compared to Db2 11 NFM

This comparison is done after REBIND in Db2 12. Most query optimizations are available at the point of Db2 12 migration and after REBIND in Db2 12, and do not require users to activate Function Level 500.

For more information about query workloads and other enhancements, see Chapter 8, “Query performance” on page 243.

UNION ALL and outer join enhancements

UNION ALL (UA) constructs combined with outer joins and nested views are more commonly used in modern SQL applications, and in Db2 features that use history tables, such as Bi-temporal tables and transparent archiving features. One of the challenges of such queries is the lack of predicate filtering, which often results in excessive work file materialization with large sorts.

Db2 12 features the following optimizations:

- Reordering OUTER JOIN tables to avoid materializations
- Pushing predicates inside UNION ALL legs or OUTER JOIN query blocks
- Sorting of outer table to ensure sequential access to the inner table of a join
- Bypassing work file usage when materialization is required
- Trimming unnecessary columns and tables
- Push ORDER BY and FETCH FIRST into UNION ALL legs

With these improvements combined, IBM tests show up to 80% CPU and elapsed time reductions for the candidate queries. Selective queries in the WAS portal workload that are shown in Figure 1-8 on page 16 include queries that use UNION ALL with views. Other query workloads, such as Customer 3 and SAP CDS FIORI workload, also benefit from these improvements.

Runtime Adaptive Index Access

Many modern applications provide generic predicates so that the user can enter the specific values for the predicates at execution time. A good example is a query with multiple LIKE predicates that use host variables or parameter markers, as shown in the following example:

```
SELECT first_name, last_name, salary FROM employees
WHERE last_name LIKE ?  AND zipcode LIKE ?  AND year_of_birth LIKE ?
AND city LIKE ? ;
```

Some predicates might be indexed well, but others might not be indexed as well. The challenge is that the predicate filtering is unknown at the point of query optimization. Db2 12 addresses this challenge by using a runtime adaptive index feature, which allows the Db2 query optimizer to defer some of the important index selectivity choices until query execution. At run time, multiple indexes are probed and the indexes with good filtering can be picked.

Runtime adaptive index support provides a stable access path for queries where the access path selection is unclear.

RDS Sort Improvements

In addition to in-memory optimizations, Db2 12 also provides the following sort enhancements:

- Sort minimization with FETCH FIRST
- Reduction in sort key size for GROUP BY and DISTINCT
- Work file process improvement by using 32 KB page work files for intermediate sorts
- Support for sparse index for the VARGRAPHIC data type
- Avoid sorting for pre-ordered chunks of data

The sort-intensive query workloads, such as Customer 2 and SAP SFIN in Figure 1-8 on page 16, showed significant benefit from the sort improvements.

1.2.3 Access path stability for dynamic SQL statements

For static SQL statements, the access path can be fixed at BIND time and the path can be stabilized through access path stability and access path reuse features that were introduced in Db2 10. Although dynamic SQL statements are dominant in Java applications, they pose challenges in achieving consistent performance because of possible access path changes.

The access path of dynamic SQL statements is determined at execution time and can be vulnerable to change. For example, the access path can change at the next full prepare. This change can occur based on various factors, such as updates in catalog statistics through RUNSTATS execution, Db2 configuration changes (such as buffer pool or sort pool size changes), or a Db2 maintenance upgrade. The dynamic statement cache can have a stabilizing effect, but the cache alone is still a volatile solution.

Db2 12 introduces a new dynamic SQL stability feature to improve the performance and manageability of dynamic SQL. This feature provides the capability to stabilize SQL statements in the dynamic statement cache by storing the information in a new set of Db2 catalog tables.

After the statements are stabilized, the stabilized access path can be retrieved and used from the catalog tables at full prepare when Db2 does not find the statements in the dynamic statement cache. Db2 can provide consistent performance with a stabilized dynamic SQL statement. Performance improvement for the dynamic prepare is also possible when the statements are found in the catalog instead of having to go through a full prepare.

This feature is expected to be useful during workload transitions, configuration changes, and when applying future Db2 maintenance upgrades.

Dynamic statement stability requires the activation of Function Level 500 or greater and provides various monitoring and statistics changes in prepare operations.

1.3 Other key features in Db2 12 for z/OS

The key improvements and features that are delivered by Db2 12 for z/OS in addition to performance improvements are described in this section. For more information about Db2 12 features, see *IBM DB2 12 for z/OS Technical Overview*, SG24-8383. Db2 12 for z/OS also includes the following other key features:

- Mobile application enablement
- Resiliency and continuous availability
- Simplified migration and continuous delivery support

1.3.1 Mobile application enablement

A large set of new SQL capabilities are introduced or enhanced in Db2 12 to encourage the application development and porting in Db2 for z/OS.

RESTful API

Db2 Adaptor for z/OS Connect enablement, introduced in Db2 11, made it faster and easier to deploy RESTful services that access Db2 data. However, this solution requires the installation and setup of z/OS Connect services.

To speed up application deployment, native REST support is now built into Db2. With native REST API support, Db2 DDF address space provides a set of services to replace the Db2 Adaptor and z/OS Connect features. JSON is used to pass input and return the results back to the requester. Native REST API support is available in Db2 12 and Db2 11 (with APAR PI66828).

Application enablement

Db2 12 expands SQL capability to continue to support and attract today’s application development.

Enhancements in application enablement include advanced triggers, which allow developers to create the logic within a trigger. An enhanced MERGE statement is now aligned with the behavior that is defined in the SQL standard and Db2 family, and now allows you to delete from the target tables. Db2 12 provides SQL propagation features that support both numeric and data dependent pagination.

Piece-wise data modification support allows the developer to specify a FETCH FIRST clause for a DELETE statement and issue a COMMIT. This ability addresses many possible concerns, such as lock contention, long-running UR, and CF effect or recovery.

Db2 provides JSON data store in BLOB columns in Db2 11. Db2 12 adds enhancements by allowing the input value to be derived by other means, such as a CASE expression, view, or table expression.

The temporal feature was introduced in Db2 10, and Db2 continues to enhance this feature based on customer requirements. Db2 12 adds the following improvements:

- Referential integrity support
- Auditing
- Logical transaction support
- Temporal inclusive/exclusive model

Cloud ready

As a first step to be cloud ready, Db2 11 and 12 delivered an enhanced installation process to support z/OS Management Facility (z/OSMF). Db2 provides a series of workflow scripts that allow customers to deploy Db2 subsystems quickly on demand. REST API calls are used to start z/OSMF scripts, which allows developers to make the requests easily.

1.3.2 Resiliency and continuous availability

One of the strengths of the System z platform is high resiliency and availability to support continuous availability. This section describes key enhancements in Db2 12 that provide ease of database management and high resiliency.

Partitioned table space enhancements

Universal table spaces are enhanced further to adapt the data growth quickly in Db2 12.

PBR Relative Page Numbers

To address ever increasing data volumes and requirement for easy partition management, Db2 12 introduces a new type of partition by range table spaces (PBRs).

By expanding the internal record identifier (RID) from 5 bytes to 7 bytes, the page number becomes relative to the partition instead of being an absolute number in the entire table space. Therefore, this new type of table space is called PBR Relative Page Numbers (RPN). PBR RPN supports larger partition sizes (DSSIZE) up to 1 TB per partition. More importantly, it supports different maximum partition sizes for a single table, which allows flexibility in managing the partitions.

Insert partitions

It was relatively easy to add a partition at the end of a partitioned table space, but inserting a partition in the middle of the table space when one of the partitions grows is not as easy. Although PBR RPN allows different sizes of partitions and relieves the need of inserting partitions in some cases, Db2 12 also can insert partitions through online schema without affecting applications. This ability still requires a subsequent REORG to allow the data pages to be redistributed, but the user must run only the REORG on the affected partitions.

Utility enhancements

Numerous utility enhancements were added to Db2 12 to keep the data available to Db2 applications and improve the utility execution efficiency.

The use of the REORG utility against one or a few partitions in Partition By Growth (PBG) table spaces was challenging because the data might not fit in the same numbers of partitions after REORG because of REORG honoring free page definitions. Db2 12 solves this challenge by adding the capability for REORG to create a partition to complete the REORG successfully.

Db2 12 removes the restriction that is related to running parallel LOAD from a single input file for PBG with the RESUME YES and SHRLEVEL CHANGE options. In our measurements, we observed up to 76% elapsed time savings by using parallel LOAD with these RESUME YES and SHRLEVEL CHANGE options.

Security

Db2 12 introduces a new TRANSFER OWNERSHIP SQL statement that can replace the original owner of a Db2 object with a new owner ID.

The UNLOAD privilege is now separated from the SELECT privilege on the table so that the customer can run UNLOAD without automatically having the SELECT privilege.

1.3.3 Simplified migration and continuous delivery support

Within a Cloud and DevOps environment, it is essential for Db2 to deliver new features faster and with high quality so that application developers can take advantage of the new features without waiting for the next release. Db2 is embarking on the journey of continuous delivery, taking Db2 12 as a starting point. The first step is to simplify Db2 12 migration, and then to build the foundation of the continuous delivery model.

Single phase migration

Based on requirements from customers, Db2 12 simplifies the migration process to have one catalog update instead of a two-step migration. Unlike prior releases, the catalog updates for Db2 12 are done at the point of CATMAINT, and the activation of new functionality is done by using a simple Db2 command when there is no longer a need for fallback or co-existing configurations. This ability simplifies the migration step and improves migration performance.

Foundation of continuous delivery

As a foundation of continuous delivery, Db2 12 introduces Function Levels (FL), which describe a set of new features that are bundled.

At the point of Db2 12 migration from Db2 11, the FL is V12R1M100. This configuration resembles conversion mode (CM) in prior Db2 releases, where Db2 allows fallback and co-existence between Db2 11 and Db2 12 members in a data sharing group.

FL V12R1M500 identifies the starting level of new Db2 12 functionality, where all Db2 12 features from the general availability code level are enabled. Application packages are still controlled by the APPLCOMPAT option to enable the new functions.

Db2 plans to ship new FLs, such as FL as a part of the maintenance stream instead of providing new versions or releases. However, there is no requirement to activate the functionality at the point of installing the maintenance. That is, the maintenance level can move forward without activating the new FL. When comfortable with the new maintenance level, the customer can choose to activate a certain FL by using the `ACTIVATE FUNCTION LEVEL` command.

1.4 How to use this book

This IBM Redbooks publication is a technical report of detailed performance measurements for most of performance-related topics for Db2 12 for z/OS. The book also includes the latest performance information about IBM Db2 Analytics Accelerator, and System z hardware information that was used by Db2 for z/OS.

The book describes the use considerations that were observed through these measurements. The measurements are meant to show how the different enhancements perform, and the topics are not limited to performance enhancements only. They can be about new functionalities, and in this case, they are compared to similar functions in previous Db2 versions or to the behavior without the function. We also provide an understanding of the new functions, and to explain what the reasonable expectations are for these new functions.

All measurements were conducted in a laboratory environment, and as such, they might be atypical because they were run in a dedicated and controlled environment with no other workloads causing resource contentions, and in some cases focusing on specific functions. Extrapolation and generalization require judgment.

This book does not cover all the features of Db2 12. For general Db2 12 information, see *DB2 12 Technical Overview*, SG24-8383.

This book includes the following chapters:

- Chapter 1, "Introduction " on page 1

  This chapter presents the executive summary, including general performance expectations in Db2 for z/OS 12.
- Chapter 2, "Scalability enhancements and subsystem performance" on page 23

  This chapter presents descriptions of most of the many new Db2 engine functions. Most of the enhancements do not require any manual intervention.
- Chapter 3, "Synergy with the IBM Z platform" on page 83

  This chapter describes the features that take advantage of System z hardware or software, and can be considered together with the Subsystem chapter.
- Chapter 4, "Data sharing" on page 131

  This chapter describes data sharing related enhancements.
- Chapter 5, "Workload level measurements" on page 155

  This chapter describes the functions and performance enhancements that are specific to remote access.
- Chapter 6, "Distributed system " on page 187

  This chapter describes the new application functions that substantially improve the productivity of application developers.
- Chapter 7, "SQL and application enablement" on page 197

  This chapter describes various enhancements in access path selection that are not apparent to the application developer. The chapter also includes the access path stability items.
- Chapter 8, "Query performance" on page 243

  This chapter describes the performance measurements that were obtained from IBM internal workloads.

- Chapter 9, “Utilities” on page 287

  This chapter describes continued performance improvements in utilities, especially REORG, RUNSTATS, and LOAD and RECOVER.
- Chapter 10, “IBM Db2 Analytics Accelerator for z/OS ” on page 297

  This chapter describes the improvements in IBM Db2 Analytics Accelerator functions and performance.
- Chapter 11, “Installation and migration” on page 323

  This chapter provides measurements of migration and considerations at migration. The chapter also covers the performance-related system parameter changes.
- Chapter 12, “Monitoring and instrumentation” on page 341

  This chapter includes IFCID updates related to performance features and monitoring guidelines.
- Appendix A, “IBM Db2 workloads” on page 375

  This appendix is an attempt at keeping up with what the maintenance stream is adding in terms of functions and performance after the general availability of Db2 12.

1.4.1 Measurement environments

Unless described otherwise, the following measurement environments are used to obtain the results that are described in this book. BIND, REBIND, or REBIND with APREUSE are done for the static plans and packages that are run under Db2 12 for z/OS. For most of the cases, Db2 11 NFM with packages bound in Db2 11 is used:

- z/OS 2.2
- System z13 dedicated z/OS and ICF LPARs
- Dedicated network connections
- Dedicated IBM DS8000® family with zHPF enabled, mostly DS8870 (unless described otherwise)

2

Scalability enhancements and subsystem performance

As with previous Db2 releases, Db2 12 removes structural constraints to support its increasing use by high-volume concurrent workloads to enhance its availability and scalability. Several improvements in the Db2 engine allow applications to access data faster while offering easier database administration.

This chapter includes the following topics:

2.1 Fast index traversal

Access through a unique index is commonly used in Db2 online applications. However, as the index grows larger, the cost of index tree access becomes more expensive, such that index tree traversal performance became one of the most important factors in Online Transaction Processing (OLTP) performance. Searching through the index B-tree is often a large CPU contributor because of the large number of getpages and data cache misses.

The index look-aside feature that is supported in previous Db2 versions significantly improved the performance of *sequential index access* but does not improve random index access.

Db2 12 introduces a new feature to improve the performance of random index access. This new feature uses an in-memory index optimization called *fast index traversal*. The term Fast Traversal Blocks (FTB) is also used for this enhancement.

2.1.1 Introduction

In older Db2 versions, Db2 must start with the root page and traverse down all index levels until it gets to the correct corresponding index leaf page whenever a random access to an index occurs. Therefore, the number of getpages that are required for a random index access is equal to the number of index levels.

Because the minimum number of index levels is two (even if only one record in the table exists), the minimum number of getpages per random index access is two. Accordingly, when the number of rows in a table increases, the number of index levels increases, and more index levels results in more getpages for random index access.

To avoid multiple getpages for random index access, a new in-memory structure called FTB was introduced in Db2 12. This in-memory structure essentially caches the root page and all the non-leaf pages of an index in memory. Whenever a random access occurs to an index that is managed by fast index traversal, Db2 can find the wanted index leaf page number from the in-memory FTB structure instead of using getpages to traverse the index tree. As a result, random index access to an index that is managed by using fast index traversal requires only a single getpage instead of multiple getpages.

Index qualifications

Db2 uses a system task to monitor and maintains a list of candidate indexes for the fast index traversal function at the Db2 subsystem level. Db2 regularly evaluates the list of candidate indexes and adds or removes them from the FTB structure, depending on their usage pattern. To be selected as a candidate for fast index traversal, the index must be a unique index with a key size of no more than 64 bytes.

As a candidate index, the index must also meet following requirements to qualify for fast index traversal:

- The index is frequently used. To determine this condition, Db2 keeps a traversal counter for each candidate index. For each random traversal, Db2 adds to the counter. The counter is adjusted downward when a sequential index access or an index split occurs. The moving average traversal count must exceed a set threshold for the Db2 subsystem for the index to be eligible for fast index traversal.
- The number of leaf pages in the index is less than 2 million.
- Db2 can allocate the FTB structure for the index without causing paging and within the defined maximum size of the FTB structure. For more information about this limit, see "Fast index traversal memory usage" on page 25.

In addition, Db2 provides a new catalog table, SYSIBM.SYSINDEXCONTROL, to override the default candidate index processing. Fast index traversal is intended to be automatically controlled by Db2. Use the SYSIBM.SYSINDEXCONTROL catalog table only to define exceptions when the automatic processing is unacceptable. Index performance optimization might be impeded if the number of rows in this catalog table becomes too large.

Fast index traversal memory usage

More memory is required to store the fast index traversal memory structures (FTBs). The user can control the size of this storage area by using the new INDEX_MEMORY_CONTOL DSNZPARM parameter. The following options are available:

- If INDEX_MEMORY_CONTOL= 'AUTO', Db2 uses a minimum of 10 MB or 20% of the currently allocated buffer pool storage, whichever is larger.
- If INDEX_MEMORY_CONTOL= 'DISABLE', Db2 disables the fast index traversal feature and no FTBs are created.
- If INDEX_MEMORY_CONTOL equals an integer 10 - 200000 (10 MB - 200 GB), Db2 allocates up to the size of the specified value for fast index traversal usage.

Performance traces and diagnostic messages

Two new IFCID trace records were introduced to track fast index traversal usage. IFCID 389 is part of statistics trace class 8. It records all indexes that use fast index traversal in the system. IFCID 477 is part of performance trace class 4 and records the allocation and deallocation activities of FTBs for fast index traversal.

A new DSNI070I console message was also introduced. This message contains information about fast index traversal memory usage, the number of indexes that use fast index traversal and the number of candidate indexes. As of this writing, this message is issued only if the fast index traversal usage changes. DSNI070I features the following format:

```
DSNI070I subsystem-id FAST INDEX TRAVERSAL STATUS: storage-size MB, USED BY:
objects-count OBJECTS, CANDIDATE OBJECTS: candidates-count
```

For example:

```
DSNI070I -DB2A FAST INDEX TRAVERSAL STATUS: 2 MB,
         - USED BY: 1 OBJECTS, CANDIDATE OBJECTS: 1
```

Db2 12 also introduces a new **`-DISPLAY STATS`** command to take an instantaneous snapshot of all the index objects that use the fast index traversal feature. This use of this command displays the number of index levels and the amount of storage that is used by the index page set or partition in the fast index traversal storage pool. A typical command uses the format that is shown in the following example:

```
-DIS STATS(IMU) LIMIT(*)
```

If the user wants to limit the maximum number of FTBs to be displayed, an integer 1 - 9999 can be specified by using the LIMIT keyword. No option is available to limit the output to a specific database or index.

Performance overhead

When an insert, update or delete operation results in a change to an index non-leaf page in a Db2 data sharing environment and that index is using fast index traversal, Db2 issues a notify request to all data sharing members to notify them of the FTB structure change.

When other data sharing members receive the notification, their FTB storage is updated immediately. This notification process can cause more overhead in a data sharing environment for GBP-dependent indexes when index structure modifications occur frequently.

2.1.2 Performance measurements

Several performance measurements were conducted to evaluate the fast index traversal feature.

Performance results for user created test scenarios

A set of specially developed workloads were used to test the performance benefits and effect of the new fast index traversal feature in different scenarios. This section describes these tests and the performance-related observations that were reached based on the test results.

In-memory index feature is not triggered

The first test compared the case in which fast index traversal is disabled (INDEX_MEMORY_CONTOL= 'DISABLE') against the case where fast index traversal is enabled (INDEX_MEMORY_CONTOL= 'AUTO') while no indexes were using the fast index traversal feature (FTB pool is 'empty'). The purpose of this test was to evaluate the overhead of monitoring possible fast index traversal candidates when the feature is enabled but no index qualifies for fast index traversal. As listed in Table 2-1, no performance degradation was found when the results of both scenarios were compared.

Table 2-1 Comparison of FTB disabled and FTB enabled (auto) but no FTB created

Db2 DBM1 CPU time in seconds/commit			
SQL workload type	**FTB disabled**	**FTB enabled but not used**	**Delta %**
Sequential insert	0.000126	0.000125	-0.79

Random index access by using single thread random select/insert/delete

This set of tests used a PBG table and one unique index with a key size less than 64 bytes and five index levels. To eliminate the effect of data page access to the performance, all of the random select/insert/delete operations that were performed by the workload use index-only access. All tests were run by using a non-data sharing configuration.

The results for these random select/insert/delete test runs by using fast index traversal (comparing to the corresponding base runs with no fast index traversal) are listed in Table 2-2.

Table 2-2 Random select/insert/delete with FTB enabled versus FTB disabled

SQL operation	**Commit frequency**	**% delta (class 2 CPU time)**
Random select	1 row per commit	-8.5
Random select	10 rows per commit	-18.2
Random select	10k per commit	-22.4
Random delete	1000 rows per commit	-11.8
Random insert to empty table	1000 rows per commit	-7.8
Random insert to populated table	1000 rows per commit	-9.2

For random select with index-only access, up to 22.4% Db2 class 2 CPU time improvement was observed, depending on the commit frequency. The select statements perform only one index getpage per row as expected. For random inserts, approximately 9% CPU time improvement was observed. For random delete, the CPU time improvement was approximately 12%.

Random index access by using multiple threads performing random insert/delete

This workload used a PBG table space with one unique clustering index with a key size less than 64 bytes. The number of concurrent threads was 10. The tests were performed by using a non-data sharing configuration. The purpose of this set of tests was to see how fast index traversal affects performance when concurrent threads are accessing the index.

By using the concurrent random insert and delete workload with fast index traversal enabled, the performance improvements were in line with the single thread tests. For concurrent random insert, approximately 11% CPU time improvement and a 57% getpage reduction were observed. For concurrent random delete, the workload showed approximately 10% of CPU time improvement and a 42% reduction in getpage activity.

Sequential index access

This test used a PBG table space that had one unique clustering index with a 56-byte key. All selects used index-only access. The singleton selects retrieved rows by using matching index access with a key value that increases sequentially. The purpose of this set of tests was to determine how quickly index traversal affects performance when an index is accessed sequentially.

The results for the sequential index access workload are listed in Table 2-3. The results indicate that no obvious performance differences existed between tests in which fast index traversal was used and tests where fast index traversal was disabled for these sequential inserts, select, and fetch workloads. This result occurred because sequential index access uses the index look-aside feature to locate index leaf pages, despite fast index traversal being enabled and used for the index.

Table 2-3 Sequential index access with FTB created versus FTB disabled

Db2 Class 2 CPU time in seconds			
SQL statement type	**FTB disabled**	**FTB used**	**Delta %**
Sequential insert	128.95	128.28	-0.52
Sequential select	50.38	50.09	-0.59
Skip sequential select	5.11	5.17	1.02
Sequential fetch	15.81	15.61	-1.28

Effect when indexes are GBP-dependent

When the fast index traversal feature is triggered for GBP-dependent indexes in a data sharing environment, an approximate 2% CPU time overhead was observed for the sequential insert test that was performed by using a two-way data sharing configuration. In this scenario, the overhead occurred because the sending and receiving of notify messages by fast index traversal during index page splits. If other members have an FTB structure defined for an index, an index split operation for that index generates one fast index traversal-related notify message.

Simulated OLTP

The fast index traversal feature was also evaluated by using a simulated local non-data sharing OLTP workload. This simulated OLTP workload consists of four tables and each table has one unique clustering index with a key size of less than 64 bytes. The SQL statements of the workload include sequential cursor fetch operations, skip sequential updates, skip sequential select statements and random inserts. There were 10 concurrent active threads during this test. FTBs were created and active for all four unique indexes.

Comparing the run in which the indexes actively use fast index traversal with the run in which fast index traversal is disabled, the results showed a 6% class 2 CPU time improvement, a 59% reduction in index getpage activity, and a 35% reduction in the total number of getpages.

Performance results by using the IBM Brokerage benchmark workload

To further evaluate the performance benefits of the fast index traversal feature for OLTP type workloads, a series of performance tests were conducted that used the IBM brokerage transaction workload using a two-way data sharing configuration. The IBM Brokerage transaction workload is a complex OLTP workload that runs various SQL procedures to simulate transactions in a brokerage firm.

The sections below offer detailed information about how the tests were performed and the analysis of the test results. The IBM Brokerage workload has about 10,000 index objects. The workload was executed with INDEX_MEMORY_CONTROL set to AUTO and DISABLE.

In addition to the INDEX_MEMORY_CONTROL=AUTO/DISABLE tests that used Db2 12 at function level V12R1M500, the two-way data sharing IBM Brokerage fast index traversal study included a pair of INDEX_MEMORY_CONTROL=AUTO/DISABLE tests that used function level V12R1M100. At that function level, Db2 does not create FTBs for indexes that are GPB-dependent.

V12R1M500 can and does create FTBs for qualified indexes, regardless of whether they are GBP-dependent objects or not. The reason for this limitation when using function level V12R1M100 is used because a Db2 member at that function level might have to coexist with other members that are still running Db2 11 that do not know how to handle fast index traversal-related notify messages. As a result, Db2 members at function level V12R1M100 can only create FTBs only for indexes that are not GBP-dependent.

Performance results at function level V12R1M100

Running the IBM Brokerage workload by using Db2 systems at function level V12R1M100 with INDEX_MEMORY_CONTROL set to AUTO, a reduction of about 12 getpages per commit was observed when comparing against the case in which fast index traversal was disabled. The results are shown in Figure 2-1. The DSNI070I message showed that approximately 300 index objects had FTBs created and approximately 120 MB of real storage was used to store these FTBs.

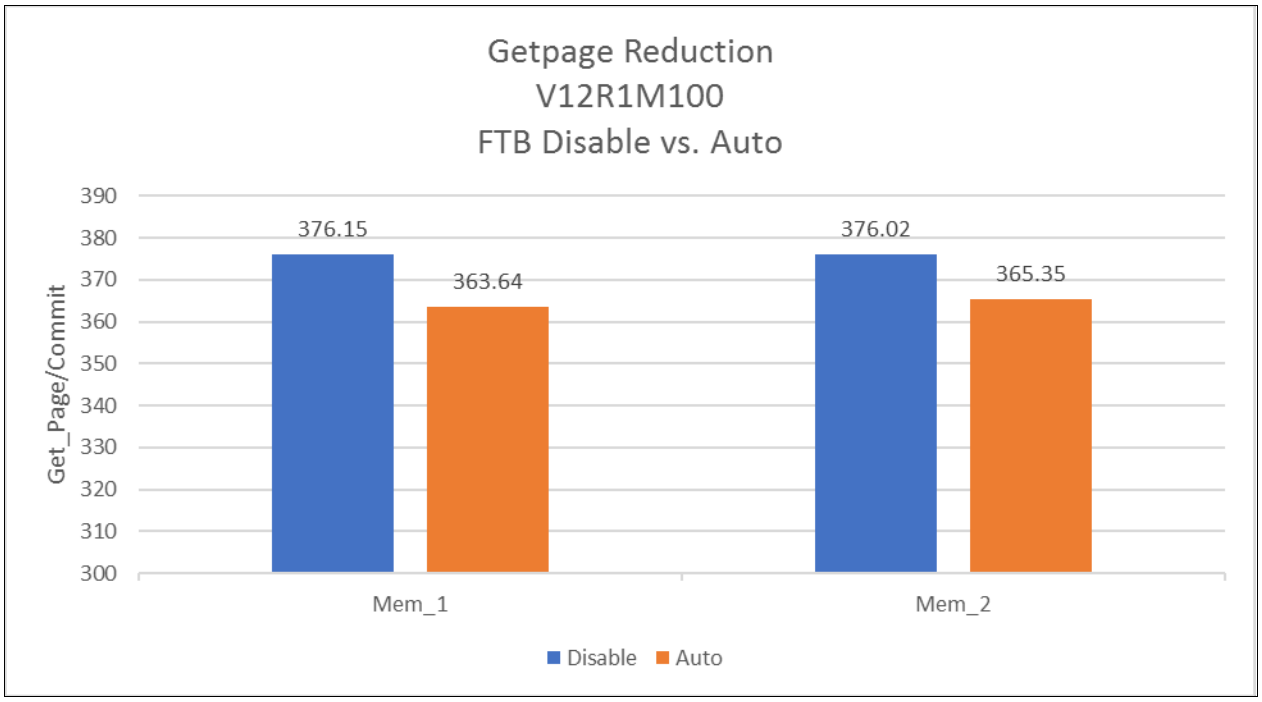


Figure 2-1 Getpage reduction between FTB AUTO and DISABLE using V12R1M100

The total Db2 CPU time per commit for each of the Db2 members for test scenarios in which fast index traversal was disabled or enabled is shown in Figure 2-2. The observed total transaction Db2 CPU times (including Db2 class 2 CPU time, MSTR, DBM1, and IRLM CPU time) are close when comparing the numbers for the two tests (within measurement noise level). Data sharing group internal throughput rates (ITR) of the two tests are also nearly the same.

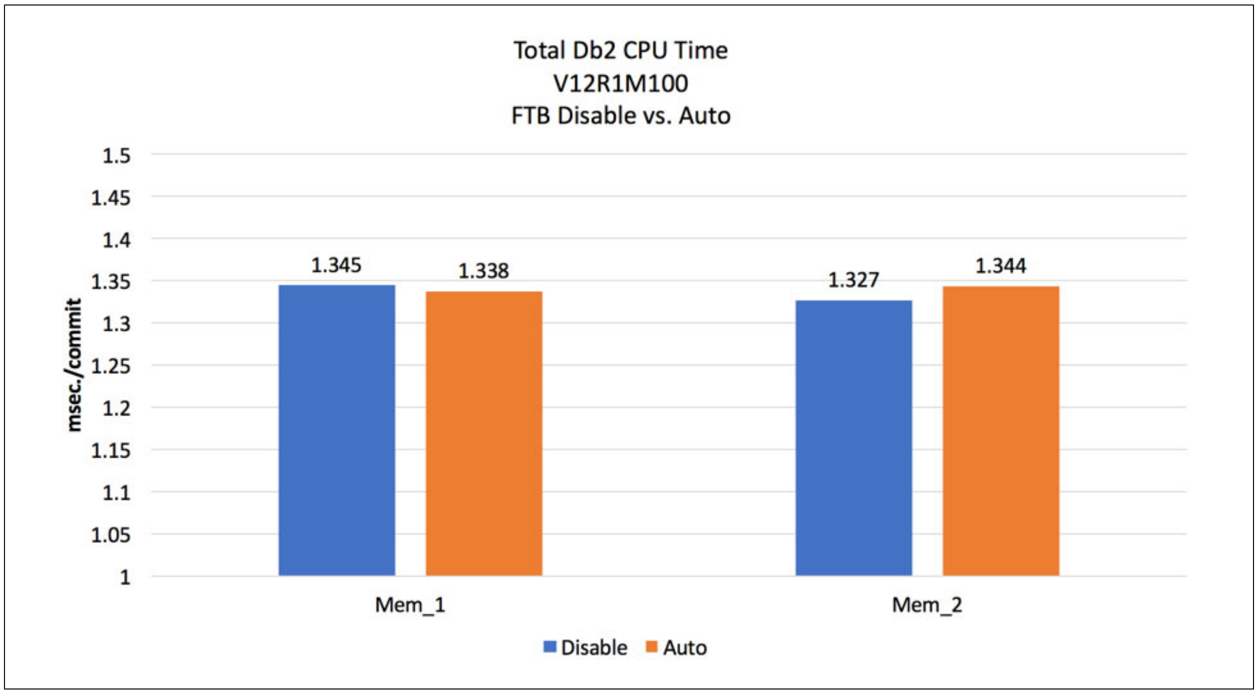


Figure 2-2 Total transaction Db2 CPU time: FTB AUTO versus DISABLE at function level V12R1M100

Performance results at function level V12R1M500

Although getpage reductions were observed with fast index traversal enabled, no significant CPU savings or throughput improvements were realized for the IBM Brokerage workload that is running in a two-way data sharing environment with the Db2 members both at function level V12R1M100.

With Db2 12 at function level V12R1M500, which does not have the non-GBP-dependency restriction, FTBs can be created for more index objects than when the system was at function level V12R1M100. Therefore, a higher getpage reduction is expected for function level V12R1M500.

As indicated by message DSNI070I, approximately 1,020 index objects had FTBs created and 415 MB of real storage was used to store all these FTBs. This result is similar to the one-way data sharing tests results. A reduction of 25 getpages per transaction was observed, compared to the case where fast index access was disabled, as shown in Figure 2-3.

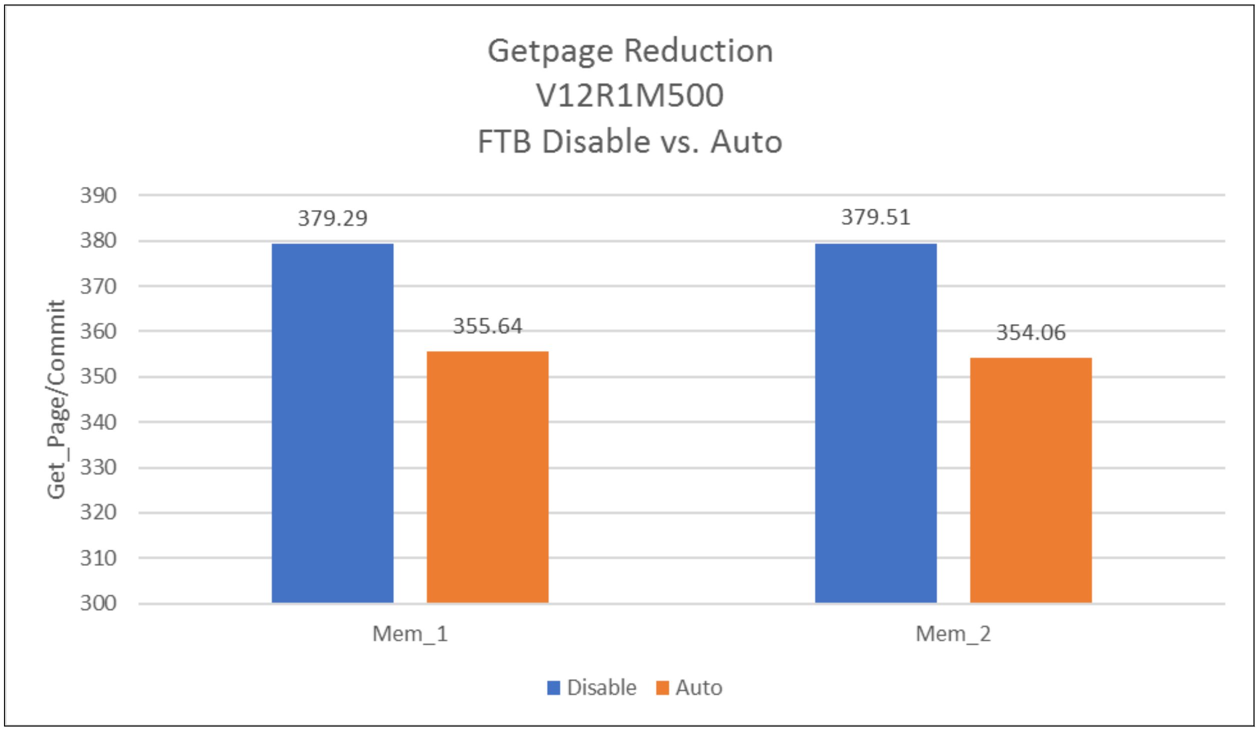


Figure 2-3 Getpage reduction between FTB AUTO and DISABLE using V12R1M500

To coordinate changes in the FTBs in a data sharing group, Db2 members use notify messages (sending and receiving) to communicate. The notify process is not necessary when the systems are at function level V12R1M100 because only non-group buffer pool-dependent indexes can use the fast index traversal feature.

This notify message traffic introduces another overhead for GBP-dependent indexes that use fast index traversal in a Db2 data sharing environment. In our test scenarios, notify message traffic was observed to be approximately 10 sends and receives every second, compared to approximately 1.5 sends and receives per second for the tests in which fast index traversal was disabled, as shown in Figure 2-4.

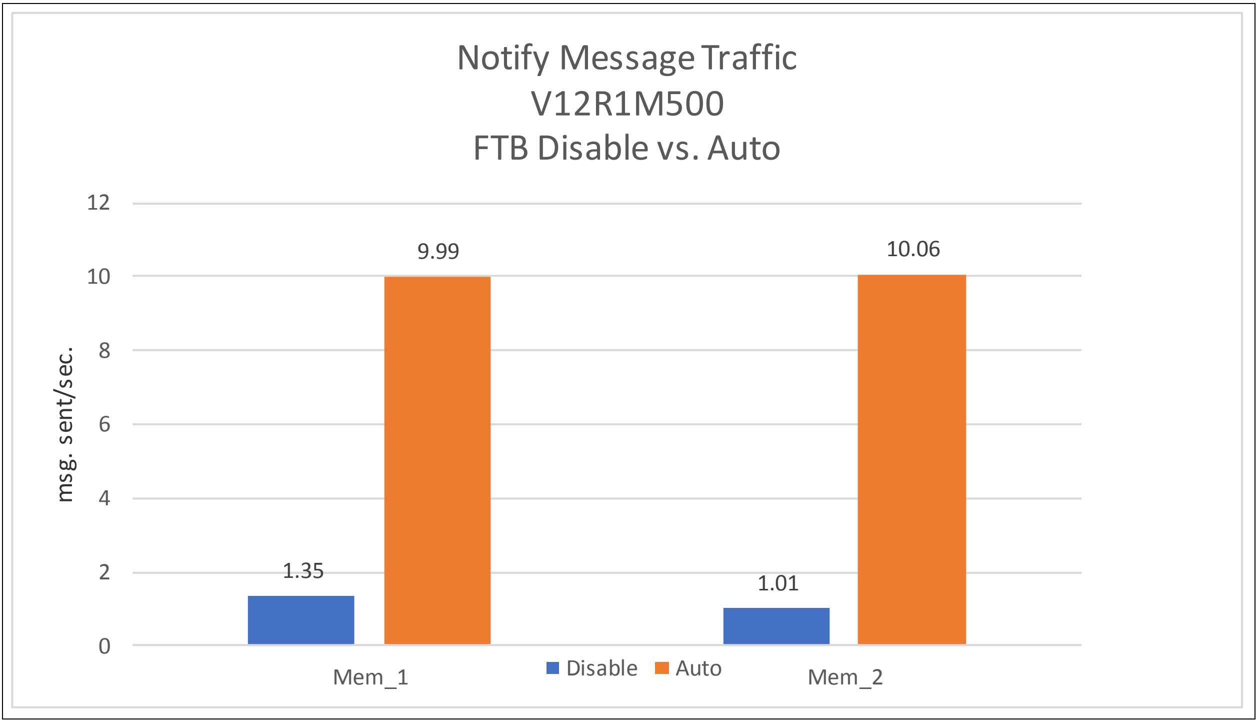


Figure 2-4 Notify message traffic: Comparing FTB AUTO and DISABLE

The observed total Db2 CPU time per commit (which includes class 2 CPU time and MSTR, DBM1, and IRLM address CPU time) shows a 2% reduction when fast index traversal is enabled (and used), compared to when fast index traversal disabled. The ITR of the data sharing group improved by 1.3% with fast index traversal enabled.

The total Db2 CPU time per commit for the respective Db2 members for scenarios that used fast index traversal AUTO versus DISABLE is shown in Figure 2-5.

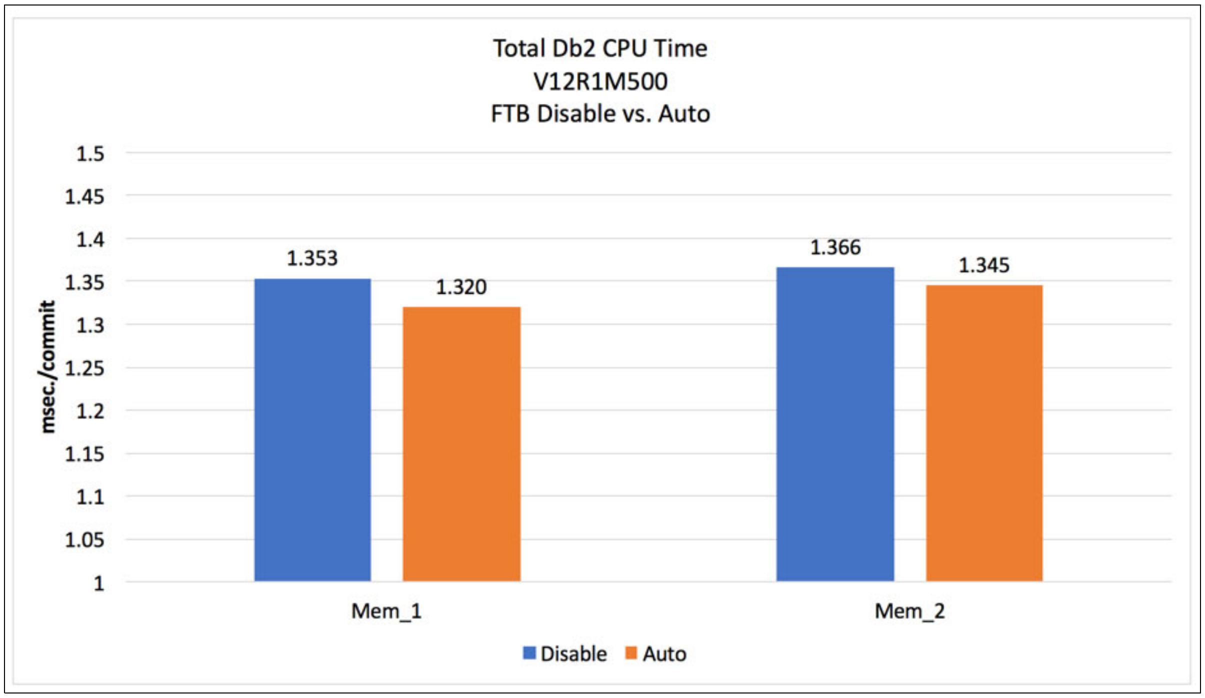


Figure 2-5 Transaction total Db2 CPU time: Comparing FTB AUTO and DISABLE

In conclusion, equivalent workload performance was observed for the tests that used INDEX_MEMORY_CONTROL AUTO versus DISABLE with Db2 at function level V12R1M100, despite a reduction of 12 getpages per commit.

With Db2 at function level V12R1M500 and fast index traversal enabled, the workload uses 25 fewer getpages per commit. Although notify message traffic increased, an overall CPU time improvement of 2% and a 1.3% increase in ITR throughput at the data sharing group level when fast index traversal was enabled and used was observed.

Performance results by using the IBM classic IRWW workload

This set of measurements were conducted in a two-way data sharing environment that used two z13 LPARs with four dedicated CPs for each LPAR, running with z/OS 2.2. The IRWW workload used the IBM IMS™ 11 transaction manager with Message Processing Program (MPPs) regions with 66% thread reuse to drive the Db2 work. The measurements were conducted simulating the migration track a typical customer follows when migrating a Db2 environment from Db2 11 NFM to Db2 12 at function level V12R1M500.

Significant performance benefits can be achieved for transactions by using random index access when the Db2 12 fast index traversal feature is enabled for the IBM classic IRWW workload, as listed in Table 2-4.

Table 2-4 Fast index traversal: Classic IRWW by using Release (Commit)

Classic IRWW Measurement	**Getpages/ commit**	**Total CPU/ commit (microseconds)**	**Getpage delta % (Db2 12 vs. Db2 11)**	**CPU delta % (Db2 12 vs. Db2 11)**
Baseline: Db2 V11 NFM	81.5	653		
Db2 12 (V12R1M100) (FTB=Auto) No rebind	70.9	629	-13%	-3.7%
Db2 12 (V12R1M100) (FTB=Auto) After rebind	70.8	621	-13%	-4.8%
Db2 12 (V12R1M500) (FTB=Disable) No new rebind	81.5	622	0%	-4.7%
Db2 12 (V12R1M500) (FTB=Auto) No new rebind	43.3	607	-47%	-7.0%

The IRWW workload runs a heavy mix of transactions by using random index access. As the fast index traversal feature is designed to help random index access performance, notable improvements in the total CPU time per commit and reductions in getpage activity were observed whenever the fast index traversal feature was enabled. (As before, total CPU time per commit = class 2 CPU time per commit + Db2 address space CPU per commit.)

As the data that is listed in Table 2-4 shows, the classic IRWW workload benefits greatly from the new Db2 12 fast index traversal feature. The best workload performance was achieved when the Db2 members were running at function level V12R1M500 with INDEX_MEMORY_CONTROL=AUTO option. Compared to the baseline Db2 11 test, it featured 47% less getpages and a 7.0% reduction in total CPU time per commit. The FTB feature contributes 2.3% to this CPU time improvement.

In comparison, the tests that were run at function level V12R1M100 showed less getpage reduction and CPU improvement because fewer indexes qualified for fast index traversal. This result occurred because many indexes in this workload were updated across the Db2 data sharing group and were GBP-dependent.

In conclusion, although getpage and CPU time savings are achieved with fast index traversal enabled when function level V12R1M100 is used in a data sharing environment, the benefits of the index traversal feature cannot be fully realized until your environment is migrated to Db2 12 at function level V12R1M500.

Performance results by using the RTW workload

Performance tests that focused on the fast index traversal feature's benefits were also conducted for the RTW workload by using a non-data sharing and a two-way data sharing group configuration. For a description of the RTW workload, refer to Appendix A, "IBM Db2 workloads" on page 375.

Non-data sharing RTW fast index traversal test results

In the non-data sharing scenario, two performance measurements were conducted that used Db2 12 at function level V12R1M500. The first measurement was run by using Db2 12 with ZPARM INDEX_MEMORY_CONTROL set to DISABLE. The second measurement was run with INDEX_MEMORY_CONTROL set to AUTO, which enabled the fast index traversal functionality.

The results of both tests in terms of total CPU time per commit (total CPU time per commit = class 2 CPU time per commit + Db2 address space CPU per commit) are shown in Figure 2-6.

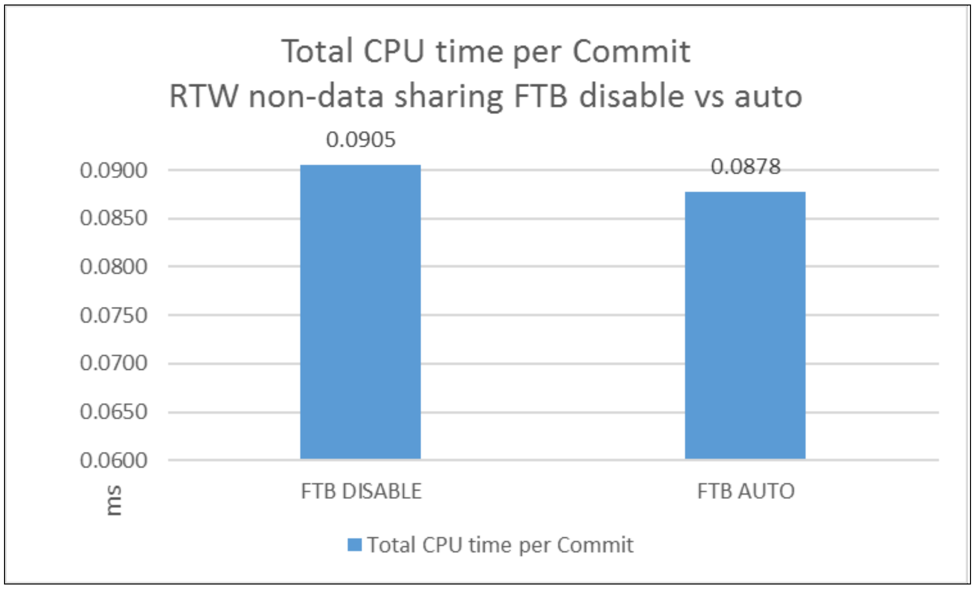


Figure 2-6 Class 2 and total CPU Time: Comparing FTB AUTO and DISABLE

As shown in Figure 2-6, when Db2 12 is run in non-data sharing mode, the RTW workload achieved a 3% reduction in total CPU time per commit with fast index traversal functionality enabled, compared to when the feature was disabled.

The internal throughput rate for the two tests is shown in Figure 2-7. The ITR numbers show an improvement of 2.16% with the fast index traversal feature enabled.

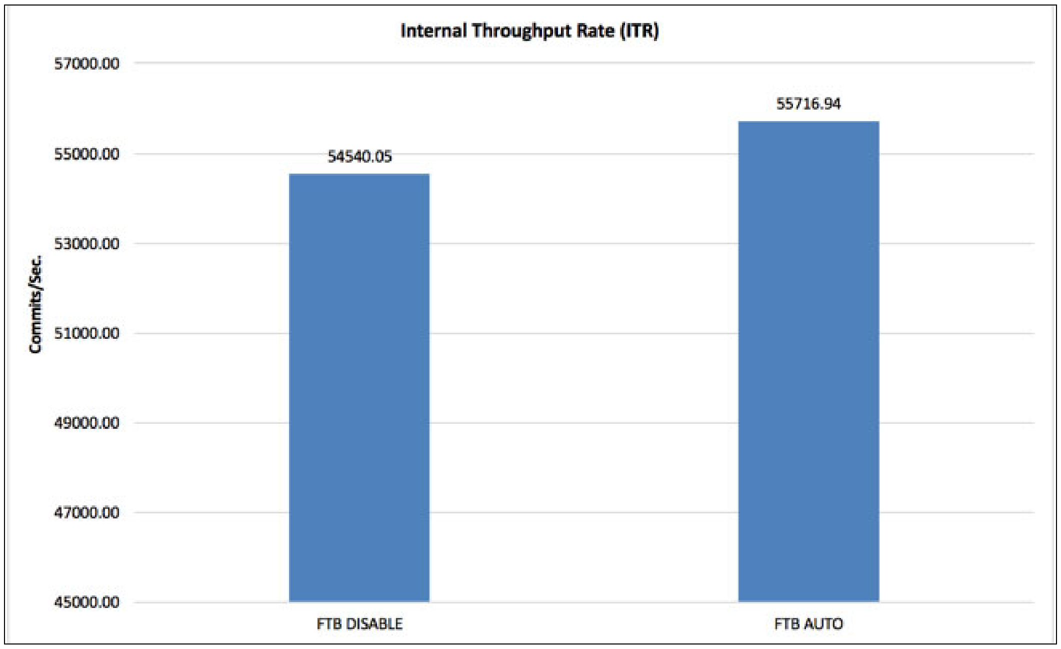


Figure 2-7 ITR: Comparing FTB AUTO and DISABLE

The improved performance of the RTW workload with fast index traversal enabled is the result of a reduction in the number of getpage requests against the index buffer pools. In total, eight indexes were managed by using in-memory structures (FTBs) when INDEX_MEMORY_CONTROL was set to AUTO. The total average getpage requests per thread dropped from 289.4 K to 236.8 K (a 18.2% reduction) compared to INDEX_MEMORY_CONTROL set to DISABLE.

Although the fast index traversal feature requires separate (extra) storage to store the in-memory managed indexes' root and non-leaf pages, the test results by using the non-data sharing RTW workload show that setting INDEX_MEMORY_CONTROL to AUTO did not cause more Db2 real storage usage compared to when the workload was run with INDEX_MEMORY_CONTROL set to DISABLE. This result occurs because the RTW workload includes small indexes and all the FTBs only took up less than 1 MB of storage, as indicated by the **`-DISPLAY STATS(IMU)`** command output.

The DBM1 address space's real storage usage information of the two test runs is shown in Figure 2-8.

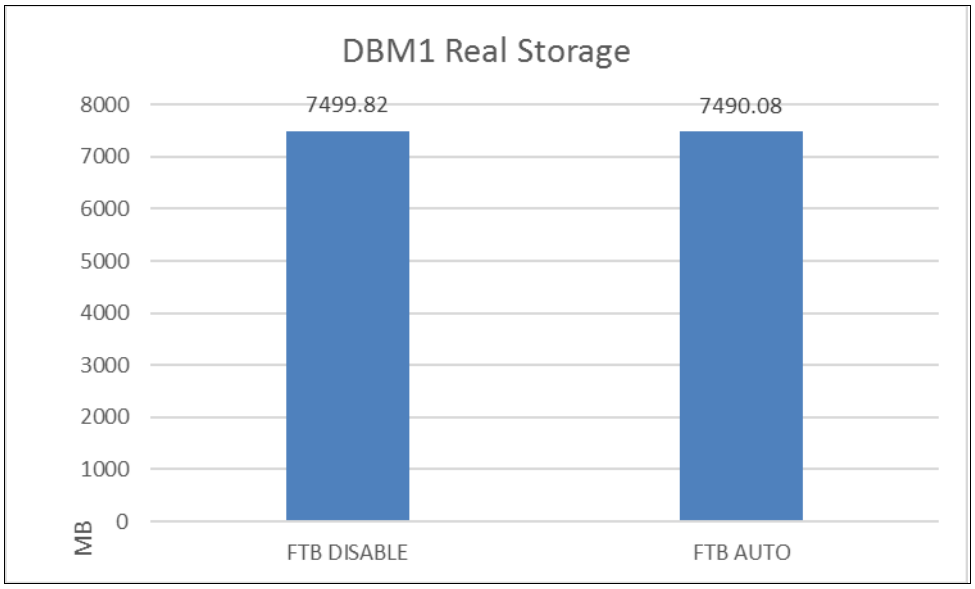


Figure 2-8 DBM1 Real storage usage: Comparing FTB AUTO and DISABLE

Data sharing RTW fast index traversal test results

In the two-way data sharing test scenario, two performance measurements were conducted, one with INDEX_MEMORY_CONTROL set to DISABLE and one with it set to AUTO. The test environment configuration can be found in 5.5, "RTW workload " on page 173.

The total CPU time per commit result of the tests (Total CPU time per commit = class 2 CPU time per commit + Db2 address space CPU per commit) is shown in Figure 2-9.

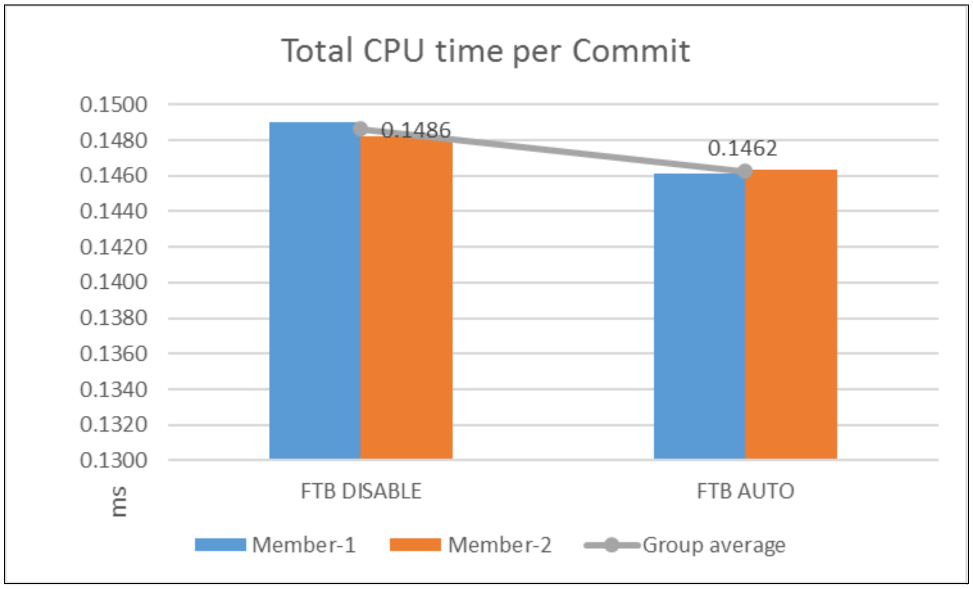


Figure 2-9 Total CPU time per Commit: FTB DISABLE versus AUTO

The group average Total CPU time per Commit values in the figure are the combined average time calculated from the data of the two Db2 members by using the following formula:

```
Total_avg = (Mem1_value * Mem1_#occur + Mem2_value *
                Mem2_#occur) / (Mem1_#occur + Mem2_#occur)
```

Figure 2-10 shows the Internal Transaction (throughput) Rate (ITR) results of the tests.

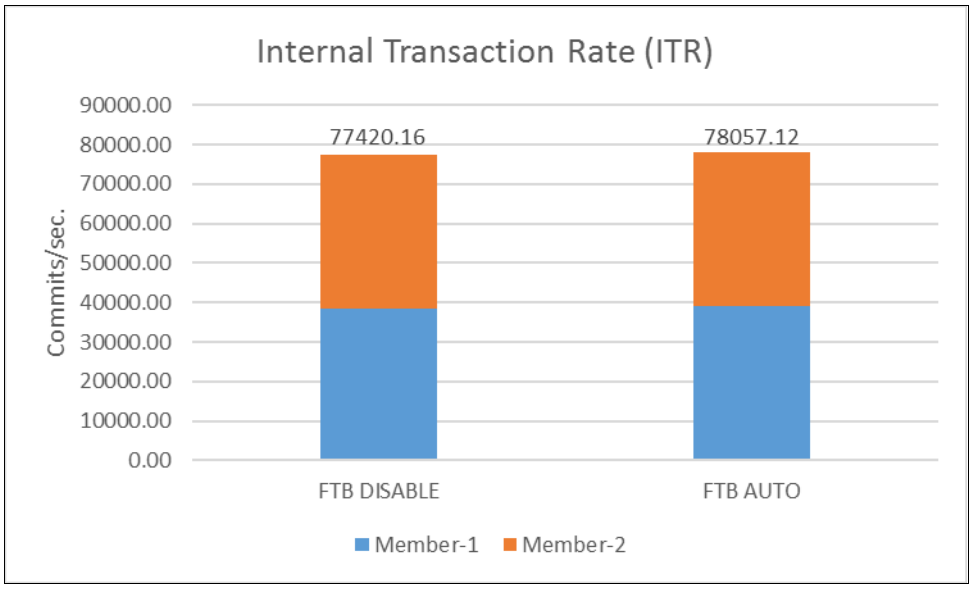


Figure 2-10 ITR: FTB DISABLE versus AUTO

The CPU time comparison that is shown in Figure 2-10 shows that with fast index traversal enabled, the two-way data sharing group achieved 1.6% improvement in Total CPU time per Commit compared to the case where fast index traversal is disabled, while the group ITR value growth achieved with the fast index traversal enabled was 0.82%.

Information about the storage usage during the two tests is shown in Figure 2-11 on page 39. The DBM1 real storage usage does not differ too much between the two measurements for either of the two Db2 members. Enabling the fast index traversal feature did not cause more real storage being allocated for the RTW workload that was measured here. The **-DISPLAY STATS(IMU)** command output showed less than 1 MB of storage was used by all the FTBs for each data sharing member.

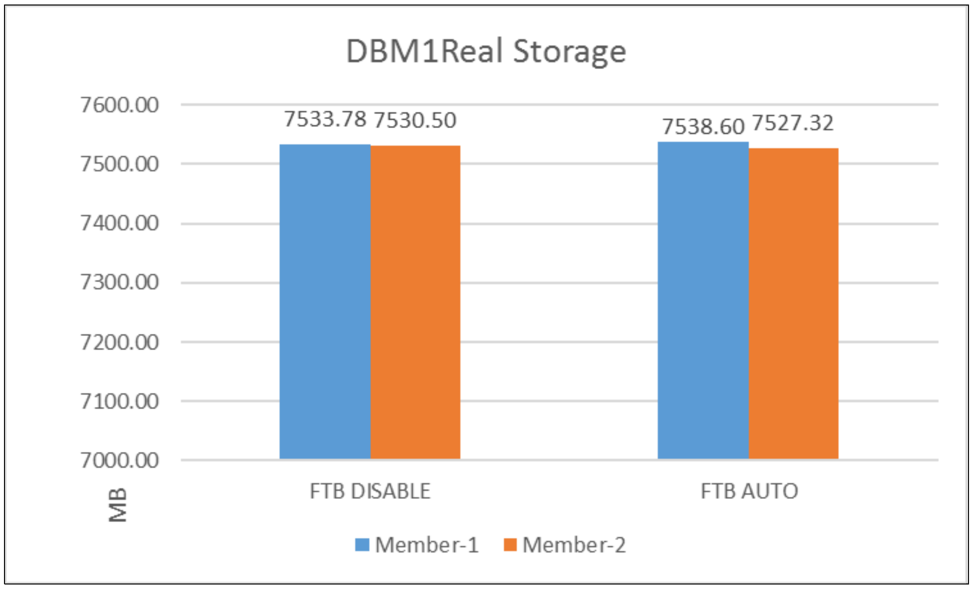


Figure 2-11 DBM1 Real storage usage: FTB AUTO versus DISABLE

RTW workload fast index traversal performance tests conclusion

Combining the non-data sharing and data sharing RTW workload test results that used Db2 12's new fast index traversal functionality, the following general conclusions were reached:

- The fast index traversal feature can bring performance benefits for the RTW workload, especially in a non-data sharing environment.
- The fast index traversal feature does not cause the amount of real storage that is used for the RTW workload to grow. This lack of growth occurs because the index sizes are small and less than 1 MB of storage is required to support the FTBs that are created.

2.2 Contiguous buffer pools

In-memory buffer pools were introduced in Db2 10 and they are enhanced in Db2 12. To emphasize their new structure, the term *in-memory buffer pool* is replaced by *contiguous buffer pool*.

2.2.1 Introduction

Db2 buffer pools are important resources for ensuring good performance because data and index pages can be cached in memory for fast access. When Db2 issues a GETPAGE request to access data in a table or index, Db2 looks for the page in the buffer pool first. When a page cannot be found in the buffer pool, Db2 issues a request to bring the page into the buffer pool from disk or the group buffer pool.

When defining a buffer pool, one of the key parameters is page steal method (PGSTEAL). When all pages in the buffer pool are allocated and new pages must be brought in, the buffer pool's chosen page stealing algorithm is used to determine which pages are to be "stolen" to make room for the new pages.

PGSTEAL features the following options:

- LRU
- FIFO
- NONE

PGSTEAL(LRU)

This option is the default page stealing method in Db2. Db2 steals the least recently used pages in the buffer pool, which ensures that frequently used pages are kept in the buffer pool. This method avoids frequently accessed pages from being stolen prematurely.

The underlying cost associated with PGSTEAL(LRU) is that Db2 must maintain an LRU chain to order the pages based on how recently they were used. This PGSTEAL option is generally recommended and used for most of the buffer pools.

PGSTEAL(FIFO)

The First In First Out (FIFO) option steals pages in chronological order without regard to how frequently a page is referenced. An advantage of this algorithm is that Db2 does not have to track page accesses because it always removes the oldest pages from the buffer pool.

Compared to PGSTEAL (LRU), PGSTEAL (FIFO) can save some CPU usage when there is no need to steal pages in the buffer pool. Therefore, PGSTEAL(FIFO) is recommended for buffer pools with no I/Os. It also is a good option for buffer pools with close to 100% hit ratio with stabilized usage.

However, if a page steal operation occurs, the more frequently used pages can be stolen prematurely. It can also be used for buffer pools that have a low hit ratio. If every getpage results in an I/O, there is no benefit in maintaining an LRU chain.

PGSTEAL(NONE)

Db2 10 introduced a third PGSTEAL option, called NONE. When PGSTEAL(NONE) is specified, Db2 (asynchronously) preinstalls the object into the buffer pool on its first access. Then, Db2 disables prefetch for the object to avoid unnecessary prefetch scheduling.

This option is useful objects that can fit in the buffer pool. If the buffer pool is not large enough to contain all the objects, Db2 uses the FIFO algorithm to manage page stealing. Therefore, if the objects cannot fit in the buffer pool because prefetch is disabled, extra synchronous I/Os can occur because frequently used pages can be stolen.

In Db2 12, the PGSTEAL(NONE) algorithm is enhanced to take more advantage of the fact that the objects are in-memory and the possibility of avoiding the buffer chain maintenance cost. This benefit is achieved by having all buffer pool control blocks laid out in memory in a sequential order and removing the need to maintain hash chains to locate a page in the buffer pool.

Contiguous buffer pools

The type of buffer pools that do not require chain maintenance are called *contiguous buffer pools*. If the objects cannot fit entirely into a contiguous buffer pool, an OVERFLOW area is always associated with each contiguous buffer pool where the pages are managed in the same way as when you specify PGSTEAL (FIFO), including the use of prefetch.

In all, 10% of the buffer pool size is set aside as OVERFLOW area, with a minimum of 50 buffers a maximum of 6400 buffers. The size of the OVERFLOW area is included in the VPSIZE of the buffer pool. Pages that cannot fit in the main portion of the buffer pool are placed in the OVERFLOW area to avoid severe performance penalty. Prefetch is enabled for pages in the overflow area.

The objects that are placed in the main portion of a PGSTEAL(NONE) contiguous buffer pool do not require any buffer chain maintenance. This feature can bring significant performance improvements over PGSTEAL(LRU) and (FIFO) defined buffer pools, especially with large buffer pools.

Generally, objects that are frequently accessed and are stable in size are good candidates for buffer pools with the PGSTEAL (NONE) attribute, provided sufficient buffer pool size is defined. If any part of an object is placed in the OVERFLOW area, Db2 cannot take full advantage of the performance benefit that is associated with contiguous buffer pools because pages in the OVERFLOW area are treated as FIFO and can cause premature page stealing to occur in a similar way as when PGSTEAL(FIFO) is used. Therefore, it is important to allocate sufficient buffers for contiguous buffer pools to avoid having any pages placed in the OVERFLOW area to obtain the best performance.

Performance benefits in elapsed time and CPU time can be expected with PGSTEAL(NONE) buffer pools when objects completely fit in the buffer pool. In such cases, objects are loaded into the buffer pools on their first access and subsequent I/Os can be avoided. Reducing (or eliminating) the number of synchronous I/Os improves elapsed time. Because every GETPAGE in a contiguous buffer pool is a direct access without any hash or LRU chain maintenance, CPU savings in accessing and releasing a page reduces Db2 class 2 CPU time.

2.2.2 Performance measurements

An OLTP brokerage workload was used to demonstrate the performance benefit of the use of PGSTEAL(NONE) compared to PGSTEAL(LRU) as buffer pool page steal algorithm.

For the first measurement, buffer pools with a high getpage count are configured to use PGSTEAL(NONE). The VPSIZE is large enough for all objects to fit completely into the buffer pool.

For the second measurement, the buffer pool setting was changed to PGSTEAL(LRU) that used the same VPSIZE. In both cases, no synchronous I/Os are needed. Therefore, all improvement in CPU or elapsed time can be attributed to the removal of the hash or LRU management.

An 8% class 2 CPU time reduction and a 7% reduction in elapsed time was observed for the benchmark run that used PGSTEAL(NONE) compared to PGSTEAL(LRU).

2.2.3 Managing a contiguous buffer pool

In DB12, PGSTEAL(NONE) is the recommended page stealing technique to minimize CPU consumption for tables and indexes that are frequently accessed and can be contained in a buffer pool.

Although the use of PGSTEAL(NONE) provides a performance advantage, the challenging task lies in identifying the tables and indexes that are good candidates to use in-memory (contiguous) buffer pools. Use the following guidelines to screen for candidates:

- Finding the Db2 objects that are referenced frequently with a high number of getpages
- Ensure that the objects can be fully contained in the buffer pool

To help with managing a contiguous buffer pool, Db2 12 introduced a new GETPAGES column in the real-time statistics tables, SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSINDEXSPACESTATS. This field reports the cumulative getpage count for a Db2 object since the last REORG, LOAD REPLACE operation or since the object was created.

The getpage count alone is not adequate to determine the best candidate objects. Getpage intensity (getpage count/object size in pages) also must be calculated and reviewed. By choosing the objects with high getpage intensity, we might avoid picking objects that are frequently accessed but reference only a small portion of a large table or index as PGSTEAL(NONE) candidates.

To further screen the objects identified with a high getpage intensity, we also must consider whether it is feasible to fit the entire object in memory. If the object is too large and cannot completely fit in the current buffer pool, a decision must be made on whether you want to increase the size of the buffer pool to use PGSTEAL(NONE), or not put the object into a contiguous buffer pool.

Also, although some objects might perfectly meet the screening guidelines in their current state, you also must take their future growth into consideration. It is not advisable to use PGSTEAL(NONE) for objects that grow quickly over time, such as objects with a high volume of inserts. Moreover, because objects that are suited and not suited for PGSTEAL(NONE) can originally be in the same buffer pool, they might need to be separated and moved to buffer pools with the appropriate PGSTEAL settings.

The next section outlines the steps that we used in our lab environment to identify objects that might benefit from the use of contiguous buffer pools.

Identifying candidates for a contiguous buffer pool

We completed several steps to identify objects that might benefit from being in a contiguous buffer pool in our complex OLTP workload. The first step is to identify objects with a high getpage count and intensity for the period of interest. After deciding on the time frame, the following the steps were completed to identify candidate objects for a contiguous buffer pool:

1. Create user RTS tables (see Example 2-1) for copying the externalized real-time statistics.

Example 2-1 Creating user RTS tables

```
//RTS#1    JOB STL,MSGLEVEL=(1,1),MSGCLASS=A,REGION=4M,CLASS=A
/*JOBPARM SYSAFF=*
//*************************************************************
//* SAMPLE DDL TO CREATE USER RTS TABLES
//*************************************************************
//JOBLIB   DD DISP=SHR,DSN=DB2DRDA.DC1K.SDSNLOAD
//STEP1    EXEC PGM=IKJEFT01,DYNAMNBR=20,TIME=1440
//SYSPRINT DD SYSOUT=*
```

```
//SYSTSPRT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//OUTFILE  DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNTRACE DD DUMMY
//SYSTSIN DD *
DSN SYSTEM(DC1K)
RUN PROG(DSNTEP2) PLAN(DSNTEP2) -
    LIB('DSNDC1K.RUNLIB.LOAD')
END
/*
//SYSIN DD *
   SET CURRENT SQLID = 'USRT002';
   DROP   TABLE RTSTABLESPACE;
   COMMIT;
   CREATE TABLE RTSTABLESPACE
    LIKE SYSIBM.SYSTABLESPACESTATS;
   COMMIT;
   DROP   TABLE RTSINDEXSPACE;
   COMMIT;
   CREATE TABLE RTSINDEXSPACE
    LIKE SYSIBM.SYSINDEXSPACESTATS;
   COMMIT;
 //
```

2. Externalize the real-time statistics before and after the period of interest and save the statistics to get the starting value and ending value of the cumulative GETPAGES columns.

   You can issue the following command to externalize the real-time statistics:

   ```
   -ACCESS DB(dbname) SP(*) MODE(STATS)
   ```

3. Issue the following SQL statements to copy the real-time statistics information into the user RTS tables after externalizing the statistics:

   ```
         INSERT INTO RTSTABLESPACE
           SELECT * FROM SYSIBM.SYSTABLESPACESTATS
                    WHERE DBNAME = 'dbname';
      COMMIT;
         INSERT INTO RTSINDEXSPACE
           SELECT * FROM SYSIBM.SYSINDEXSPACESTATS
                    WHERE DBNAME = 'dbname';
         COMMIT;
   ```

4. After the RTS data for the wanted period is collected (before and after statistics), we ran the query in Example 2-2 on page 44 to identify table spaces and indexes with the highest getpage intensity and the buffer pools these objects are in.

   To compute the getpage intensity of each table space and index, we calculated the number of getpages in the interval and divided it by the number of active pages of the objects. The buffer pools where the objects are stored are determined by merging the objects with information in SYSTABLESPACE and SYSINDEXES.

   The long query that is shown in Example 2-2 on page 44 performs the following steps. Its output was directly used to identify the candidate objects to use contiguous buffer pools:

   a. SELECT from RTSTABLESPACE and RTSINDEXSPACE to calculate the delta for the number of getpages and reduce the table to one row per object (DBNAME, NAME, and PARTITION).

b. Join the resulting (TS/IS) tables with SYSIBM.SYSTABLESPACE and SYSIBM.SYSINDEXES to obtain BPOOL and PGSIZE information.

c. Group fields to add getpages at the DBNAME,NAME (TT/II) level.

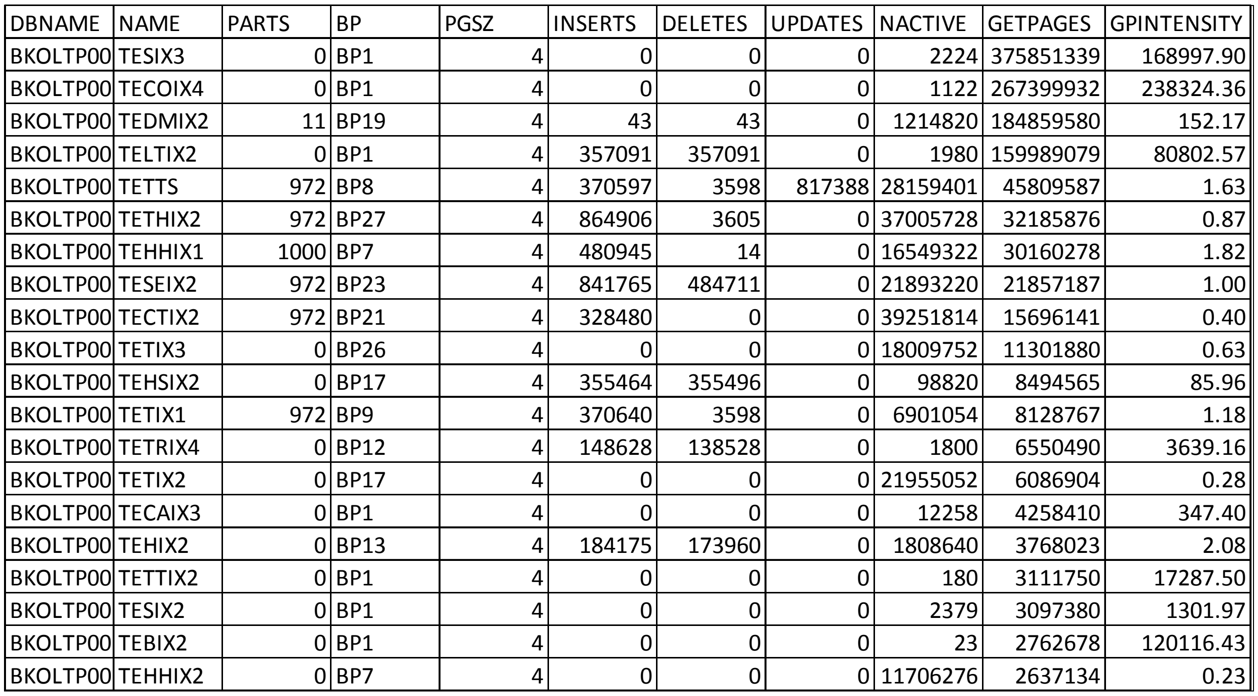
d. Calculate GPINSTENSITY for all table spaces and index spaces.

e. UNION ALL the table spaces and index spaces to produce final data (FI) and sort by GETPAGES or GPINTENSITY.

Example 2-2 Determining getpage intensity

```
SELECT
      SUBSTR(FI.DBNAME,1,8)          AS DBNAME,
      SUBSTR(FI.NAME,1,8)            AS NAME,
      SUBSTR(CHAR(FI.PARTS),1,5)     AS PARTS,
      SUBSTR(FI.BP,1,6)              AS BP,
      SUBSTR(CHAR(FI.PGSIZE),1,4)    AS PGSZ,
      SUBSTR(CHAR(FI.INSERTS), 1,10) AS INSERTS,
      SUBSTR(CHAR(FI.DELETES), 1,10) AS DELETES,
      SUBSTR(CHAR(FI.UPDATES), 1,10) AS UPDATES,
      SUBSTR(CHAR(FI.NACTIVE),1,10)  AS NACTIVE,
      SUBSTR(CHAR(FI.GETPAGES),1,11) AS GETPAGES,
      DECIMAL((FLOAT(GETPAGES) / NACTIVE),12,4) AS GPINTENSITY
  FROM (
  (SELECT TT.DBNAME,
          TT.NAME,
          MAX(TT.PARTS)   AS PARTS,
          MAX(TT.PGSIZE)  AS PGSIZE,
          TT.BPOOL        AS BP,
          SUM(TT.NACTIVE) AS NACTIVE,
          SUM(TT.INSERTS) AS INSERTS,
          SUM(TT.DELETES) AS DELETES,
          SUM(TT.UPDATES) AS UPDATES,
          SUM(TT.GETPAGES)AS GETPAGES
   FROM (
   SELECT  TS.TIME            AS TIME,
           TS.DBNAME     AS DBNAME,
           TS.NAME       AS NAME,
           TS.PARTS        AS PARTS,
           TS.REORGINSERTS AS INSERTS,
           TS.REORGDELETES AS DELETES,
           TS.REORGUPDATES AS UPDATES,
           TS.GETPAGES   AS GETPAGES,
           TS.NACTIVE    AS NACTIVE,
           ST.BPOOL AS BPOOL,
           ST.PGSIZE     AS PGSIZE
    FROM  (
  SELECT  T2.UPDATESTATSTIME AS TIME,
          T2.DBNAME          AS DBNAME,
          T2.NAME            AS NAME,
          T2.PARTITION       AS PARTS,
          T2.NACTIVE         AS NACTIVE,
          T2.REORGINSERTS - T1.REORGINSERTS AS REORGINSERTS,
          T2.REORGDELETES - T1.REORGDELETES AS REORGDELETES,
          T2.REORGUPDATES - T1.REORGUPDATES AS REORGUPDATES,
          T2.GETPAGES - T1.GETPAGES AS GETPAGES
  FROM RTSTABLESPACE T1, RTSTABLESPACE T2
  WHERE
      T1.DBNAME = T2.DBNAME AND
      T1.NAME = T2.NAME AND
      T1.PARTITION = T2.PARTITION AND
      T2.UPDATESTATSTIME > T1.UPDATESTATSTIME AND
      T2.GETPAGES > T1.GETPAGES) TS,
                         SYSIBM.SYSTABLESPACE ST
     WHERE TS.DBNAME = ST.DBNAME   AND TS.NAME = ST.NAME
       ORDER BY  DBNAME,    NAME, PARTS, TIME            ) AS TT
  GROUP BY TT.DBNAME, TT.NAME, TT.BPOOL
)
UNION ALL
(
   SELECT II.DBNAME,
          II.NAME,
          MAX(II.PARTS)    AS PARTS,
          MAX(II.PGSIZE)   AS PGSIZE,
          II.BPOOL         AS BP,
          SUM(II.NACTIVE)  AS NACTIVE,
          SUM(II.INSERTS)  AS INSERTS,
          SUM(II.DELETES)  AS DELETES,
          SUM(II.UPDATES)  AS UPDATES,
          SUM(II.GETPAGES) AS GETPAGES
  FROM (
   SELECT  IS.TIME         AS TIME,
           IS.DBNAME       AS DBNAME,

           IS.NAME         AS NAME,

           IS.PARTS        AS PARTS,
           IS.REORGINSERTS AS INSERTS,
           IS.REORGDELETES AS DELETES,
           IS.REORGUPDATES AS UPDATES,
           IS.GETPAGES     AS GETPAGES,

           IS.NACTIVE      AS NACTIVE,

           ST.BPOOL        AS BPOOL,

           ST.PGSIZE       AS PGSIZE
   FROM  (
  SELECT  I2.UPDATESTATSTIME  AS TIME,
    I2.DBNAME                 AS DBNAME,
    I2.NAME                   AS NAME,
    I2.PARTITION              AS PARTS,
    I2.REORGINSERTS - I1.REORGINSERTS AS REORGINSERTS,
    I2.REORGDELETES - I1.REORGDELETES AS REORGDELETES,
    0                         AS REORGUPDATES,
    I2.NACTIVE                AS NACTIVE,
    I2.GETPAGES - I1.GETPAGES AS GETPAGES
  FROM RTSINDEXSPACE I1, RTSINDEXSPACE I2
  WHERE
      I1.DBNAME = I2.DBNAME  AND
      I1.NAME = I2.NAME AND
      I1.PARTITION = I2.PARTITION AND
      I2.UPDATESTATSTIME > I1.UPDATESTATSTIME AND
      I2.GETPAGES > I1.GETPAGES) IS,
                         SYSIBM.SYSINDEXES    ST
     WHERE IS.DBNAME = ST.DBNAME   AND IS.NAME = ST.NAME
       ORDER BY  DBNAME, NAME, PARTS, TIME               ) AS II
  GROUP BY II.DBNAME, II.NAME, II.BPOOL
  )
) AS FI
ORDER BY (FLOAT(FI.GETPAGES) / FI.NACTIVE) DESC,
         FI.GETPAGES DESC;
```

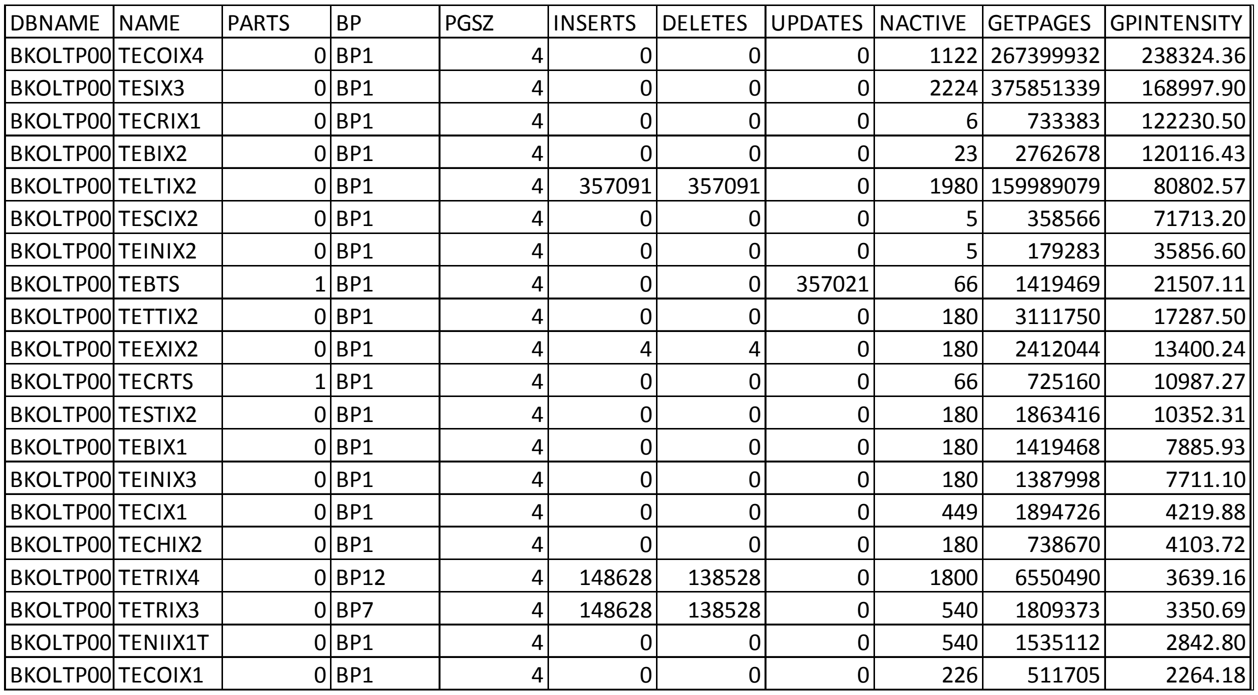
The query result for our workload is shown in Figure 2-12.



DBNAME	NAME	PARTS	BP	PGSZ	INSERTS	DELETES	UPDATES	NACTIVE	GETPAGES	GPINTENSITY
BKOLTP00	TESIX3	0	BP1	4	0	0	0	2224	375851339	168997.90
BKOLTP00	TECOIX4	0	BP1	4	0	0	0	1122	267399932	238324.36
BKOLTP00	TEDMIX2	11	BP19	4	43	43	0	1214820	184859580	152.17
BKOLTP00	TELTIX2	0	BP1	4	357091	357091	0	1980	159989079	80802.57
BKOLTP00	TETTS	972	BP8	4	370597	3598	817388	28159401	45809587	1.63
BKOLTP00	TETHIX2	972	BP27	4	864906	3605	0	37005728	32185876	0.87
BKOLTP00	TEHHIX1	1000	BP7	4	480945	14	0	16549322	30160278	1.82
BKOLTP00	TESEIX2	972	BP23	4	841765	484711	0	21893220	21857187	1.00
BKOLTP00	TECTIX2	972	BP21	4	328480	0	0	39251814	15696141	0.40
BKOLTP00	TETIX3	0	BP26	4	0	0	0	18009752	11301880	0.63
BKOLTP00	TEHSIX2	0	BP17	4	355464	355496	0	98820	8494565	85.96
BKOLTP00	TETIX1	972	BP9	4	370640	3598	0	6901054	8128767	1.18
BKOLTP00	TETRIX4	0	BP12	4	148628	138528	0	1800	6550490	3639.16
BKOLTP00	TETIX2	0	BP17	4	0	0	0	21955052	6086904	0.28
BKOLTP00	TECAIX3	0	BP1	4	0	0	0	12258	4258410	347.40
BKOLTP00	TEHIX2	0	BP13	4	184175	173960	0	1808640	3768023	2.08
BKOLTP00	TETTIX2	0	BP1	4	0	0	0	180	3111750	17287.50
BKOLTP00	TESIX2	0	BP1	4	0	0	0	2379	3097380	1301.97
BKOLTP00	TEBIX2	0	BP1	4	0	0	0	23	2762678	120116.43
BKOLTP00	TEHHIX2	0	BP7	4	0	0	0	11706276	2637134	0.23

Figure 2-12 Sample output: Getpage and getpage intensity sorted by GETPAGES

The query result sorted by GPINTENSITY is shown in Figure 2-13 .



DBNAME	NAME	PARTS	BP	PGSZ	INSERTS	DELETES	UPDATES	NACTIVE	GETPAGES	GPINTENSITY
BKOLTP00	TECOIX4	0	BP1	4	0	0	0	1122	267399932	238324.36
BKOLTP00	TESIX3	0	BP1	4	0	0	0	2224	375851339	168997.90
BKOLTP00	TECRIX1	0	BP1	4	0	0	0	6	733383	122230.50
BKOLTP00	TEBIX2	0	BP1	4	0	0	0	23	2762678	120116.43
BKOLTP00	TELTIX2	0	BP1	4	357091	357091	0	1980	159989079	80802.57
BKOLTP00	TESCIX2	0	BP1	4	0	0	0	5	358566	71713.20
BKOLTP00	TEINIX2	0	BP1	4	0	0	0	5	179283	35856.60
BKOLTP00	TEBTS	1	BP1	4	0	0	357021	66	1419469	21507.11
BKOLTP00	TETTIX2	0	BP1	4	0	0	0	180	3111750	17287.50
BKOLTP00	TEEXIX2	0	BP1	4	4	4	0	180	2412044	13400.24
BKOLTP00	TECRTS	1	BP1	4	0	0	0	66	725160	10987.27
BKOLTP00	TESTIX2	0	BP1	4	0	0	0	180	1863416	10352.31
BKOLTP00	TEBIX1	0	BP1	4	0	0	0	180	1419468	7885.93
BKOLTP00	TEINIX3	0	BP1	4	0	0	0	180	1387998	7711.10
BKOLTP00	TECIX1	0	BP1	4	0	0	0	449	1894726	4219.88
BKOLTP00	TECHIX2	0	BP1	4	0	0	0	180	738670	4103.72
BKOLTP00	TETRIX4	0	BP12	4	148628	138528	0	1800	6550490	3639.16
BKOLTP00	TETRIX3	0	BP7	4	148628	138528	0	540	1809373	3350.69
BKOLTP00	TENIIX1T	0	BP1	4	0	0	0	540	1535112	2842.80
BKOLTP00	TECOIX1	0	BP1	4	0	0	0	226	511705	2264.18

Figure 2-13 Sample output: getpages and getpage intensity sorted by GPINTENSITY

5. After objects are identified for contiguous buffer pools, the new VPSIZE is calculated for the buffer pool by summing the NACTIVE pages for all the identified objects and adding 6400 buffers for the OVERFLOW area. Then, the buffer pool size is modified by using the following **ALTER BUFFERPOOL** command:

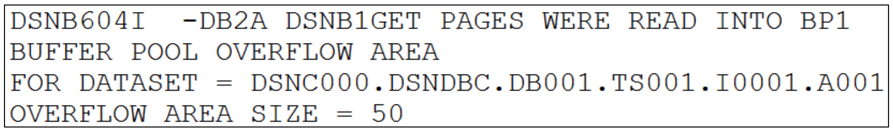
   ```
   -ALTER BUFFERPOOL(BPxx) PGSTEAL(NONE) VPSIZE(nnnnn)
   ```

   If the buffer pool is not using PGSTEAL(NONE), the change is a pending change and it takes effect the next time the buffer pool is allocated.

6. If objects must be moved from one buffer pool to another, they must be altered to reflect the new buffer pool by using an SQL ALTER statement, as shown in the following example:

   ```
   ALTER INDEX ... BUFFERPOOL BPxx;
   ALTER TABLESPACE ... BUFFERPOOL BPxx;
   ```

After the changed buffer pool is active, monitor to check whether the OVERFLOW area of the contiguous buffer pool is being used. When the OVERFLOW area is used, consider making the buffer pool (VPSIZE) larger. Message DSNB604I is written to the Db2 xxxxMSTR output and system log when pages are read into OVERFLOW area, as shown in Figure 2-14.

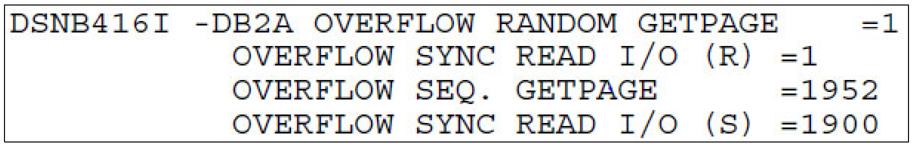


```
DSNB604I  -DB2A DSNB1GET PAGES WERE READ INTO BP1
BUFFER POOL OVERFLOW AREA
FOR DATASET = DSNC000.DSNDBC.DB001.TS001.I0001.A001
OVERFLOW AREA SIZE = 50
```

Figure 2-14 DSNB604I message

If a DSNB604I message is received for a specific object and you want to get the complete object into the contiguous part of the buffer pool, increase the size of the buffer pool (to allow for more pages), and stop/start the object (to trigger reloading the object into the contiguous part of the buffer pool).

When the OVERFLOW area is used, DSNB416I is issued as part of the output of **DISPLAY BUFFERPOOL** command, as shown in Figure 2-15.



```
DSNB416I -DB2A OVERFLOW RANDOM GETPAGE     =1
               OVERFLOW SYNC READ I/O (R) =1
               OVERFLOW SEQ. GETPAGE      =1952
               OVERFLOW SYNC READ I/O (S) =1900
```

Figure 2-15 DSNB416I message

Information about whether the OVERFLOW area of a particular buffer pool is used can also be obtained from the Db2 statistics record, as shown in Example 2-3.

Example 2-3 Db2 Statistics information about the usage of the BP OVERFLOW area

```
BP1    READ OPERATIONS     QUANTITY  /SECOND  /THREAD  /COMMIT
-------------------------- --------  -------  -------  -------
BPOOL HIT RATIO (%)           25.39
BPOOL HIT RATIO (%) SEQU        N/C
BPOOL HIT RATIO (%) RANDOM    25.39
GETPAGE REQUEST             5000.0K  7408.37  5000.0K     1.00
 GETPAGE REQS-SEQUENTIAL       0.00     0.00     0.00     0.00
  IN-MEM OVFL SEQ REQS         0.00     0.00     0.00     0.00
 GETPAGE REQS-RANDOM        5000.0K  7408.37  5000.0K     1.00
  IN-MEM OVFL RND REQS      3794.4K  5622.06  3794.4K     0.76

SYNCHRONOUS READS           3730.6K  5527.50  3730.6K     0.75
 SYNC READS-SEQUENTIAL         0.00     0.00     0.00     0.00
  IN-MEM OVFL SEQ READS        0.00     0.00     0.00     0.00
 SYNC READS-RANDOM          3730.6K  5527.50  3730.6K     0.75
  IN-MEM OVFL RND READS     3730.6K  5527.50  3730.6K     0.75
```

Note: As of this writing, the only way to determine which objects are using the overflow area is to look for occurrences of the DSNB604I message, which makes it an interesting message to capture by using some type of automation.

2.2.4 Usage considerations for contiguous buffer pools

The contiguous buffer pool feature is available in Db2 12 at function level V12R1M100 and beyond. It is essential for performance that objects are entirely contained in the contiguous buffer pool. If an object does not fit in the VPSIZE area of a buffer pool and a portion of the pages is placed in the OVERFLOW area, the performance benefit of moving to PGSTEAL(NONE) from the default PGSTEAL(LRU) most likely is not optimal, as described in 2.2.3, “Managing a contiguous buffer pool” on page 42. For the same reason, we do not recommend the use of contiguous buffer pool for tables that can grow rapidly unless a larger VPSIZE can be specified to accommodate the object’s growth to avoid the use of OVERFLOW area.

Even when the object completely fits in the buffer pool, the CPU benefit that is achievable depends on how much of the total getpage activity is for pages from contiguous buffer pools.

For example, when we observed 8% CPU reduction during our benchmark, 76% of the total getpages during the measurement were from contiguous buffer pools and 24% of the total getpages were from buffer pools with a PGSTEAL(LRU) setting.

To demonstrate the negative impact when an object is not fully contained in the contiguous area of the buffer pool, a controlled experiment was conducted using a batch program where hundred percent of the data in a table is accessed randomly via a uniform distribution pattern. To trigger the use of the overflow area during PGSTEAL(NONE) measurements, the VPSIZE is reduced by 25% in subsequent tests.

Because the getpage pattern has a uniform distribution, this results in an additional 25% pages being requested from the overflow area in subsequent tests, and when not present in the overflow area, the getpage request triggers a synchronous I/O request, resulting in more synchronous reads with a smaller VPSIZE. The VPSIZE for page steal algorithm LRU and NONE is the same for each test scenario. The following VPSIZEs were used during the different tests: 508000, 381000, 254000 and 127000 (4K) pages. Note that in each scenario the VPSIZE is large enough for a maximum overflow area of 6400 pages.

As shown in Figure 2-16, the first bar in the chart is the CL2 CPU time for LRU as page steal algorithm and the second bar is for NONE as page steal algorithm. The lines in the chart are the number of synchronous read requests from the buffer pool with page steal algorithm LRU (solid line) and NONE (dotted line).

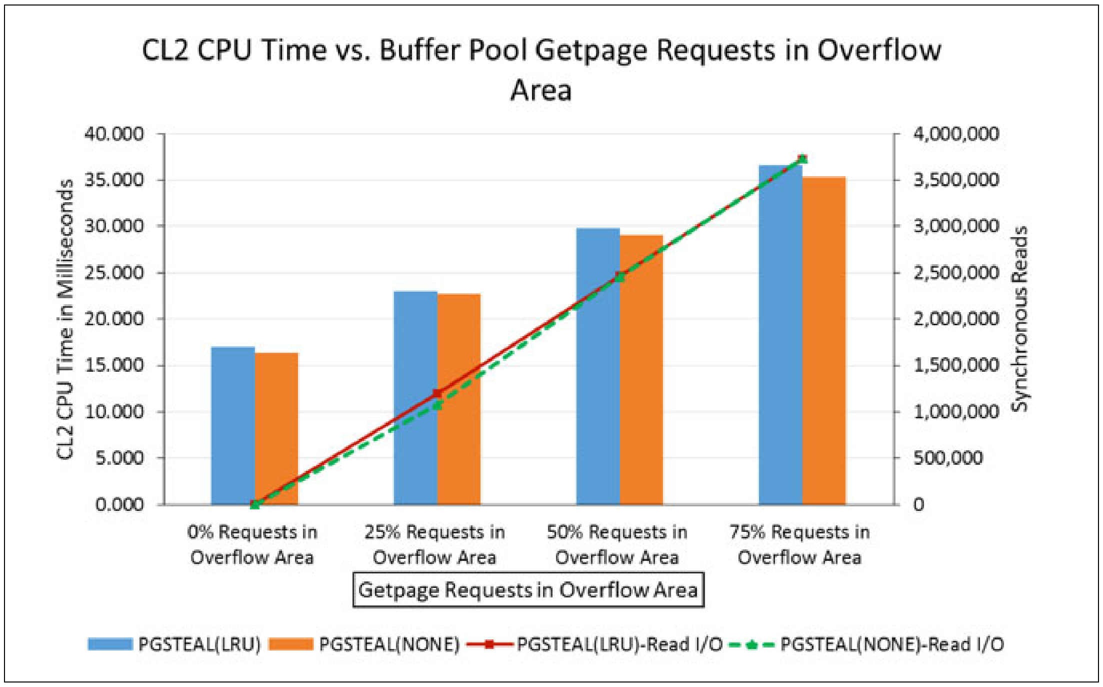


Figure 2-16 Class 2 CPU time effect of a varying number of pages in the overflow area

The bars in the chart show an increase in Db2 class 2 CPU time as more getpage requests are from the overflow area in the buffer pool and resulting in I/O when not found in the overflow area. The increased synchronous reads are the result of smaller VPSIZE. When no getpage requests from the overflow area and no synchronous reads are present, a 3% reduction in CPU was observed when using a contiguous buffer pool.

The workload that is represented in Figure 2-16 on page 49 is single threaded. We expect that with multiple threads, the Db2 class 2 CPU reduction is more profound for PGSTEAL(NONE) if the requests are from a contiguous buffer pool only. However, with multiple threads when most of the requests are from the overflow area, a considerable increase was observed in Db2 class 2 CPU time with PGSTEAL(NONE) compared to PGSTEAL(LRU).

Just looking at Figure 2-16 on page 49 may give the impression that PGSTEAL(NONE) will always outperform PGSTEAL(LRU) for equal VPSIZEs. While this is true in this scenario, please keep in mind that this is one measurement point from a very controlled environment. We are using a uniform random distribution of the getpage activity here. In such a scenario there is little benefit of maintaining an LRU chain as both scenarios result in a very similar number of synchronous I/O requests. However, real life workloads rarely have a uniform random distribution getpage pattern, and typically have either objects are more used than others, or areas within an object that are more accessed, or a combination of both, in which case maintaining an LRU chain typically results in considerably fewer I/Os.

2.3 Insert Algorithm 2

Insert transactions are common across the Db2 for z/OS customers and are often critical to their business needs. High-volume concurrent insert operations on journal tables with high volumes of data is a typical business scenario. The performance of such insert workloads always is challenge for customers.

In an application journaling scenario, data rows tend to be populated quickly with key values that are close (such as time stamps or ever increasing transaction numbers) by many concurrent threads. In many cases, rows are inserted at the end of the table spaces or partitions and the inserted data is not organized by a clustering index.

The insert rate of these tables is often throttled by the amount of contention that occurs during space search, as different threads typically all try to insert at the same place into the table at the same time.

Before Db2 12 for z/OS, customers can use the MEMBER CLUSTER and APPEND YES attributes to force new data rows to be inserted at the end of the current partition, instead of going through the traditional available space search algorithm to alleviate contention. However, even when these attributes are used, high concurrent insert scenarios can still show many page latch contentions.

To address this bottleneck during the search for available, Db2 12 introduces a new space search algorithm called *Insert Algorithm 2*.

2.3.1 Insert Algorithm 2 overview

Db2 12 takes a significant step forward with the introduction of this new insert algorithm. Insert Algorithm 2 streamlines the available space search operation to improve insert throughput for concurrent threads.

Because the use of the MEMBER CLUSTER attribute indicates that rows do not need to be inserted in clustering order, all UTS table spaces with the MEMBER CLUSTER attribute are considered as qualified candidates to use the new insert algorithm.

APPEND YES is not required to enable the new insert algorithm. The “old” space search algorithm can still be used and is now called *Insert Algorithm 1*. The new insert algorithm is called *Insert Algorithm 2*.

Note: Db2 systems must be at function level V12R1M500 or greater before the new insert algorithm can be used.

The use of the new insert algorithm can be controlled by using one of the following methods:

- You can indicate which insert algorithm to use by specifying the DDL INSERT ALGORITHM x keyword on the CREATE or ALTER TABLESPACE statement, where x can be set to on of the following values:
  - 0

    This default value indicates that the insert algorithm for tables in this table space is determined by the DEFAULT_INSERT_ALGORITHM subsystem parameter. It cannot be specified on the ALTER TABLESPACE statement.
  - 1

    The insert algorithm for tables in this table space is the basic insert algorithm. This algorithm is used for all objects before Db2 12.
  - 2

    The insert algorithm for tables in this table space is the new insert algorithm when the MEMBER CLUSTER option is also specified.
- A new system parameter DEFAULT_INSERT_ALGORITHM can also be specified. It indicates which insert algorithm to use as the default for qualifying table spaces that use the INSERT ALGORITHM 0 attribute. When the DSNZPARM value is set to 1, Insert Algorithm 1 is used; when it is set to 2, Insert Algorithm 2 is used. The default value for DEFAULT_INSERT_ALGORITHM is 2.

2.3.2 How Insert Algorithm 2 works

In this section, we described what is occurring inside the Db2 engine to make this process work. A high-level overview is shown in Figure 2-17.

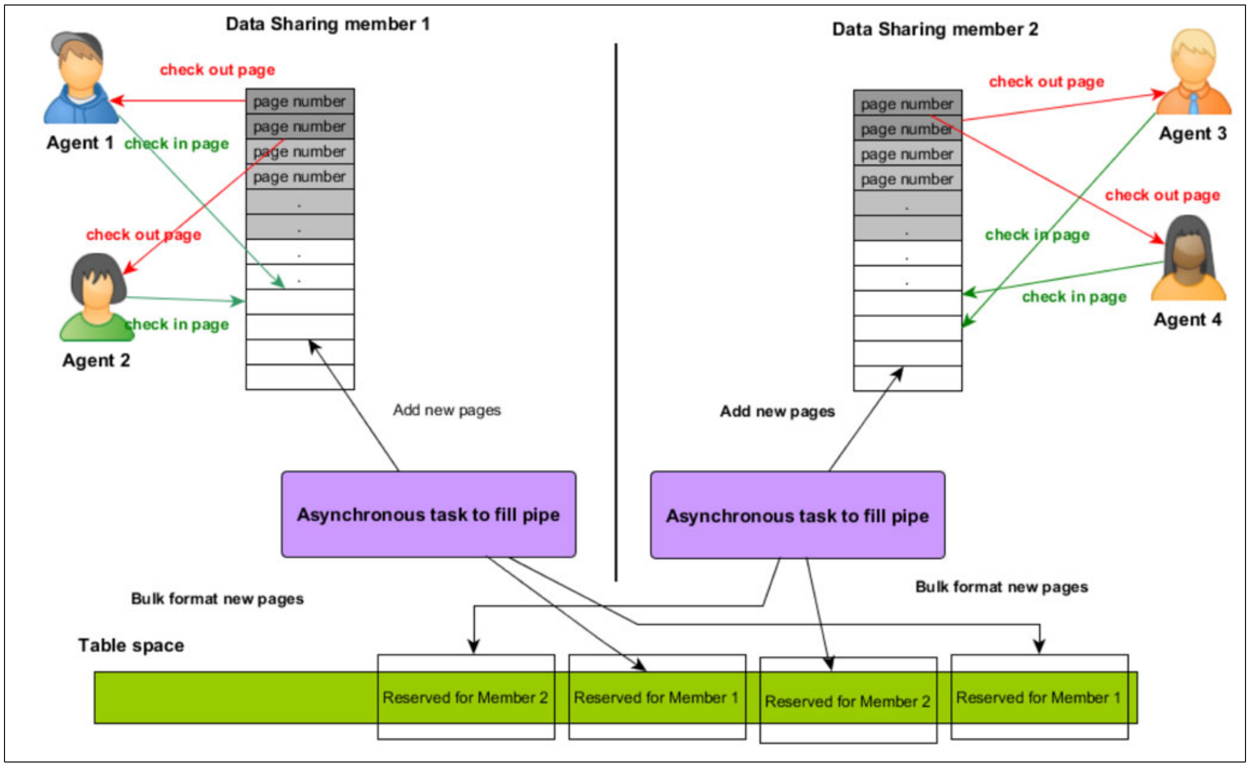


Figure 2-17 Insert Algorithm 2 overview

The system that is shown in Figure 2-17 is a data sharing group with two members. Each member has two concurrent users, all of which are inserting rows into the same table space or partition (shown in green in Figure 2-17).

When a page set is physically opened and Insert Algorithm 2 is used, an in-memory structure (called an *insert pipe*) is created. An insert-pipe contains a list of pages that are available for this member to use for insert processing for this page set partition. As each member opens a page set separately, a pipe is created on each member that uses the page set partition. A system agent is created to fill up the pipe asynchronously, making sure that pages are always available (newly formatted pages or existing pages with sufficient free space) for the threads to use for inserting rows.

Each member “reserves” a number of pages in the table space or partition, and those pages are put into the pipe. This process is similar to the MEMBER CLUSTER concept. These reserved pages per member are the rectangles that are marked with “Reserved for Member x” in Figure 2-17.

When a user wants to insert a row into the table, they “check out” a page from the pipe. Then, the user is the sole user of that page. The user holds on to the page until the page is full or a commit point is reached. Then, the space map information is updated and the page is put back on the pipe and becomes available for other users to check out.

For example, when Agent 1 is inserting data into the page that they checked out and Agent 2 wants to insert a row, Agent 2 checks out another page from the pipe to use.

With the pipe being filled up asynchronously all the time, pages should always be available for the threads to use and they do not have to search for space to put their rows. They check out a page with enough available space from the pipe and place it back on the pipe when they commit. Also, no interference occurs between the pages that are used by different members of the data sharing group. Each member uses its own set of pages (belonging to the same space map page) to fill up their pipe.

Insert Algorithm 2 is used only to speed up the insert of the data rows. If the table has one or more indexes defined, inserting the index keys is not affected by this enhancement.

2.3.3 Performance measurements

If space search in a table space is the major bottleneck during insert operations, insert throughput is significantly improved after the new Insert Algorithm 2 in Db2 12 is adopted. In addition to a throughput increase, the CPU consumption and the number of log records that is produced can also be reduced with the use of Insert Algorithm 2. During our tests, no extra space growth was observed.

In addition to the performance measurements that are described in this section, a large high-performance insert benchmark was performed, which succeeded in inserting more than 10 million inserts per second into a single table by using Insert Algorithm 2. For more information about these tests, see 5.1, “High volume INSERT benchmark ” on page 156.

Insert with no index defined

The measurements were performed in a Db2 12 two-way data sharing environment and the objects being inserted into are group buffer pool-dependent. The table is defined as PBR UTS with the LOCKSIZE PAGE, APPEND YES, and MEMBER CLUSTER attributes. The workload consists of 100 parallel insert threads, each inserting 100 rows per commit by using multi-row insert from JDBC clients. These threads run concurrently on both data sharing members.

Although it is not a typical use case, no index was created for the test table to better demonstrate the full potential of the Insert Algorithm 2. The first measurement was run by using Insert Algorithm 1; the second measurement was run with Insert Algorithm 2.

As shown in Figure 2-18 on page 54, the insert rate increased from around 950,000 per second to over 1,130,000 per second by using Insert Algorithm 2, which indicates an 8% improvement. The class 2 elapsed time per transaction reduced dramatically to 0.004 seconds per transaction, comparing against the baseline’s 0.009 seconds per transaction or about 54%. Also, the Db2 class 2 CPU time per transaction decreased by approximately 15%.

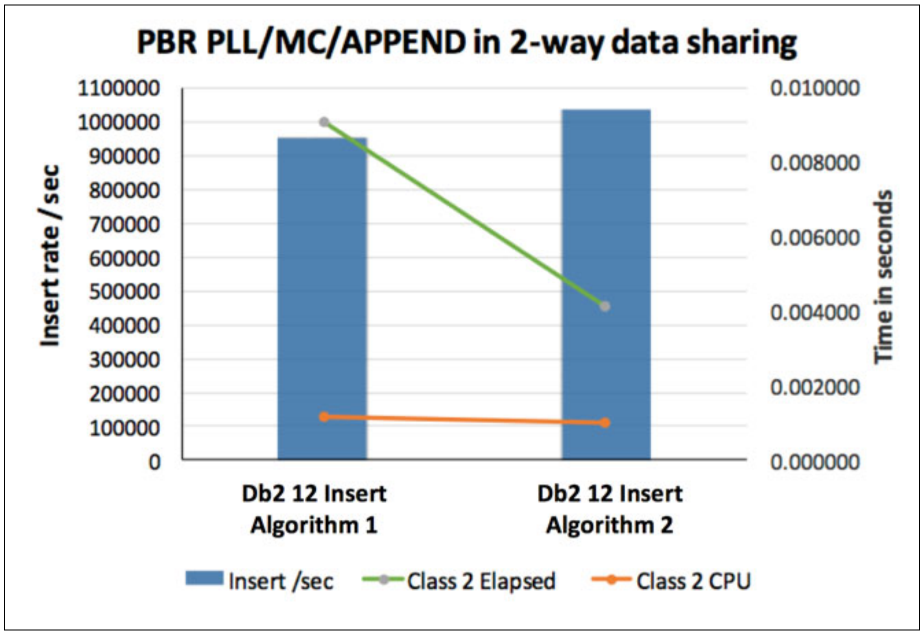


Figure 2-18 PBR MC APPEND YES insert comparison

It is important to note that there is more than 2x Db2 elapsed time improvement while the total throughput improvement is relatively small. Our analysis showed the application throughput increase was limited due to the poor Java application and network performance. This issue exists with both Insert Algorithm 1 and 2 but it is not exposed in the Insert Algorithm 1 scenario, as the main bottleneck is the Db2 elapsed time in that case - mainly long page latch suspension time. With Insert Algorithm 2, the page latch wait time is reduced significantly, and the previously hidden bottleneck now surfaces.

On the same note, if there are multiple indexes defined and the bottleneck is mostly index updates, then the improvements from Insert Algorithm 2 can be reduced. This can also be the case when the inserts are done by a single, or a few threads and there is no contention against the table space. In that case the expected improvement from Insert Algorithm 2 is none or small.

Insert with indexes defined

To evaluate how Insert Algorithm 2 performs when a table features indexes, another set of measurements was run with group buffer pool-dependent PBR journal tables that were defined with MEMBER CLUSTER, LOCKSIZE ROW, and COMPRESS YES in a two-way data sharing environment.

Three tables were used in this test with one index, two indexes, and three indexes. The tables included a clustering index, so all the rows are inserted in the order of their sequential keys. The workload consisted of 100 insert threads, each inserting 100 rows per commit by using multi-row insert. These threads ran concurrently on the two data sharing members. The first measurement was run with Db2 12 with Insert Algorithm 1 and the second with Db2 12 that used Insert Algorithm 2.

The difference in insert rate and CPU time per transaction (in seconds) between both measurements is shown in Figure 2-19. The Db2 class 2 CPU time per transaction is similar in both cases. The insert throughput increased by 26% from 325,000 per second in Db2 12 using Insert Algorithm 1 to over 410,000 per second in Db2 12 using Insert Algorithm 2.

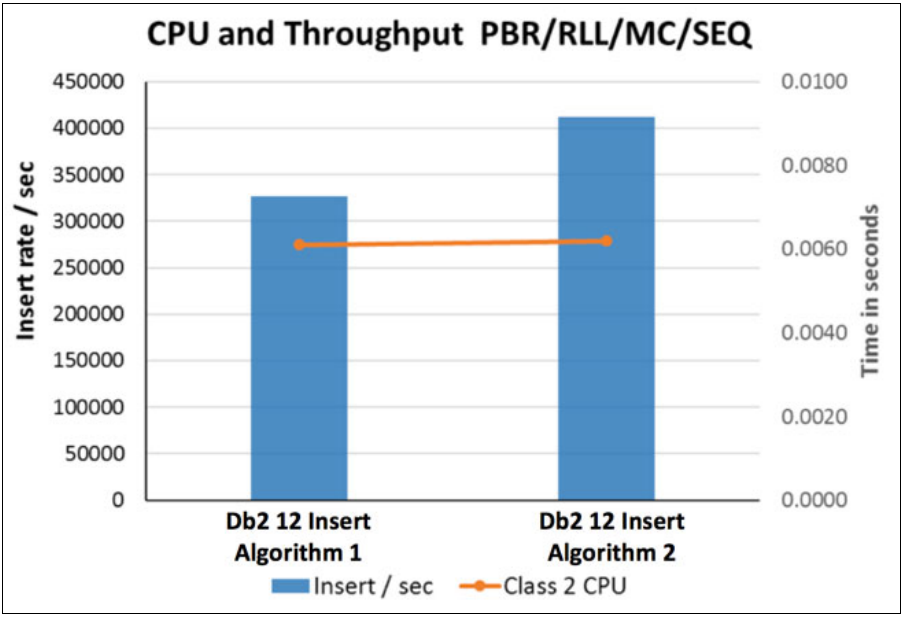


Figure 2-19 PBR/RLL/sequential insert comparison

Random insert and delete

Insert Algorithm 2 tends to benefit sequential insert by eliminating page contention. The results of a scenario that used random inserts mixed with delete transactions is shown in Figure 2-20.

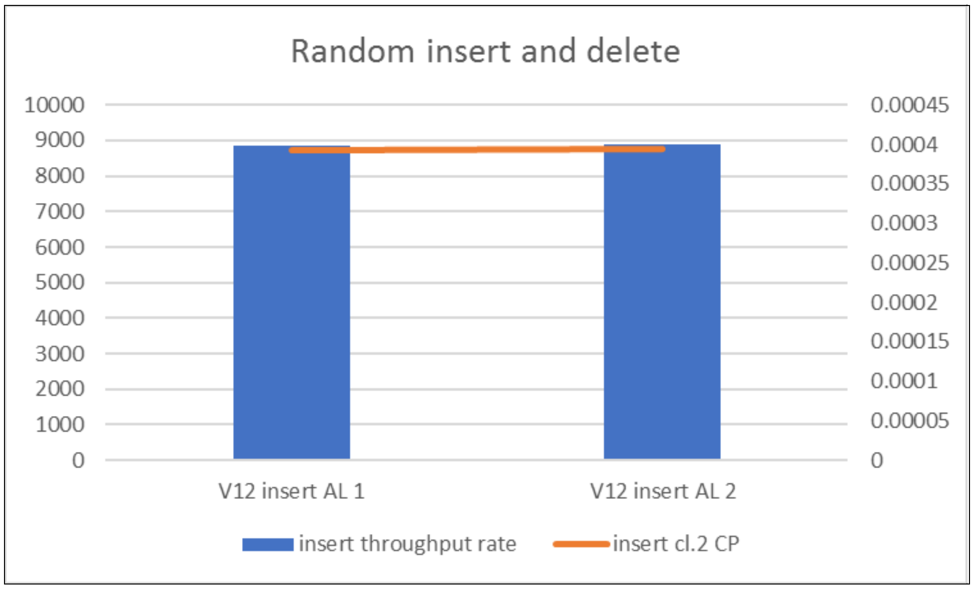


Figure 2-20 PBR/PLL random insert and delete comparison

In this case, the bottleneck is not page contention. Adopting Insert Algorithm 2 in such a scenario did not cause any performance difference. This measurement was run by using a PBR table that was defined with PAGE LEVEL LOCKING and COMPRESS YES with one index in a two-way data sharing environment.

The workload consists of 180 insert threads and 120 delete threads. These threads ran concurrently on both data sharing members. The first measurement was run with Db2 12 that used Insert Algorithm 1 and the second measurement used Insert Algorithm 2.

In this scenario, almost no difference was observed in the insert rate and CPU consumption between both insert algorithms.

2.3.4 Insert Algorithm 2 usage considerations

How much benefit can be realized from the use of Insert Algorithm 2 varies depending on the workload's specific characteristics. Generally, workloads that are constrained by lock/latch contentions on the space map pages and data pages are likely to benefit more from the new insert algorithm.

Also, by solving the space search bottleneck when Insert Algorithm 2 is used, bottlenecks can occur in other areas that are exposed by the faster inserts. It is possible for the throughput bottleneck to shift to the following areas:

- Index leaf page latch contention.
- Log write I/O performance.
- Format write during data set extending.
- BP/GBP size is not large enough. With the increased throughput of the use of Insert Algorithm 2, you might need to increase the size of the local or group buffer pool to use the system to its full potential.

Index insert speed is also vital for insert transactions. When index inserts are slow, the benefit the insert operations get from the new insert algorithm for the data pages can become less evident.

Insert Algorithm 2 is disabled internally at the partition level if lock escalations or LOCK TABLE statement being used. A new message DSNI055I is issued if Insert Algorithm 2 cannot be used for the partition. In this case, Db2 switches back to the old insert algorithm. The DSNI055I message uses the following format:

```
DSNI055I INSERT ALGORITHM 2 HAS BEEN DISABLED FOR THIS PARTITION. DBID=dbid
PSID =psid PARTITION=partition-number REASON=reason-code LOCAL TIME OF
ERROR=local-timestamp REASON FOR ERROR=error-reason-code
TRACE=internal-trace-information
```

At the time of this writing, a stop and start of the table space partition is needed to allow Db2 to re-enable Insert Algorithm 2 for that partition after DSNI055I was issued for it.

To support Insert Algorithm 2, more real storage is required to store the pipe information. If not enough real memory is available to back the extra storage that is needed, Insert Algorithm 2 is not be enabled. Customers who want to use this new feature must plan for the extra real storage usage and might also need to adjust the local buffer pool and group buffer pool size for the qualifying table space partitions and the Db2 members.

To monitor the use of Insert Algorithm 2, several enhancements were made to the Db2 instrumentation component. For more information, see "Insert Algorithm 2 IFCID changes" on page 350 for more details.

2.4 Logging enhancements

Although the larger than 4 GB active logs is the most visible enhancement in the Db2 logging area in Db2 12, other enhancements also were made that can bring significant performance benefits. These enhancements include latch class 19 removal and reduced forced log writes in a data sharing environment when identity columns are used.

2.4.1 Greater than 4 GB active log support

Before Db2 12, active log data sets can be up to 4 GB. Customers with heavy insert, update, and delete workloads observe frequent log switches, even when 4 GB active logs are used. Because Db2 schedules a system checkpoint when it switches active logs, active log data set switches can have a noticeable effect on transaction response time and throughput.

Also, many installations try to keep at least six hours of log data on active log data sets to meet the recovery SLAs they signed with their users. With the current limitation of 93 active log data set pairs that each have maximum size of 4 GB, the total amount of active log data set storage can be insufficient to cover a six-hour period's Db2 log data.

Db2 12 active log size enhancement

Db2 12 for z/OS greatly increases the size limit of active log data sets to 768 GB per active log data set. The maximum number of active log data set pairs remains 93.

This functionality is available after function level V12R1M500 is activated.

As before, the following methods are available to add active log data sets. They also apply to adding active log data sets that are greater than 4 GB:

- Use the Db2 change log inventory stand-alone utility, DSNJU003, with the NEWLOG option to add a log data set to the BSDS when Db2 was stopped.
- Use the Db2 `-SET LOG NEWLOG` command.

Performance measurements

An OLTP workload with heavy insert, update, and delete activity was used to evaluate the performance benefits of this enhancement. Two sets of performance measurements were conducted. The first measurement was done by using 23 GB active log data sets and the second measurement used 3.96 GB active log data sets, which served as a baseline.

Db2 system checkpoint frequency is set to CHKFREQ=10,000,000 in both measurements. For the test with 23 GB active log data sets, a system checkpoint is triggered in period #3 and #6. During those periods, the External Transactional Rate (ETR) drops, as shown in Figure 2-21 on page 58.

During the 3.96 GB active log data set measurement, an active log full condition (indicated by DSNJ002I message), occurs in period #3 and #6. However, the system experienced much steeper ETR drops than during the 23 GB active log measurement. This result occurs because the active log full conditions triggered system checkpoints. Together with the actual log switching overhead, the transaction throughput took a greater hit than during the 23 GB measurement in which only system checkpoints (driven by CHKFRFREQ) occurred.

The effect difference on the ETR between the two measurements shows that active log switches cause a bigger disruption to transaction throughput than system checkpoints.

Therefore, bigger active log data sets can improve Db2 performance by avoiding frequent active log switches.

Note: Although CHKFRFREQ was set to 10,000,000 for the 3.96 GB measurement, the active log full conditions occur before 10,000,000 records were created and no system checkpoint based on the ZPARM value was triggered. In this case, the ZPARM value no longer controls the checkpoint frequency and is instead driven by the small size of the active log sets.

A minute-by-minute ETR distribution of both measurements is shown in Figure 2-21.

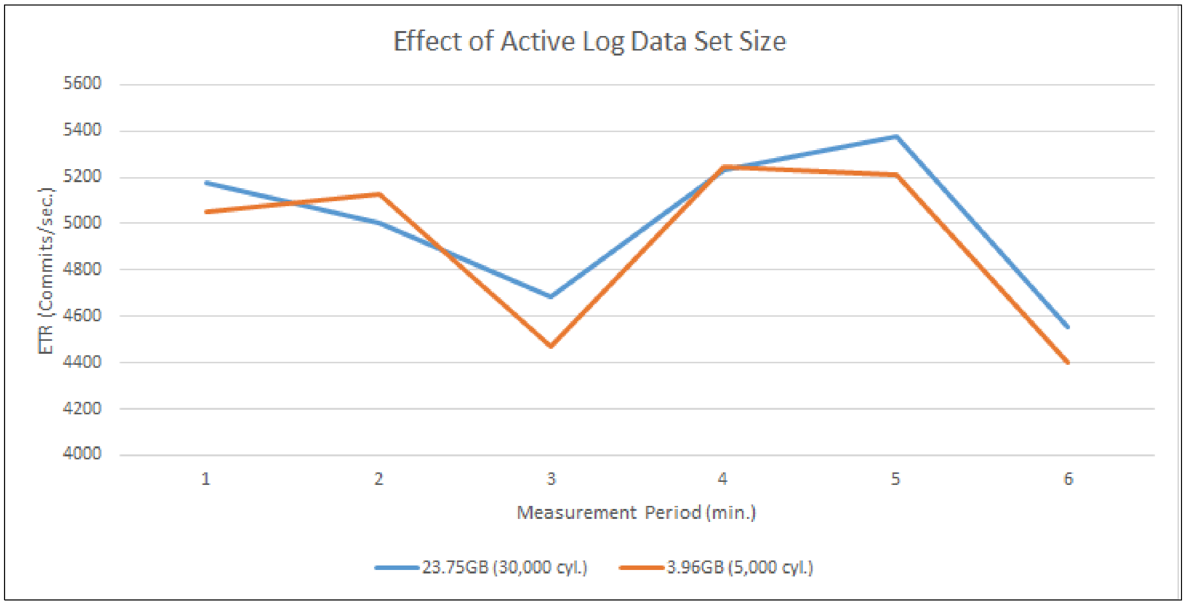


Figure 2-21 ETR throughput rate effect of different active log size

Over the six-minute measurement period, the workload Internal Throughput Rate (ITR) and the total Db2 CPU time difference between the two measurements were within measurement noise level. However, the overall ETR was higher for the measurement that used 23 GB active log data sets because the workload execution did not suffer from the “hiccups” that accompany active log full and switch conditions. The performance data of the two measurements is listed in Table 2-5.

Table 2-5 The 23 GB versus 3.95 GB measurements

Measurement ID	**3.96 GB Active log measurement**	**23 GB Active log measurement**	**Delta (32 GB versus 3.96 GB)**
CPU	2964 (z13)	2964 (z13)	
# of General CPs	12	12	
z/OS level	z/OS 2.1	z/OS 2.1	
Active log data set size	3.96 GB (5000 cyl)	23.75 GB (30000 cyl)	
Workload			
ETR (commits/second)	4917.140	5003.889	+1.76%
CPU (%)	60.64	62.03	
ITR (commits/second)	8108.741	8066.885	-0.52%
CPU time (msec/commit)			
Class 2 CPU	1.164	1.159	
MSTR CPU	0.005	0.005	
DBM1 CPU	0.013	0.012	
IRLM	0.003	0.003	
Total Db2 CPU	1.185	1.179	-0.51%

This Db2 12 feature of allowing greater than 4 GB active log data sets allows users to have more control over the Db2 checkpoint frequency by using the DSNZPARM settings. This feature avoids frequent active log full and switch conditions dictating how often actual Db2 system checkpoints occur.

Usage considerations

The use of larger than 4 GB active log is allowed only when function level V12R1M500 is activated. The new maximum size for a Db2 active log data set is 768 GB. The stand-alone log preformat utility DSNJLOGF was enhanced to allow preformatting up to 768 GB.

When Db2 detects active log data sets that are greater than 4 GB logs before CURRENT ACTIVATED LEVEL(V12R1M500), the Db2 log manager issues error message DSNJ158I. If this condition is detected during Db2 restart, Db2 ends with an abend reason code of 00E80084. If this condition is detected when a `-SET LOG` command is issued, the command fails.

After CURRENT ACTIVATED LEVEL(V12R1M500), a new message DSNJ159I that is similar to DSNJ158I is issued when the maximum size of 768 GB is exceeded.

Consider the following points regarding archive logs when the use of bigger active logs is planned:

- The size of the archive log data set, which is a sequential data set, should match the size of the active log data set.
- Adjust the archive DSNZPARMs DSN6ARVP PRIQTY and SECQTY properly. The allocation unit is always CYLINDER, as the ALCUNIT SDSNZPARM is deprecated in Db2 12. These ZPARMs are online changeable.
- If the SMS data class has the OVERRIDE SPACE field set to Y, ensure that the primary and secondary quantity that is specified in data class are correct.
- The use of SMS data class extended format (EF) for archive logs is recommended.
- Ensure that archiving can keep up with the logging volume that is produced on the active log data sets.

2.4.2 Removing log write latch (LC19)

Before Db2 12, the log write latch is acquired when space is allocated in the log buffer to block the log writer and any back-out readers from using the buffer. Because only one latch is available for the entire log output buffer, heavy concurrent (logical) log write activity can cause contention on the log latch to increase significantly. This result often makes LC19 contention a performance bottleneck for update-intensive workloads, such as high concurrent insert workloads, which produce large amounts of log records.

Db2 12 introduces an enhancement that removes LC19 contentions by eliminating the use of the log write latch, which was previously required during the allocation and initialization of log buffers. Db2 12 reduced the key information that is necessary to perform buffer initialization and space allocation in the log buffer. It also uses a more efficient serialization mechanism when changing to log buffer information.

Performance measurements

To evaluate this enhancement, an insert workload that used a partition-by-range (PBR) table space with 200 partitions was used. The table space is defined with the MEMBER CLUSTER attribute and uses page level locking. The table has a partitioned clustering index defined.

The test was done by using a two-way data sharing environment with active threads, each sequentially inserting into different partitions of the table space from the two Db2 members. This configuration ensures that latch contentions other than for LC19 are kept to a minimum. By varying the number of concurrent threads from 2 to 180, different levels of insert rates and LC19 contention intensities were achieved.

Log latch (LC19) contention results

The test results for the LC19 contention relief enhancement are listed in Table 2-6 on page 61. By using a Db2 12 code library without this enhancement and varying the number of concurrent threads from 2 to 180, LC19 contention rate grew from a small number (0.15) of latch contentions per second, to a severe (113 K) contention rate. Running the same scenario that uses a Db2 12 code library that included this enhancement, the LC19 contention rate was almost always zero across all cases. This result demonstrates that the enhancement is effective in eliminating LC19 contention.

Table 2-6 Log latch contention LC19 for different levels of concurrent threads

Number of concurrent threads	LC19 contention rate per second	
	Db2 12 pre-enhancement code library	Db2 12 post-enhancement code library
2	0.15	0.00
10	285.61	0.11
100	54487.86	0.01
180	113000	0.01

CPU time improvement

Because of the LC19 contention elimination, a reduction in Db2 Class 2 CPU time was also observed. As listed in Table 2-7, this enhancement significantly improves the Class 2 CPU time of the test application by up to 41%. The more severe the LC19 contention was during the run without the enhancement, the more CPU improvement is observed in the matching post-enhancement test.

Table 2-7 Db2 Class 2 CPU time improvement for different levels of concurrent threads

Number of concurrent threads	Db2 12 pre-enhancement code library	Db2 12 post-enhancement code library	% Delta
2	421.66	423.19	+0.36
10	68.00	59.40	-12.65
100	12.89	9.17	-28.80
180	10.67	6.22	-41.72

Note: The class 2 CPU time increased by 0.36% for the post-enhancement test when only two concurrent threads were used. However, this small increase is well within the measurement noise level and should not be deemed as a proof of a performance degradation.

Throughput

The insert throughput rate results of the tests did not show significant differences between the pre- and post-enhancement measurements. After the log write latch contention is removed, the top class 3 suspension time shifted to “service task switch/update commit” time during the post-enhancement test runs. This result is because of the log write I/O bottleneck becoming more severe. To relieve the new bottleneck, active log data sets must be relocated to faster storage (disk) devices.

2.4.3 Forced log write removal for identity columns

When an identity column reaches its CACHE value (or when a new value is generated when NOCACHE is used) in a data sharing environment, the log buffers are written (forced out) to the active log data sets. This process causes a performance overhead that is non-existent in a non-data sharing environment.

The objective of this enhancement is to maintain data integrity for identity columns in a data sharing environment by using a new Db2 internal interface that avoids the log force writes. The enhancement also applies to the values that are generated for XML DOCID columns.

Performance measurements

To test the performance benefits of this enhancement, we used 10 batch jobs, five jobs running on each of the two-way data sharing members, and all jobs inserting rows into a table with an identity column in parallel. The jobs commit after each insert. The identity column is defined with CACHE 20 and NO ORDER.

Before this enhancement, a (synchronous) log write I/O event occurs when 20 identity column values are used and a new set of cached values is obtained. With this enhancement, the log write I/Os are eliminated when the CACHE value is reached, which results in a decrease in class 3 log write I/O suspension time. In addition, latch class 11 contention is also significantly reduced, which results in a notable reduction in Db2 latch suspension time. The measurement results are listed in Table 2-8.

Table 2-8 Performance results for forced log write elimination for identity columns

	Before enhancement		**With this enhancement**		**Comments**
Db2 member	**MEM_1**	**MEM_2**	**MEM_1**	**MEM_2**	
Class 3 suspension (average time, average event)					
Db2 latch	16.650 sec. with 16,252 events	7.930 sec. with 35,695 events	13.482 sec. with 2,142 events	5.737 sec. with 26,208 events	Reduced due to less LC11 contention
Log write I/O	2.126 sec. with 5,649 events	1.467 sec. with 5,599	0.403 sec. with 626 events	0.253 sec. with 618 events	Reduced by this enhancement
Log activity					
Log records created	3200.0K	3116.3K	3201.8K	3115.6K	No change
Log CI created	79358.00	75769.00	79295.00	75718.00	No change
Log write I/O requests	520.3K	785.8K	442.0K	739.3K	Reduced by this enhancement
Log CI written	528.5K	791.3K	450.3K	744.5K	Reduced by this enhancement
Latch contention					
Latch class 11 contention	246.55/sec.	257.59/sec.	0.07/sec.	144.10/sec.	Reduced by this enhancement
CPU usage					
Class 2 CPU time (sec.)	26.130	40.126	25.251	37.263	
MSTR (sec.)	13.090	19.709	12.512	20.599	
DBM1 (sec.)	3.301	1.060	2.556	1.157	

IRLM (sec.)	6.058	8.547	5.398	8.605	
Total CPU time (sec.)	48.579	69.442	45.717	67.624	About 4% improvement with this enhancement
Total CPU time (sec.)	48.579	69.442	45.717	67.624	About 4% improvement with this enhancement

Usage considerations

The removal of log force write for identity columns and XML DOCID columns in a Db2 12 data sharing environment is automatically enabled when Db2 is at function level V12R1M100 and greater. No special configuration is required to use this enhancement.

Note: This enhancement does not apply to sequences.

2.5 Range-partitioned table spaces with relative page numbering

Before Db2 12, the maximum partition size is limited to 256 GB. In cases when the partition size is 256 GB, a maximum of 64 partitions is allowable for table spaces that use a 4 K page size. Although the maximum number of partitions is 4096, the actual maximum number of partitions a table can have is determined by its partition size and its page size. This limitation exists because pages are numbered sequentially at the table space level and the partition number of a data page is implicitly imbedded in the first n bits (where n depends on the number of partitions) of its page number.

In Db2 12, a new type of partition-by-range universal table space is introduced. This new type of range-partitioned table space uses relative page numbering where the page numbers start from zero at the partition level. Pages are no longer absolute page numbers in the table space. This new type of table space is called Partition-By-Range - Relative Page Numbering (PBR RPN).

PBR RPN table spaces can have a maximum partition size of 1 TB. The maximum number of partitions continues to be up to 4096 partitions, which means the maximum size of a PBR RPN table space is 4 Petabytes (4 PB) with 4096 partitions. The number of partitions is no longer dependent on the partition size. PBR RPN table spaces also allow certain attributes, such as DSSIZE, to be specified at the partition level instead of the table space level.

To enable this partition-independent page numbering, the size of a Record ID (RID) is expanded to 7 bytes. Internally, Db2 always uses the new 7-byte RIDs. As a result, longer log records are created in Db2 12, regardless of whether PBR RPN table spaces are used. Log records that are related to table spaces and indexes increase by 20 bytes.

Range-partitioned table spaces with relative page numbering can be created in Db2 12 at function level V12R1M500 and beyond only. A new system parameter called PAGESET_PAGENUM is also introduced by this enhancement. It controls whether page numbering of newly created PBR table spaces is relative or absolute by default. The default setting for the parameter is to use absolute page numbering (this setting is to have the same behavior as in previous Db2 versions).

The PAGESET_PAGENUM parameter can be overridden on the CREATE TABLESPACE statement by specifying PAGENUM RELATIVE or ABSOLUTE for relative or absolute page numbering. An ALTER TABLESPACE with PAGENUM RELATIVE can be used to convert table spaces from absolute to relative page numbering. However, conversion from relative to absolute page numbering is not supported.

2.5.1 Performance measurements

Several performance measurements were conducted that used workloads with regular SELECT/INSERT/UPDATE/DELETE operations and a Db2 utilities workload. All measurements were run on PBR tables with absolute and relative page numbering to compare the CPU and elapsed time of the different types of PBR table spaces.

Batch performance results: Single and concurrent threads

To evaluate the performance effect of the use of partition-by-range table spaces with relative page numbering, a single threaded batch program with random SELECT and sequential INSERT, UPDATE, and DELETE statements against a table with one index was run. One test used partition-by-range table spaces that featured absolute page numbering. The other test used partition-by-range table spaces that featured relative page numbering. The two tests did not show any significant performance differences.

A batch program that used a table space scan was also run against the two types of table spaces. The measurement results showed a 1% Db2 class 2 CPU time increase for the range-partitioned table space that used relative page numbering.

Lastly, an INSERT workload was run against the two types of table space to further validate the performance of PBR RPN objects. The workload was run by using 100 threads in a two-way data sharing environment, inserting into objects defined with MEMBER CLUSTER and APPEND YES attributes, and by using row level locking. No significant performance differences were observed comparing the CPU and elapsed time between the tests done that used a PBR table space defined with absolute page numbering versus a PBR RPN table space.

Utility performance measurements

Utility performance that used range-partitioned table spaces with relative page numbering and range-partitioned table spaces with absolute page numbering is equivalent for most utility performance tests that were run in a Db2 for z/OS lab environment (for more information, see Table 2-9 on page 65). The only exception was the online REORG that materializes the DDL changes when converting from PBR to PBR RPN.

An online REORG is required after changing the PAGENUM option from ABSOLUTE to RELATIVE to materialize the change to become a range-partitioned table with relative page numbering. This REORG (which is also known as post-conversion REORG) is expected to take longer than a regular REORG for the following reasons:

- The materializing REORG invokes inline statistics regardless of whether the STATISTICS option is specified.
- The materializing REORG disables parallelism during the unload phase.

Note: Online REORG of multiple parts of a PBR RPN table space requires the use of partition-level inline image copy data sets. Since Db2 11, you can take inline image copies at the partition level during online REORG of a range of partitions. The use of partition-level inline image copies is optional for all types of partitioned table spaces, but its usage is mandatory for PBR RPN.

Although it requires you to ensure that your COPYDDN TEMPLATEs use the &PA or &PART keyword, the use of partition level inline copies can greatly help to speed up partition level recovery that needs to restore data from an (inline) image copy created by online REORG. Without the use of partition level inline image copies, the restore phase of RECOVER must process a single image copy data set of all partitions combined, which was created during online REORG, to find the pages for the partition being recovered.

The measurement results of running diverse types of Db2 utilities, comparing CPU and elapsed time, when a UTS PBR table space is used with absolute page numbering versus a UTS PBR tables space that uses relative page numbers are listed in Table 2-9.

Table 2-9 Utility performance comparing PBR versus PBR RPN

Db2 12 z/OS 2.01, no zIIP engines	**PBR versus PBR-RPN**	
Table with 20 partitions 100M rows, 6 indexes	**CPU (CP) Delta %**	**Elapsed Time Delta %**
LOAD 100M	0%	2%
REORG	0%	-2%
REORG STATSISTICS TABLE INDEX	0%	1%
REBUILD INDEX PI	-1%	3%
REBUILD INDEX PI PART 1	1%	3%
REBUILD INDEX NPI	-2%	2%
REBUILD INDEX NPI PART 1	0%	3%
REBUILD 6 INDEXES	0%	1%
COPY SHRLEVEL REFERENCE	0%	1%
ONLINE REORG	0%	-1%
ONLINE REORG STATISTICS TABLE INDEX	0%	2%
ONLINE LOAD with 1 NPI	1%	1%

2.6 IMS-Db2 Connection pooling

Db2 for z/OS added connection pooling support for threads that use the IMS-Db2 attachment facility. This feature is enabled by using the IMS SSM PROCLIB member by setting the external subsystem module table (ESMT) parameter to DSNMIN20 instead of DSNMIN10. This support is available in Db2 11 and Db2 12, and requires IMS 14 or above. For more information about prerequisites, see 2.6.2, “Usage considerations” on page 67.

When this feature is enabled, Db2 maintains a separate connection pool for each IMS-dependent region. Within each pool, there are multiple entries (maximum 50), each one keyed from a Db2 plan name plus the active user name.

With this feature, if a user runs the same IMS transaction (plan or program) from the same IMS region, Db2 can reuse the resources that are associated with a pooled thread entry from a previous execution of that plan or program-user combination. This configuration avoids going through the process of deallocating and allocating those resources, which is typically associated with plan switches.

This feature can also result in a significant Db2 CPU time reduction for cases where creating and ending threads constitutes a large part of the IMS transaction cost.

The reduction in Db2 CPU usage can come at the cost of increased real storage usage. The storage increase on the Db2 side for connection pools themselves is negligible. However, the increase in storage usage for Db2 to keep plans and packages that are allocated and reusable is application-dependent. Any increase in storage usage should be weighed against the CPU savings that are achieved by using this feature. On the IMS side, a storage increase of about 10 KB per pooled entry per IMS region is to be expected.

Db2 JCC Statement caching

For IMS applications that are using the IMS External Subsystem Attach Facility (ESAF) connection pooling and JDBC calls (Db2 JCC using dynamic SQL), another option is available to cache the Db2 JCC prepared statements, combined with the use of the Db2 local dynamic statement cache. To enable this feature, the following set-up tasks must be completed:

- Specify DSNMIN20 (instead of DSNMIN10) as external subsystem module table (ESMT) parameter to enable ESAF connection pooling.
- Set the `maxStatements` property in `com.ibm.db2.jcc.DB2BaseDataSource` to a value greater than `0`. The default value of maxStatements is `0`, which means the statement pooling is disabled.
- Set the `keepDynamic` property in `com.ibm.db2.jcc.DB2BaseDataSource` to true.
- Set the Db2 system parameter CACHEDYN=YES to enable the global dynamic statement cache. Size the DSC pool by using the EDMSTMTC system parameter.
- Bind the Db2 packages that are started with the KEEPDYNAMIC(YES) option.
- Specify DB2JCC_ESAF_THREAD_NOTIFICATION = YES in the IMS DFSJVMEV (ENVIRON=) member. The ENVIRON parameter specifies the name of the IMS PROCLIB data set member that contains the JVM environment settings.

  The DB2JCC_ESAF_THREAD_NOTIFICATION parameter specifies whether you intend to invoke JDBC calls from any application that runs in the IMS-dependent region. When you set the parameter to YES, it also indicates that IMS notifies the Db2 JCC driver when pooled threads are in use or purged from the pool. If the application is issuing JCC calls, Db2 IMS attachment connection pooling is enabled, and maxStatements are used, you must specify YES.

The DB2JCC_ESAF_THREAD_NOTIFICATION property tells IMS to notify Db2 JCC which pool thread or connection token is in use. This setting allows JCC to set context to the proper connection and to find any previously cached statements (triggered by setting the keepDynamic and maxStatements properties).

The DB2JCC_ESAF_THREAD_NOTIFICATION property also allows JCC to clean up connections or statements that were purged from the ESAF Db2 thread pool and is required to keep things in sync.

2.6.1 Performance measurements

Several measurements were conducted to evaluate the performance benefit of enabling IMS thread pooling. A one-way data sharing group was set up for this purpose to run the classic IRWW workload, which uses the IMS-Db2 local attachment and static SQL. The following sets of measurements were done:

- A workload run without enabling IMS-Db2 connection pooling
- A workload run after enabling IMS-Db2 connection pooling

The key data points from these measurements are listed in Table 2-10.

Table 2-10 IMS thread pooling measurement results

Note: all times are in seconds	**Baseline (No IMS Thread Pooling) (A)**	**IMS Thread Pooling Enabled (B)**	**Delta (B-A)/A**
Class 2 CPU/Commit	0.000292	0.000296	1.4%
MSTR CPU/Commit	0.000062	0.000024	-61.3%
DBM1 CPU/Commit	0.000026	0.000024	-7.7%
IRLM CPU/Commit	0.000000	0.000000	0.0%
Total Db2 CPU/Commit	0.000380	0.000344	-9.5%
Class 1 Elapsed/Commit	0.009355	0.228806	2345.8%

Table 2-10 shows a significant reduction in overall Db2 CPU time per transaction for this workload when the IMS-Db2 thread pooling feature is used. Most of the CPU reduction comes from the elimination of the overhead that is associated thread deallocation and allocation activities that each new IMS transaction must go through without the pooling feature.

The Db2 Class 1 elapsed time shows a significant increase comparing the thread pooling enabled measurement to the baseline. This result is an artifact of how accounting records are cut. Because accounting records are always cut at thread termination or when a thread is reused, the class 1 timer does not stop incrementing for the thread until it gets reused when a thread is pooled and waiting to be reused. Depending on the application and overall workload in the system, there can be longer periods between Db2 thread reuses and longer class 1 elapsed time might be reported as a result.

At the time of this writing, the measurements of ESAF connection pooling combined with Db2 JCC Statement caching are being conducted, but the expectations are that this configuration brings more performance improvements.

2.6.2 Usage considerations

This feature is available in Db2 12 at function level V12R1M100. The complete functionality is available only when the PTFs for Db2 12 APARs PI74192 and PI74193 are also installed.

The functionality also is retrofitted to Db2 11 by using APARs PI61982 and PI61983. It is available with IMS 14 with APAR PI60400 that is installed or a later IMS release.

When you plan to use JCC statement caching, be sure to be at JCC 3.69.66 APAR(PI59538) or JCC 4.19.66 APAR(PI59547) level or higher. You enable this feature by using the subsystem member (SSM) in the IMS PROCLIB by specifying DSNMIN20 as the external subsystem module table (ESMT) parameter instead of DSNMIN10.

This support is valid for IMS applications that are running in IMS Message Processing Regions (MPP) regions.

As each connection in the pool is a thread into Db2, you might have to adjust the CTHREAD DSNZPARM parameter.

The connection pool is per IMS dependent region. To reuse a connection (thread) from the pool, the Db2 plan name and the active user name must match. If the names do not match, Db2 allocates a new entry in the connection pool (and goes through normal create thread processing), or remove an (LRU) entry and reinitialize it for the new thread.

By limiting the number of `planname-signon_id` combinations that can run in a dependent region, you can maximize the amount of thread reuse, as each time a new transaction comes in, an entry(thread) can be reused.

For a better statement pooling experience, set maxStatements to the maximum number of unique SQL statements that are run through the prepared statement from the JDBC applications under IMS MPP region, if the maximum number of unique SQL statements is a reasonable number. The internal statement cache requires extra memory. If memory becomes constrained, you might have to increase the JVM size or reduce the value of maxStatements.

2.7 Db2 prefetch-related enhancements

OLTP transactions and longer running queries rely heavily on Db2 prefetch engines being available when needed. It is also important that prefetch activity is triggered only when it is beneficial to use a prefetch engine to achieve good performance. Db2 12 increases the number of prefetch engines and enhances the dynamic prefetch algorithm to trigger prefetch I/O more intelligently.

2.7.1 Increased numbers of Db2 prefetch engines

Db2 uses prefetch engines to reduce I/O wait time by triggering disk I/O ahead of applications that are requesting these data pages. Prefetch reads are done by system tasks called prefetch engines. They run in the DBM1 address space and normally read multiple pages in a single I/O operation.

Starting in Db2 10, Db2 expanded the use of prefetch engines. For example, when the index I/O parallelism feature in Db2 10 is used, Db2 uses a prefetch engine to bring a single index page into the buffer pool during an insert operation. Also, a disorganized index might trigger list prefetch that requires a prefetch engine to perform the list prefetch I/O operation to avoid multiple synchronous I/Os.

Db2 prefetch engines can also be heavily used during query processing when Db2 CPU parallelism is used, as each parallel task can be driving prefetch operations.

This increased usage of prefetch engines brings elapsed time reduction in I/O operations, given that the prefetch requests can complete quickly. But, as demand for prefetch engines increases, so does the possibly of reaching the maximum number of prefetch engines that can be used at any one time.

When a prefetch request is received while all prefetch engines are busy processing other tasks, the new request is processed as a synchronous I/O until a prefetch engine becomes available. This conversion might affect the elapsed time of applications and eliminate the expected benefit of scheduling the prefetch I/Os.

To reduce such an effect, Db2 12 increases the maximum number of prefetch engines from 600 to 900 per subsystem. With 900 engines and the optimization to avoid scheduling unnecessarily dynamic prefetch requests as described in 2.7.2, “Avoiding unnecessary dynamic prefetch scheduling ” on page 69, you can expect a reduction in the number of times you see a nonzero value for PREF.DISABLED-NO READ ENG (field QBSTREE) in the Db2 statistics record IFCID 2 in Db2 12. Therefore, its effect on application elapsed time can be avoided.

2.7.2 Avoiding unnecessary dynamic prefetch scheduling

Db2 can use the following types of prefetch requests when it is reading in multiple data pages in a single I/O operation:

- List prefetch

  This type is used to prefetch data pages based on a list of qualifying record identifiers (RIDs) accumulated from one or more indexes. These RIDs might be sorted before accessing the data. List prefetch is typically used for non-sequential or sparse data access. A list prefetch request is scheduled even if all the requested data pages are in the buffer pool. List prefetch can also be automatically used to prefetch disorganized index leaf pages if an index scan results in synchronous I/Os.

- Sequential prefetch

  This type is used for table scans. The prefetch quantity is determined based on the size of the buffer pool that is used for the object being prefetched. Sequential prefetch checks for buffer pool page residency before scheduling the prefetch and avoids the request if all the pages are in the buffer pool.

- Dynamic prefetch

  Unlike list prefetch and sequential prefetch, dynamic prefetch can be triggered automatically at execution time based on Db2’s sequential detection algorithm. If pages are being accessed sequentially, Db2 can start dynamic prefetch as appropriate and continue monitoring the data access pattern, which enables and disables dynamic prefetch based on the sequential nature of the data access pattern.

Before Db2 12, dynamic prefetch scheduling had no awareness of whether the pages being prefetched were in the buffer pool. This lack of awareness often resulted in scheduling a prefetch engine to bring in several pages, only to determine that all pages were in the buffer pool. To verify how often all the pages were already in the buffer pool and scheduling the prefetch engine could have been avoided, you can check your Db2 statistics report. If DYNAMIC PREFETCH REQUESTED (field QBSTDPF) is much larger than DYNAMIC PREFETCH READS (field QBSTDIO), which indicates that Db2 triggered many prefetch requests but they resulted in few actual I/O operations, you are likely to benefit from this enhancement.

In addition to the use of extra CPU, the unneeded prefetch scheduling often led to performance issues because of contention that is related to serialization mechanisms when hot pages were concurrently accessed by several threads at the same time. Another side effect of this is that because only 600 prefetch engines per subsystem were available in Db2 11, and 900 in Db2 12, scheduling these (unneeded) prefetch engines can cause Db2 systems to frequently run out of prefetch engines.

This performance bottleneck was greatly reduced in Db2 12 with the enhancement to dynamic prefetch that avoids prefetch scheduling when the pages are determined to be in the buffer pool. Db2 12 dynamically tracks whether dynamic prefetch requests for an object result in an actual prefetch I/O. If no pages are read in from disk, Db2 stops scheduling dynamic prefetch for the object by the thread. When Db2 detects that dynamic prefetch scheduling is turned off but pages are read in from disk (synchronously), dynamic prefetch scheduling is reenabled.

Avoiding dynamic prefetch scheduling for resident buffer pool pages results in an increased maximum transaction throughput because of reduced CPU usage in the DBM1 address space, reduced contention for dynamic prefetch engines, and reduced contention for buffer pool exclusive latches (latch class 14 and 24) that are used for serialization inside the buffer pool.

Note: In some cases, it is possible to see a slight increase in the number of synchronous reads for certain dynamic prefetch access patterns in Db2 12. This increase is a result of dynamic prefetch scheduling being automatically enabled and disabled based on buffer pool page residency.

The PGSTEAL(NONE) buffer pool option was introduced in Db2 10 and further enhanced in Db2 12. This option allows for “in memory” database objects to be loaded into the buffer pool. This option avoids all prefetch activity in such buffer pools, and in Db2 12, most getpage overhead for objects can be 100% pinned in memory. PGSTEAL(NONE) is a great solution for performance-critical objects if you can determine which objects are good candidates for an in-memory buffer pool.

Compared to PGSTEAL(NONE), dynamic prefetch scheduling avoidance requires no user tuning skills or expert knowledge of data access patterns. It also does not require an object to be contained in memory entirely to take advantage of its performance benefits.

This feature is automatically enabled without special configurations with Db2 12 function level V12R1M100 and beyond.

Performance results for dynamic prefetch scheduling avoidance

To evaluate this enhancement, several measurements were performed by using internal IBM workloads. The Crystal Reports workload is a complex OLTP workload. It showed 80% reduction in the number of dynamic prefetch requests, a 90% reduction in the number of latch class 24 contentions, and a 60% reduction in LC14 contention (LC14 and LC24 are Buffer Manager latches). Culmultaively this resulted in a 5% reduction in total CPU time.

The IBM brokerage workload was also used to evaluate the performance benefits of this enhancement. In Db2 11, the workload issued approximately five dynamic prefetch requests per commit, which resulted in only 0.07 page reads per commit. The difference between five and 0.07 indicates that most of the prefetch requests are scheduled for buffer pool resident pages and are unnecessary.

In Db2 12, the number of dynamic prefetch requests per commit was reduced to 0.8, which is approximately 84% fewer dynamic prefetch requests than Db2 11. Also, in this workload, the reduced number of dynamic prefetch requests resulted in a large reduction in LC24 buffer pool latch contentions (94%). The latch contentions of other buffer pool-related latches also notably decreased. The overall effect is a 2% improvement in Db2 Class 2 CPU time and a 37% reduction in DBM1 CPU consumption compared to Db2 11.

2.8 Performance enhancements for short transactions

The next set of enhancements target environments with high volume and short running transactions. In such environments, relatively simple operations can make up a considerable portion of the elapsed and CPU time of such short running transactions. The enhancements that are described in this section target specific areas and achieve performance improvements mostly by keeping information around longer or caching information.

2.8.1 DGTT enhancements

A Declared Global Temporary Table (DGTT) is used to store records temporarily. They are commonly used to sort or query intermediate result tables that contain many rows but only a small subset of those rows are to be stored permanently.

Db2 11 provided a NOT LOGGED option for DGTTs to improve the performance of populating or updating DGTTs. Db2 11 also provided a technique to avoid the prepare for repeated SQL executions to further improve DGTT processing.

However, the cost of creating a DGTT remains a concern for short running transactions in Db2 11. If an application creates a DGTT and uses it only lightly by inserting a few records before dropping the DGTT, the relative cost of declaring the temporary table can be notably expensive compared to the total cost of the transaction. The cost of declaring DGTTs can be high because of the dynamic nature of the tables.

Unlike regular tables, there is no permanent information stored in the Db2 catalog or directory for DGTTs. During the DGTT declare process, Db2 must “clone” part of the Db2 catalog information within the DGTT work file table space, which are sometimes referred to as *virtual catalog tables*.

Db2 12 provides an in-memory copy of the catalog information that is required to declare a global temporary table. Instead of looking up Db2 catalog or directory information whenever a DGTT is created, this in-memory catalog copy is used. The information is saved in memory when the first DGTT is created after Db2 startup and is shared among all threads in the Db2 subsystem thereafter. Furthermore, another optimization is implemented to reduce the number of getpage operations against the work file table space during the declare DGTT process.

This enhancement is available in Db2 V12R1M100 and beyond.

Performance measurement

Several performance measurements were run to assess the benefits of Db2 12's enhancements for DGTT processing. During the measurement, simple Db2 transactions were concurrently run. They all ran a DECLARE GLOBAL TEMPORARY TABLE statement, inserted 10 - 20 rows into the DGTT, selected a few rows, and then, committed by using the ON COMMIT DROP TABLE option. The data manipulations that was performed against the DGTT in these transactions are simple to best demonstrate the benefit of the enhancement.

As shown in Figure 2-22, a 32% Db2 CPU time saving and 26% elapsed time reduction is observed for these short running transactions when comparing Db2 12 against Db2 11.

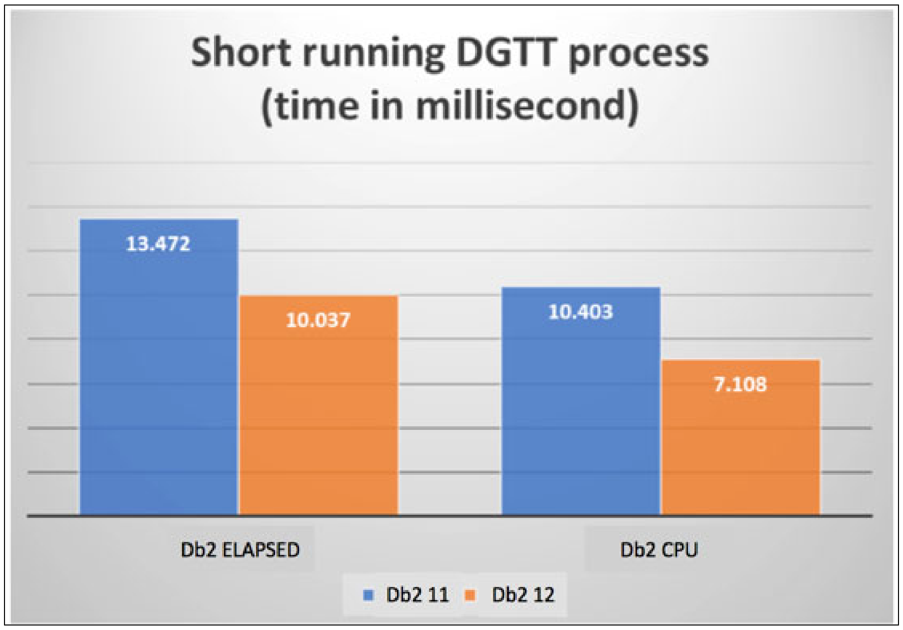


Figure 2-22 Short running DGTT transactions

The CPU time saving that is shown in Figure 2-22 is the result of the reduction in the number of getpages against the Db2 catalog and work file table spaces, as shown in Figure 2-23.

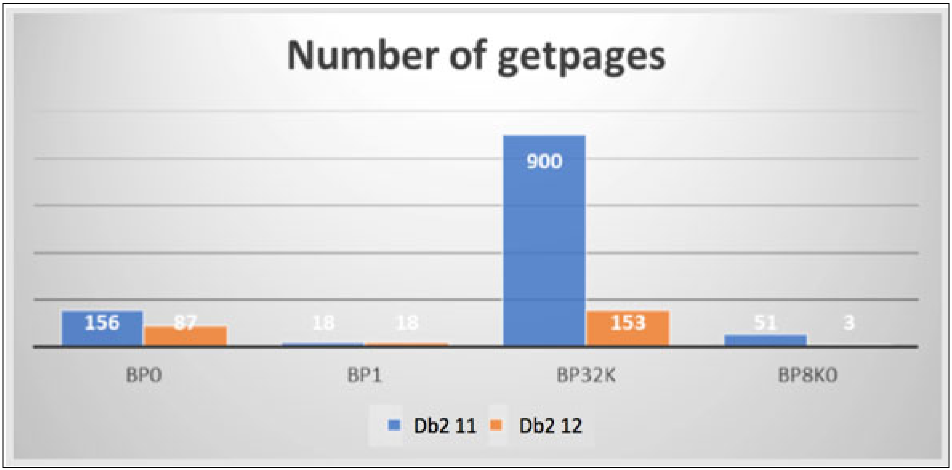


Figure 2-23 Number of getpages per buffer pool

The transactions in this measurement are lightweight and are designed to demonstrate the Db2 12 DGTT enhancements that are described in this section. If many inserts, fetches, updates, or deletes are performed against the DGTT, the process of declaring the DGTT becomes less significant and the overall benefit of this enhancement is less visible. Therefore, the degree of the improvement is highly dependent on the complexity of DGTT process.

2.8.2 Claim and declaim performance improvements

Before Db2 12, all claims that were held by an agent are released (declaimed) at COMMIT time, even for packages bound with the RELEASE (DEALLOCATE) option. This result does not benefit the performance of applications that are bound with RELEASE (DEALLOCATE), which typically have short-running SQL statements only that are run within a commit scope. As a result, the CPU cost that is incurred from the frequent claiming and declaiming can take up a significant portion of the total CPU time of such applications.

In Db2 12, Db2 skips most declaim activity at COMMIT time for packages that are bound with RELEASE (DEALLOCATE). In this way, the CPU cost of declaiming at commit time is avoided. This process also avoids the CPU cost of reacquiring the claims again after the commit. However, it is not always possible for Db2 to avoid declaim processing.

Db2 releases the claim if another agent is waiting on the claimed object, as we still want to allow utilities and other processes to “break-in” and perform their job. Declaims for writes are also not skipped. In addition, claims on Db2 catalog or directory objects, LOBs, objects touched by scrollable cursors, and claims against XML objects that use versioning are always released at COMMIT time.

This declaim avoidance is effective in Db2 12 at function level V12R1M100 and beyond. It does not require any special configuration or setup besides the use of the RELEASE(DEALLOCATE) BIND option. This feature also works for dynamic SQL with KEEPDYNAMIC(YES).

Performance measurements

Performance measurements were conducted that used the Brokerage workload. The packages are bound with RELEASE (DEALLOCATE) to demonstrate how many claims can be skipped and how much CPU time reduction these claim skips can bring in Db2 12 compared to Db2 11.

As shown in Figure 2-24, the average Db2 class 2 CPU time reduction is close to 6% in the Db2 12 measurement compared to the Db2 11 measurement.

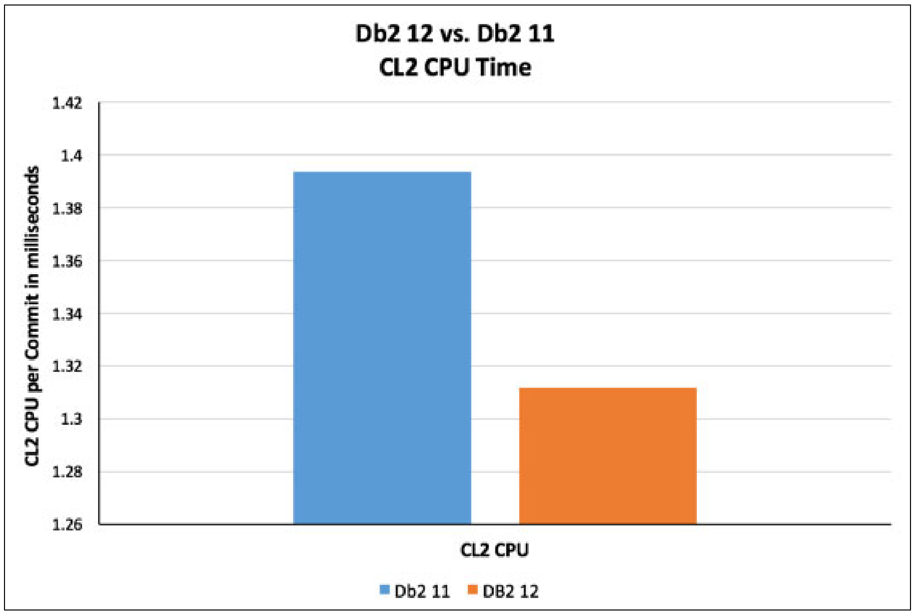


Figure 2-24 Class 2 CPU reduction for the Brokerage workload run in Db2 12 versus run in Db2 11

The number of claim requests is shown in Figure 2-25. As expected, the number of claim requests per transaction dropped. During this measurement, a 25% reduction in the number of claim requests was observed for the Db2 12 measurement compared to the Db2 11 measurement.

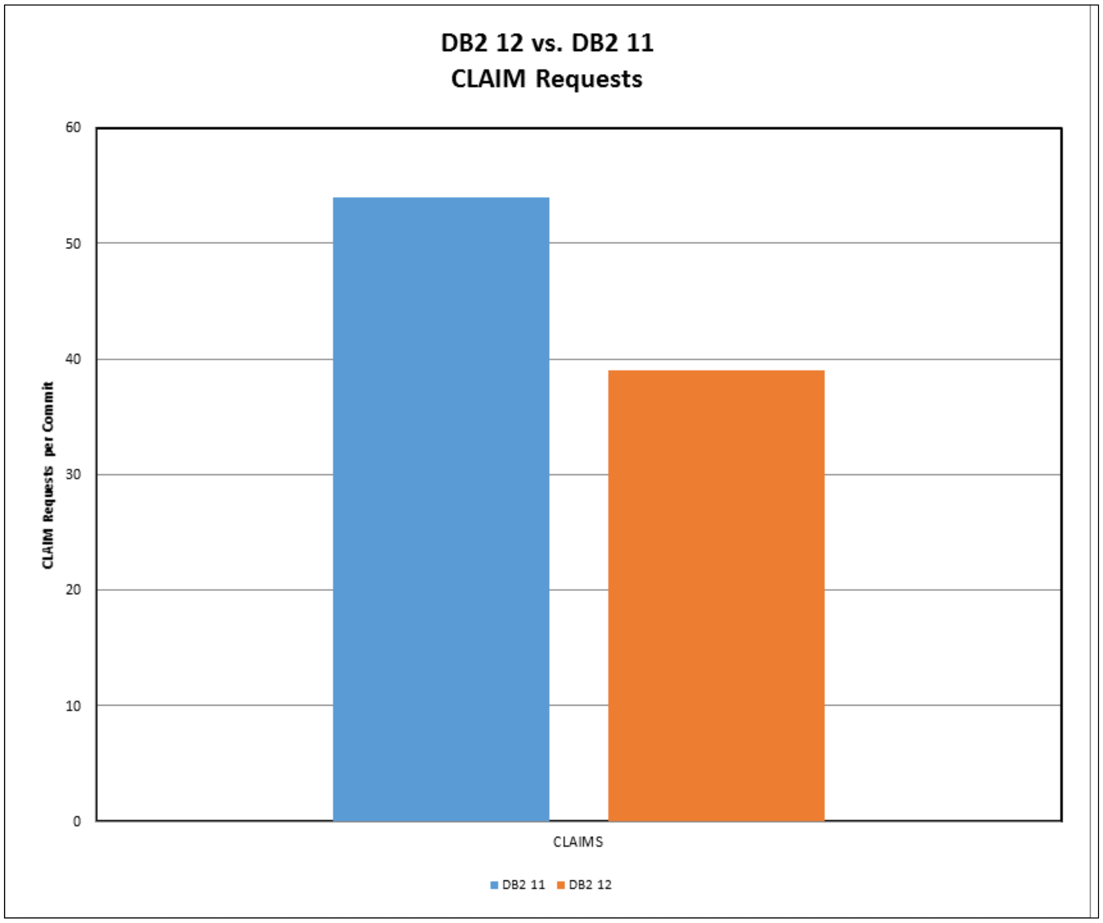


Figure 2-25 Claim requests reduction for the Brokerage workload run in Db2 12 versus run in Db2 11

2.8.3 SELECT INTO optimization

Db2 12 introduces a fast path mechanism for SELECT INTO SQL statements that reference an equal predicate with a unique index in the WHERE clause. This kind of SELECT statement always has a result set of one or zero rows. Such statements are categorized as singleton selects with unique index access.

Db2 12 provides a shorter code path to perform matching key index scan for this type of SELECT statement. By using this specialized code path, Db2 can avoid having to use the general runtime environment and the overhead that is associated with its ability to handle all variations of SELECT INTO statements.

All SELECT INTO statements with unique index access benefit from this optimization in Db2 12 at function level V12R1M100 and beyond. However, a REBIND is required to enable this “fast path” for SELECT INTO statements. The use of APREUSE(ERROR) is sufficient to allow Db2 to use the enhancement when applicable.

Performance measurement

To demonstrate the benefit of this new fast path SELECT capability, a batch program was used. It randomly selected rows by using SELECT INTO statements that have matching keys in unique indexes for predicates in the WHERE clauses.

The measurement results showed up to 3.6% Db2 class 2 CPU time reduction when the SELECT INTO optimization was used, compared to when the optimization was not used.

2.9 Db2 Storage Manager enhancements

In Db2 12, more components, such as EDM, LOB, and XML, take advantage of the Db2 Storage Manager facilities to manage the storage they use, instead of self-managing all the pieces of the storage they need.

2.9.1 Scalability and performance improvement for EDM pools

EDM pools are allocated with the defined size based on several system parameter values when Db2 is started. Before Db2 12, EDM pool management was performed in a way to use the pools as efficiently as possible without exceeding the defined size. This process involves frequent maintenance of storage usage chains for new EDM requests. As these EDM pool areas grow larger, they present some scalability challenges. As Db2 latch contention (LC24) becomes more notable, the cost to maintain these chains also grows, which causes EDM performance to degrade.

In Db2 12, the storage management of the EDM dynamic statement cache area, DBD pool, and skeleton pool is changed to avoid the frequent chain maintenance and remove the scalability inhibitors when larger EDM pools are used.

The size of these EDM pool areas is defined by the following system parameters:

- EDMSTMTC: Size limit for the dynamic statement cache pool
- EDMDBDC: Size limit for the DBD cache pool
- EDM_SKELETON_POOL: Size limit for the skeleton cache pool (for SKCTs and SKPTs)

Performance measurements

To evaluate these changes, several performance tests were conducted. For each of the EDM pool areas, specific measurements were performed.

Dynamic statement cache (DSC) pool

Two types of tests were run to evaluate the enhancement in Db2 12 related to the pool management of the dynamic statement cache.

Performance results for user created test scenarios

A special workload was created to observe Db2 12 performance when populating the EDM dynamic statement cache pool. The test scenario first primes the EDM statement cache pool with a series of different dynamic statements, making sure that the pool is fully used, and approximately 20% of the oldest dynamic statements were replaced.

Then, during the measurement period, the workload runs through the same dynamic statement execution cycle, which always triggers free and get activity for EDM dynamic statement pool storage. By using this scenario, the number of statements that were run were counted and the performance data was captured.

The workload was run with 10 concurrent threads. Multiple tests with EDMSTMTC set to 100 MB, 300 MB, 1000 MB, 2000 MB, and 3000 MB were run by using Db2 11 and Db2 12.

As shown in Figure 2-26, the use of a larger EDM statement cache significantly improves the dynamic statement external throughput rate (statements run per second) during the Db2 12 tests when compared against Db2 11. The fact that the throughput during the Db2 12 tests was constant (regardless of the EDMSTMTC value) proves that the enhancement performed well. In comparison, the throughput kept dropping with bigger EDMSTMTC values after the dynamic statement cache size was larger than 1000 M during the Db2 11 tests.

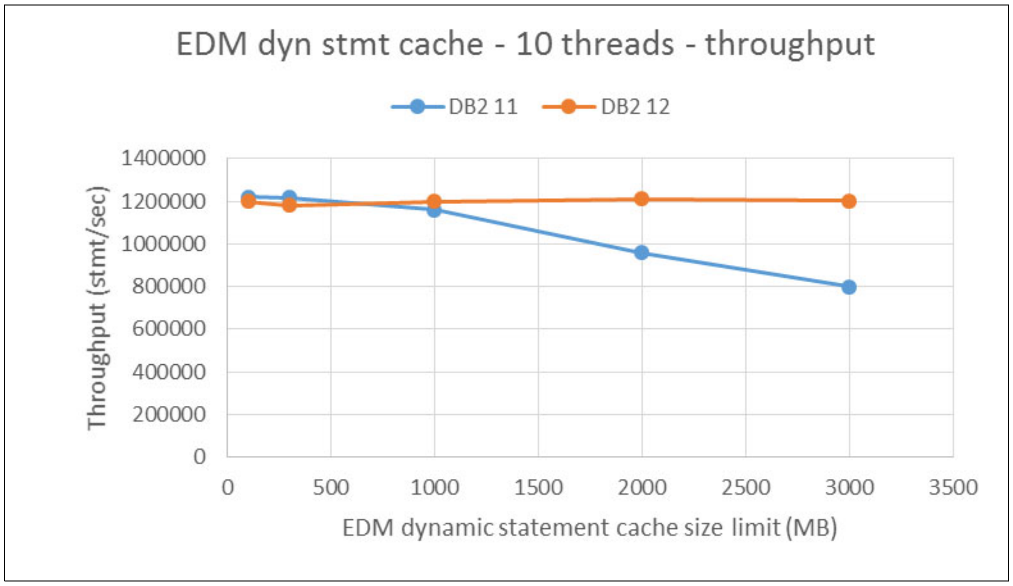


Figure 2-26 Effect of EDM dynamic statement cache pool size (limit) on throughput

Figure 2-27 shows that the latch class 24 (LC24) contention rate reduced significantly when comparing Db2 12 tests against the Db2 11 tests. The larger the dynamic statement cache pool size is, the bigger the actual reduction. Db2 12 almost eliminates the workload's LC24 contention, no matter how large the value of EDMSTMTC is. In Db2 11, it becomes more severe as the size of the statement cache pool increases.

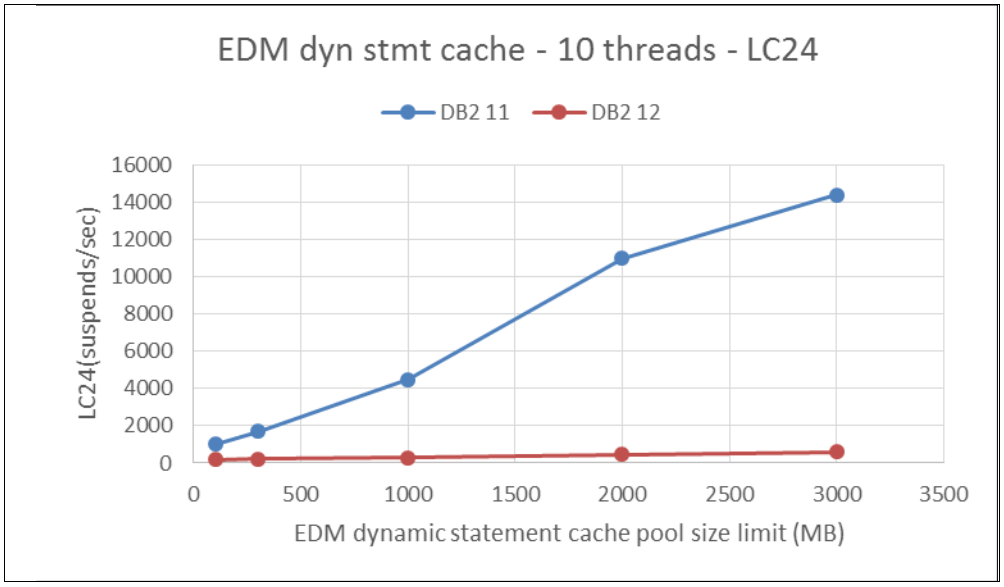


Figure 2-27 Effect of EDM dynamic statement cache pool size (limit) on LC24

Note: Approximately 10% more real storage usage was observed for the Db2 12 tests compared with the corresponding Db2 11 tests. This result is triggered by the changes in the way the DSC pool is managed, switching from the tightly packed storage size managed by EDM, to the discrete size of storage blocks managed by Db2 storage manager.

Performance results for distributed IRWW dynamic SQL workload

Performance measurements that used the IBM distributed IRWW dynamic SQL workload showed no CPU degradation after implementing this change in EDM storage management. This result is expected as the workload does not use many distinct SQL statements. However, a 2% increase in real storage usage was observed.

DBD cache pool

A special workload was created to evaluate Db2 12's capabilities to populate and manage the EDM DBD cache pool, comparing against Db2 11. The test first primes the EDM DBD pool with a set of different database objects, making sure that the pool is filled and approximately 20% of the oldest database objects in the DBD cache were replaced. Then, the same program is run again, referencing all the database objects in the same order, always triggering freeing up and acquiring storage in the EDM DBD cache.

Performance measurement data was gathered during the second run of the program. The tests were conducted several times by using EDMDBDC sizes of 10 MB, 50 MB, 100 MB, 500 MB, 1 GB, and 4 GB with Db2 11 and Db2 12 in a non-data sharing configuration.

The results showed that the workload's Db2 class 2 CPU time and the throughput rate that used Db2 12 are similar to Db2 11. CPU time and throughput were constant throughout the tests, no matter how large the DBD cache pool size limit was set. The real storage usage in Db2 12 increased up to 9% compared to Db2 11 real storage usage. The amount of the storage increase depended on the DBD pool size limit. It can be attributed to the differences in the way storage manager is managing the storage compared to EDM.

Skeleton cache pool

Also, for the EDM skeleton cache pool, a special workload was created to evaluate the performance of Db2 12's ability to populate and manage this EDM storage area. The workload first primes the EDM skeleton pool with a series of different packages, making sure that the EDM skeleton pool is fully used and approximately 20% of the packages were replaced in the pool. Then, the workload is run again to cycle through all packages, triggering freeing and obtaining storage in the EDM skeleton pool whenever a package is run.

Performance data was collected during the second run. The workload was run by using different EDM_SKELETON_POOL size values, varying 10 MB - 4 GB by using Db2 11 and Db2 12 in a non-data sharing environment in single-thread mode and running 50 concurrent threads.

For the single thread tests, the Db2 CPU time and throughput in Db2 12 are equivalent to Db2 11.

For the non-data sharing 50 concurrent threads tests, the following performance results were observed:

- Regarding the Db2 class 2 CPU time, as the skeleton pool size (limit) increases, CPU time performance in Db2 11 degrades significantly, while there was only a minor CPU time increase during the Db2 12 tests (see Figure 2-28). Compared to Db2 11, the Db2 12 tests showed up to 56% CPU time improvement when comparing against a run that used the same EDM_SKELETON_POOL value.

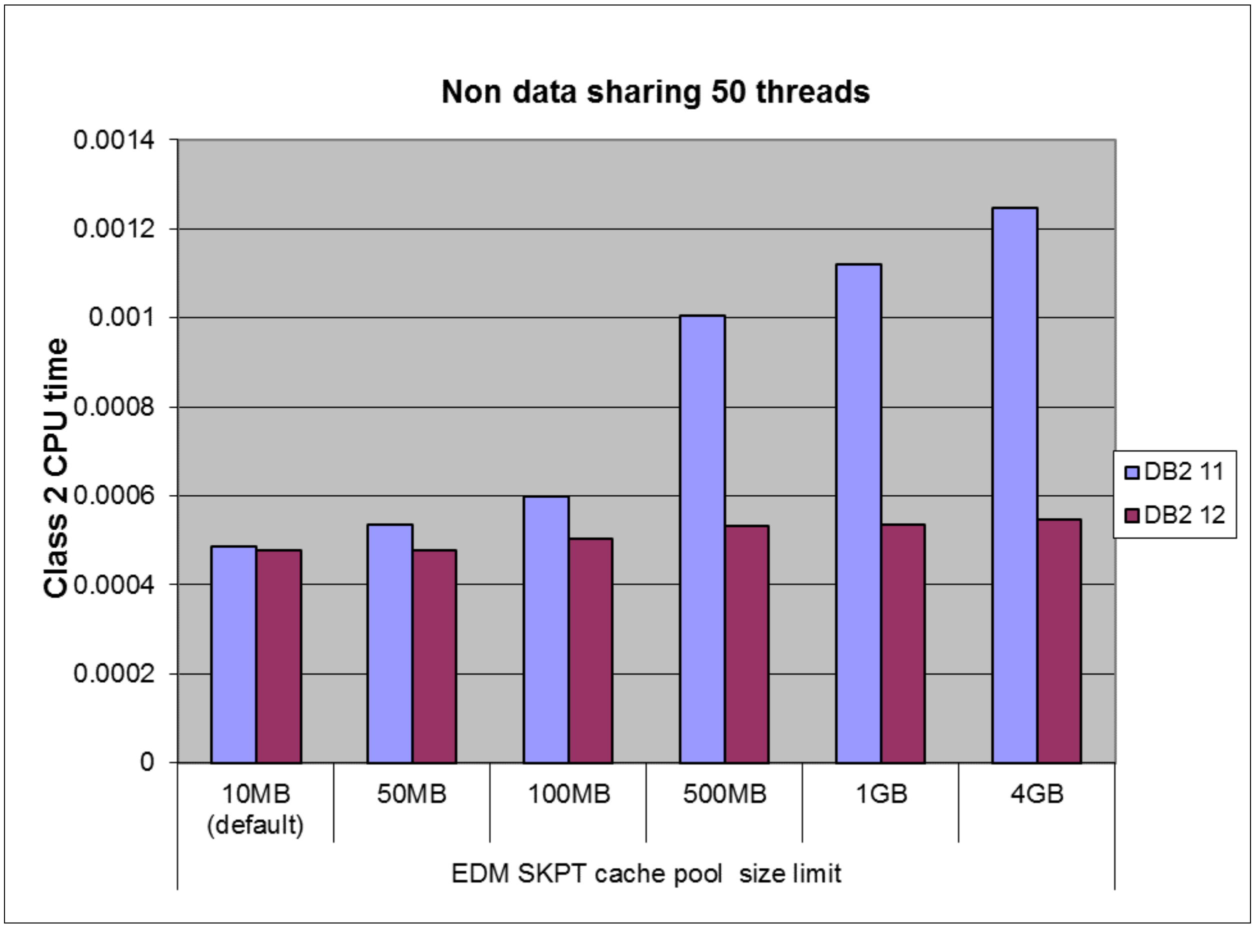


Figure 2-28 Effect of varying skeleton pool size on CPU time performance

- Regarding Db2 latch contention, the Db2 11 runs show severe latch contention (LC24) when the skeleton pool size was bigger than 500 MB. During the Db2 12 tests, the EDM latch contention (LC24) was close to non-existent for all different skeleton pool sizes, as shown in Figure 2-29.

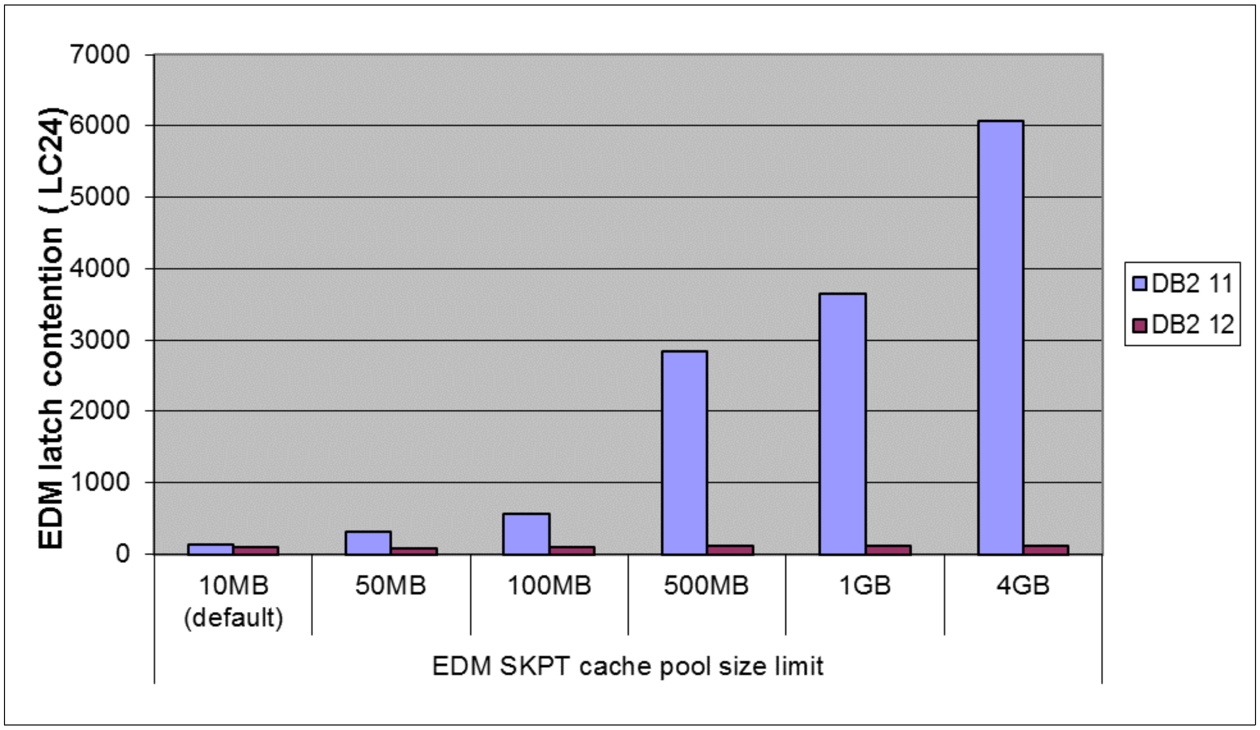


Figure 2-29 Effect of varying skeleton pool size on EDM latch contention

- A throughput improvement of up to 4% was observed in Db2 12.
- Db2 real storage usage in Db2 12 increased up to 6%.

In conclusion, Db2 12 significantly improves the scalability and performance for EDM pool management. The performance in Db2 12 is independent from the cache pool size limit. However, some increase in real storage usage is observed.

Because there is no performance change for a pool size up to 50 MB in Db2 12, and many customers can benefit from a larger skeleton pool, the default size limit for the skeleton pool is increased from 10 MB to 50 MB in Db2 12.

2.9.2 Storage pool management improvement for XML and LOBs

Before Db2 12, the LOBVALA, LOBVALS, XMLVALA, and XMLVALS DSNZPARM settings are used to specify the maximum amount of storage that can be used by an agent (thread), or by the entire subsystem for LOB and XML storage. In the past, it was difficult for customers to determine the proper values for these DSNZPARM parameters.

Db2 12 changes the XML/LOB storage management design to take advantage of the centralized and improved real and virtual storage management capabilities of the Db2 storage manager component. With the new design, the LOB and XML storage-related DSNZPARM parameters listed were removed so customers no longer are concerned about specifying appropriate values for their systems. Instead, the Db2 storage manager component assumes responsibility for monitoring the use of the LOB/XML related storage and controls the storage allocation, contraction, release of the required storage in a more effective way. This enhancement is available at function level V12R1M100.

In Db2 12, the individual LOB/XML storage pools that are used by all threads are consolidated into a single 4 GB storage pool, which is managed by the Db2 Storage Manager component.

By consolidating into a single pool, performance is improved in the storage management code path of the LOB/XML Manager component. In addition, commit processing manages only storage in a single storage pool rather than having to consider contracting several individual storage pools, which also improves efficiency.

2.10 Miscellaneous enhancements

The last two enhancements that are described in this section have no externals but can still bring significant performance improvements.

2.10.1 zPLX 3.2.0 Performance enhancement

Most of the Db2 code is written in a programming language called PL/X.

Before Db2 12, Db2 modules were compiled with PL/X 2.4.2. zPLX 3.2.0 is the first release of the zPLX compiler to support AMODE 64. The AMODE 64 version of compiler can use more memory, which allows for successfully compiling larger modules.

It also eases certain limitations on optimizations that were sometimes encountered because of memory constraints by the previous releases of the compiler. In addition, scalability of the compiler is improved with its 64-bit back-end and new options are introduced in support of the latest IBM z13 hardware.

By using several internal Db2 performance workloads, including OLTP, queries, batch, utility, and some customer workloads, the top 221 performance-sensitive Db2 modules were recompiled by using the zPLX 3.2.0 compiler to use its ability to produce more optimized code for these Db2 load modules. The other Db2 modules continue to use PL/X 2.4.2.

When the zPLX 3.2.0 compiler became generally available, several internal benchmarks were run. The performance improvement for the IBM Brokerage OLTP workload was 4.4% of the total Db2 CPU time. The total CPU time includes Db2 class 2 CPU time, and MSTR, DBM1, and IRLM CPU times. The SAP banking workload showed a 2.7% performance improvement. The TPC-D query workload showed a 4.5% reduction in class 1 CPU time.

2.10.2 Latch class 23 reduction

Although this enhancement is not related to the Db2 logging environment, it is described here because it also reduces Db2 latch contention.

Latch Class 23 (LC23) is used by the Db2 Buffer Manager component to serialize a number of different types of events. One is related to serializing access to information that is related to (buffer manager) page latch contention. As a result, when high page latch contention is observed in a certain workload, more high LC23 contention occurs, which aggravates the situation.

During the Db2 12 development process, this code path was reexamined and the cases where storing the extra page latch information was not necessary were eliminated. Therefore, in addition to the efforts to reduce page latch contention (for example, by using the Db2 12 Insert Algorithm 2) even during scenarios that still show high page latch contention, the LC23 contention that often resulted from that effort was greatly reduced or eliminated.

3

Synergy with the IBM Z platform

As with previous Db2 versions, Db2 12 for z/OS takes advantage of the latest improvements that are available on the IBM Z platform. Db2 12 increases its synergy with z Systems hardware and software to provide better performance, stronger resilience, and better functions for an overall improved value.

Db2 12 also adds more z Systems Integrated Information Processor (zIIP) eligibility, and can use zEDC cards to enable LOB compression.

This chapter includes the following topics:

3.1 Large object compression

In Db2 Version 3, data compression for data in table spaces was introduced and uses the Lempel-Ziv algorithm with a dictionary. In Db2 Version 9, index compression capability was added. Data in XML table spaces, introduced in V9, can also be compressed with the same technique that is used by standard table space compression.

In Db2 12, large objects (LOBs) also can be compressed. Character LOB (CLOB), binary LOB (BLOB), and double byte CLOB (DBCLOB) data types are supported for compression.

LOB data compression can also be enabled on all LOB table spaces that are associated with the Db2 directory (SYSSPUXA and SYSSPUXB, which is associated with SPT01, and SYSDBDXA, which is associated with DBD01). When the new COMPRESS_DIRLOB DSNZPARM is set to YES (NO is the default), the LOB table spaces in the Db2 directory are compressed the next time that they are reorganized.

LOB compression in Db2 12 is not dictionary-based, as with table space compression. Instead, it uses the zEDC hardware card that has a different compression algorithm.

Good candidates for LOB compression are LOB columns that store data in formats, such as `.docx`, `.txt`, XML, and HTML.

In Db2 12, you can use DSN1COMP to provide an estimation of the compression ratio. To run DSN1COMP against the LOB table spaces that are not yet compressed, you must have a machine with a zEDC express card installed.

3.1.1 LOB compression prerequisites

Before LOBs can be compressed, the following conditions must be met:

- The Db2 12 system must be operating at active function level V12R1M500 or above. Otherwise, the COMPRESS YES attribute for a LOB table space in a DDL statement cannot be specified.
- The base table space that is associated with the LOB table space must be a Universal Table Space (UTS).
- The compression attribute of LOBs can also be inherited from their base tables. Consider the following points:
  - For an implicitly created LOB table space with a base table space that is a partition-by-range universal table space, its compression attribute is inherited from the associated partition of the base table.
  - For a partition-by-growth (PBG) base table space, the compression attribute of the LOB table space is inherited from its associated base partition when the partition is created initially. For any other LOB table space that is added because of the growth partition of PBG table space, its compression attribute is inherited from the previous partition's LOB table space.
- To enable the LOB compression feature, the system must have the IBM zEnterprise Data Compression (zEDC) capability and the Peripheral Component Interconnect Express (PCIe or PCI Express) hardware adapter that is called zEDC Express installed. The use of zEDC requires that the following prerequisites are met:
  - z/OS V2R1 (or higher) operating system.
  - IBM zEnterprise EC12 (with GA2 level microcode) or IBM zEnterprise zBC12.

  - zEDC Express feature. This System z compression accelerator card can improve the speed of data compression and is sharable across up to 15 partitions and up to eight cards per CPC.
  - zEDC Express software is enabled in an IFAPRDxx parmlib member.

  For more information about the zEDC hardware card and its specifications, see *Reduce Storage Occupancy and Increase Operations Efficiency with IBM zEnterprise Data Compression*, SG24-8259.

  For a Db2 data sharing environment, it is highly recommended to make the zEDC hardware available to all LPARs of the data sharing group to prevent any performance effect to the system. This suggestion also applies to the disaster recovery site. The zEDC hardware is not validated when DDL CREATE ALTER TABLESPACE statements are run.

  During DML operations, Db2 processes the LOB data that is based on the compression attribute that is specified for the LOB table space. If the zEDC hardware is present in the system, the data is treated as compressed if the LOB table space has the COMPRESS YES attribute. If the zEDC card is not present, the data cannot be compressed and is stored uncompressed, despite the LOB table space's COMPRESS YES attribute.

  In a data sharing environment, it is possible that not all the data sharing members include a zEDC card installed. In that case, it is still possible for a member without a zEDC card to retrieve the compressed data. The data is decompressed by using a software algorithm. However, the use of software decompression severely affects the system performance and is not a recommended setup.

- The LOB data must be bigger than its defined page size to be compressed. If the total length of the LOB is less than the defined page size, the LOB data is not compressed because a single LOB page cannot be used by multiple LOBs and it is not worthwhile to compress a LOB smaller than a page. Therefore, regardless of the COMPRESS attribute of the LOB table space, such LOBs are not compressed.

3.1.2 LOB compression considerations

The following considerations must be taken when LOB compression is implemented as several areas can be affected:

- Possible overhead to compress the LOB during insert
- Decompression overhead when the data is selected
- Effect on the amount of DASD storage that is required

Lab tests show that compression ratios for certain CLOBs can range 50% - 96%, with an increase in INSERT throughput of ~80% and a SELECT throughput increase of ~250% in certain cases.

The following LOB compression considerations are described in this section:

- File formats suitable for LOB compression
- Compression ratios

File formats suitable for LOB compression

Certain file formats are not eligible for compression by using the zEDC card. When a file is compressed in its original format, no benefit is gained from compressing it again before inserting into a Db2 (LOB) table space.

The following file types are typically stored as binary LOBs and created such that they are compressed:

- JPG
- PDF (parts can be compressed; other parts might not be compressed)
- PNG
- PPT
- XLSX (compressed XLS format)

Attempting to compress these types of file formats causes the zEDC hardware to ignore the compression. Db2 and zEDC recognize that no benefit is gained in compressing these types of files.

Compression ratios

During the evaluation of the LOB compression feature, several different file types were used. The following compression ratios were observed:

- DOCX: Up to 73% pages were saved.
- JPG: 0% pages saved (as JPGs are already compressed).
- XML: Up to 96% pages saved (XML documents that are stored as LOBs, not XML documents that use the XML data type).
- Db2 directory: Up to 83% pages saved.

Each of these file types were tested by using a configuration that includes 2 zEDC cards and they all each use a separate LOB table space.

These compression ratios are only an indication of what might be achieved when these file types are stored in a compressed LOB table space. The compression ratio varies greatly depending on the contents of files that are stored.

Note: Pages in a LOB table space cannot be shared between two LOBs (regardless of whether they are compressed). For example, if two LOBs from the same LOB table space are 4 KB each (even when the table space has an 8 KB page size), Db2 cannot store both LOBs into the same page. They take up two separate (8 KB) pages. Considering this point, when a LOB’s size is smaller than the LOB table space’s page size, Db2 does not compress it because compressed or not, the LOB uses one page to be stored. For example, a 7 KB LOB that uses a LOB table space with an 8 KB page size is not compressed.

3.1.3 Use of DSN1COMP to estimate LOB compression

DSN1COMP is a stand-alone utility that can be used to estimate space savings that can be achieved with Db2 data compression for the evaluated table spaces and indexes.

With the introduction of LOB compression in Db2 12 that uses zEnterprise Data Compression (zEDC) hardware, a new option, PARM=’LOB’, was added to the DSN1COMP utility to estimate the compression space savings. This option also estimates the compression ratio for LOB table spaces if they are to be compressed.

If the LOB table space is uncompressed, the utility makes a single pass through the LOB table space and collects data up to the average LOB size (or 1 MB as the maximum size). If zEDC hardware is present, the collected data is sent to the hardware card to be compressed and statistics are returned for DSN1COMP to produce the report.

If the LOB table space is compressed because it is defined with the COMPRESS YES option, DSN1COMP collects statistics of the individual LOB metadata from the LOB map pages and produces a report without collecting and compressing data by using the zEDC card again.

Db2 does not compress LOBs that are less than the current page size that is used for the object. In addition, different types of files have different compression ratios. Therefore, it is useful to estimate how much your LOBs can be compressed by checking the compression ratio and data pages that are saved by using the DSN1COMP utility before compression on those LOBs is implemented.

Db2 does not support compressing individual pieces of a linear data set. Also, because LOB table spaces are linear page sets, compression applies to the entire (non-partitioned) LOB table space. Therefore, all of the pieces (A001, A002, and so on) should be evaluated for compression savings by using DSN1COMP. However, for DSN1COMP, it is sufficient to specify only the A001 data set on the SYSUT1 DD-statement. Db2 finds the other pieces if they exist.

A non-partitioned table space must contain all pieces when an image copy of the LOB table space is used as input to DSN1COMP.

For a partitioned table space, the image copy can contain a single partition, a subset of partitions, or all partitions.

A sample JCL statement that can be used to estimate space savings for a (currently uncompressed) LOB table space that uses DSN1COMP is shown in Example 3-1. Excerpts from the report that is produced also are shown.

Example 3-1 Sample output for uncompressed LOB table space

```
JCL:
//COMP1    EXEC PGM=DSN1COMP,PARM='LOB'
//SYSUT1 DD DISP=SHR,DSN= DSNC000.DSNDBC.MYLOBDB.MYLOBTS1.I0001.A001
Output:
DSN1999I START OF DSN1COMP FOR JOB DSN1CMP  STEP2
DSN1998I INPUT DSNAME = DSNC000.DSNDBC.MYLOBDB.MYLOBTS1.I0001.A001 , VSAM
DSN1944I DSN1COMP INPUT PARAMETERS
                  PROCESSING PARMS FOR LOB DATASET:
                  NO PARAMETERS.
DSN1940I DSN1COMP COMPRESSION REPORT
 DSN1COMP run on uncompressed LOB table space
 LOB table space statistics
 --------------------------------------------------------
 Number of LOBs                                         200 LOBs
 Minimum LOB size                                         3 KB
 Maximum LOB size                                        28 KB
 Average LOB size                                        25 KB
 LOB compression ratio
 --------------------------------------------------------
 Total LOB data compressed                            2,195 KB
 Total LOB data uncompressed                          5,143 KB
 Percentage of KB saved                                  58 %
 Data pages needed for compressed LOB table space       371 Pages
 Data pages needed for uncompressed LOB table space   1,090 Pages
 Percentage of Data pages saved                         66 %
 Current page size                                       4 KB
 EVALUATION OF COMPRESSION WITH DIFFERENT PAGE SIZES
 Note: System Pages may contain LOB data!
 ---------------------------------------------------------
4K page size:
   Minimum System pages required                        221 Pages
   Additional Data pages needed for compressed LOBs     371 Pages
   Additional Data pages needed for uncompressed LOBs 1,090 Pages
   Data pages saved (not including system pages)        66 %
 8K page size:
   Minimum System pages required                        221 Pages
   Additional Data pages needed for compressed LOBs     191 Pages
   Additional Data pages needed for uncompressed LOBs   540 Pages
   Data pages saved (not including system pages)        65 %
 16K page size:
   Minimum System pages required                        221 Pages
   Additional Data pages needed for compressed LOBs      11 Pages
   Additional Data pages needed for uncompressed LOBs   180 Pages
   Data pages saved (not including system pages)        94 %
 32K page size:
   Minimum System pages required                        221 Pages
   Additional Data pages needed for compressed LOBs       0 Pages
   Additional Data pages needed for uncompressed LOBs     0 Pages
   Data pages saved (not including system pages)         0 %
DSN1994I DSN1COMP COMPLETED SUCCESSFULLY,   2,520 PAGES PROCESSED
```

The DSN1COMP output that is shown in Example 3-1 on page 87 indicates that the utility was run against an uncompressed LOB table space.

The “LOB table space statistics section” that is shown in Example 3-1 on page 87 provides a summary about the number of LOBs and the minimum, maximum, and average LOB size. As shown in Example 3-1 on page 87, the average LOB size of 25 KB is close to the maximum LOB size of 28 KB. Therefore, it can be assumed that most of the LOBs are much larger than the current page size of 4 KB. If so, we can expect some benefit from the use of compression.

The “LOB compression ratio section” that is shown in Example 3-1 on page 87 shows that the LOB compression ratio is 58%. However, we want to know how many pages we can save. As LOB pages cannot be shared by multiple LOBs, a good compression ratio of the actual LOBs does not necessarily mean that we can save an equal number of data pages. In the case of Example 3-1 on page 87, Db2 estimates that it can reduce the number of pages by 66% by compressing the LOB table space by using the current page size of 4 KB.

In addition to evaluating the LOB compression by using the current page size, DSN1COMP calculates how many pages might be saved if another page size were used. This information is provided in the “EVALUATION OF COMPRESSION WITH DIFFERENT PAGE SIZES” section of Example 3-1 on page 87. As shown in this example, the use of a 16KB page size for the LOB table space results in a reduction in the number of LOB data pages by 94%.

Note: The calculation provides the percentage of saved *LOB data pages*. The LOB map pages, which can also contain LOB data, are not included in this calculation. LOB map pages are part of the number of *system pages* in the output.

Also, as shown in Example 3-1 on page 87, no space savings is expected when a 32 KB page size is used. As LOBs cannot share their data pages and the maximum LOB size in this case is 28 KB, no LOBs in the table space is compressed under the 32 KB page size.

3.1.4 Performance characteristics of LOB compression

Several measurements were conducted to evaluate the Db2 12 LOB compression feature, with a focus on select and insert performance, by using throughput rate and CPU performance as the main performance indicators.

Throughput performance for insert and select with compressed LOBs

When the measurements to evaluate the Db2 12 LOB compression feature were taken, significant throughput benefits were observed. Because the number of LOB pages can be reduced when the LOB is compressed, SQL INSERT activity requires fewer writes to the LOB table space. Also, fewer getpage requests are needed when data is retrieved from the LOB table space. In certain cases, this process can result in a large increase in throughput performance for INSERT and SELECT activities.

The tests were conducted by using the following measurements:

- Different numbers of concurrent users (10, 20, 40, 80, and 100 threads)
- Different LOB sizes of 250 KB and 500 KB
- Different types of LOBs (XML and JPG)

During these performance measurements, we only considered the CLOB XML data because JPG data cannot be compressed. These tests run sequential inserts and randomized selects by using a generic key. We observed that with compression, the size of the documents can be greatly reduced (with compression ratio of ~96%).

Insert activity

The throughput rates for the insert workload (in GB per second) by using different LOB sizes and a varying number of concurrent threads is shown in Figure 3-1. The largest boost to performance occurs when 500 KB LOBs are compressed compared to non-compressed LOBs. When 20 threads are used, the throughput rate in GB per second is approximately 80% more with compressed 500 KB LOBs compared to non-compressed 500 KB LOBs.

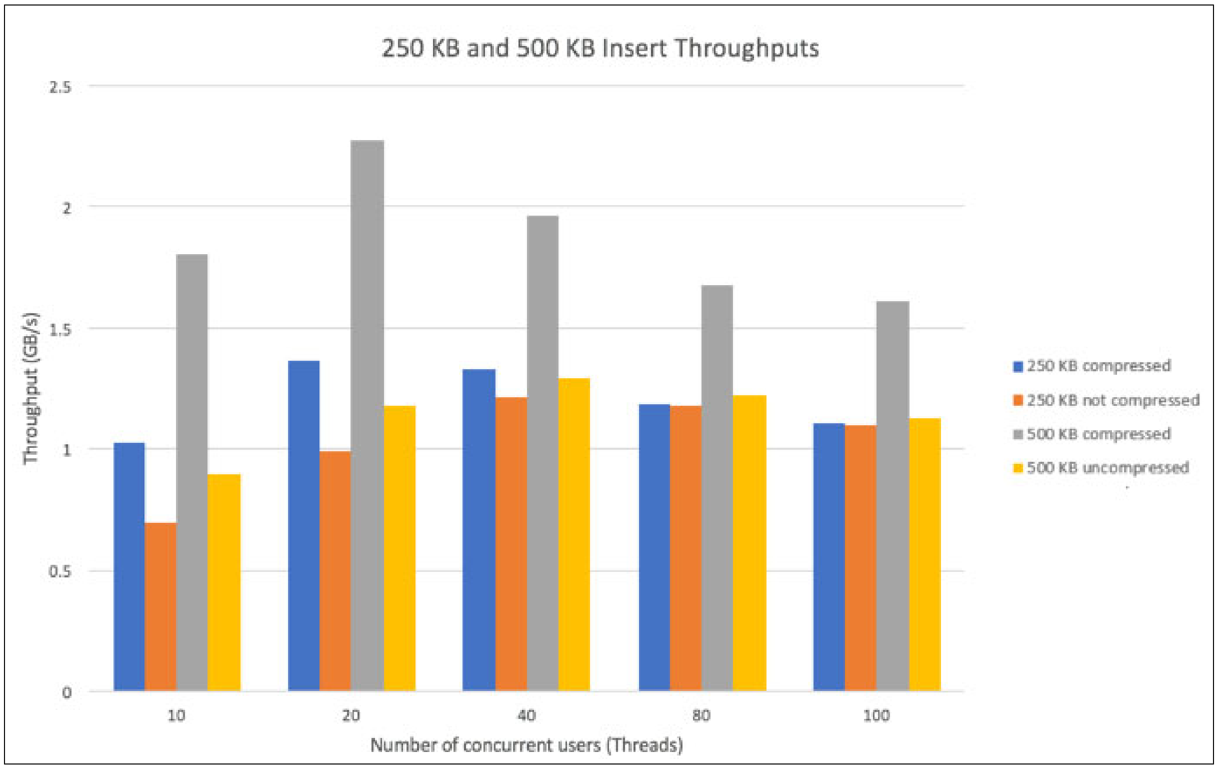
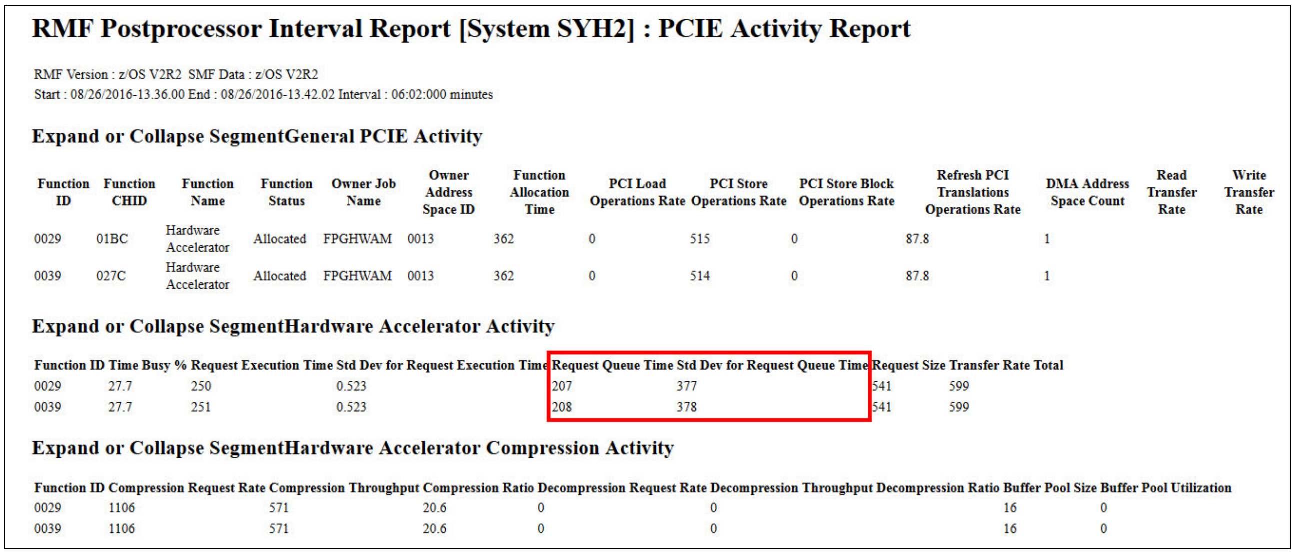


Figure 3-1 250 KB and 500 KB insert throughputs comparison

Counter intuitively, the throughput seems to decrease as the number of threads increases. At 20/40 threads, the throughputs of the 250 KB and 500 KB compressed LOBs start to degrade. However, the LOB insert throughput rate for non-compressed LOBs does not degrade until 40/80 threads. This result shows the hardware limitations of the zEDC card.

At ~20 - 40 concurrent users, the queue times on the zEDC card (in the IBM RMF™ reports) start to become a significant factor. The benefit of the savings from smaller numbers of page writes is diminished by the longer zEDC queue times as the zEDC hardware hits its bandwidth limitation.

When SMF record type 74 subtype 9 data is collected, RMF Monitor III gathers the data that is required for the PCIE Activity Report. You can use the RMF postprocessor to produce a report about the PCIe activity by using the REPORTS(PCIE) report option. The information about the Request Queue Time is in the "Hardware Accelerator Activity section of the report". A sample output of the report is shown in Figure 3-2 on page 91.



RMF Postprocessor Interval Report [System SYH2] : PCIE Activity Report

RMF Version : z/OS V2R2 SMF Data : z/OS V2R2
Start : 08/26/2016-13.36.00 End : 08/26/2016-13.42.02 Interval : 06:02:000 minutes

Expand or Collapse SegmentGeneral PCIE Activity

Function ID	Function CHID	Function Name	Function Status	Owner Job Name	Owner Address Space ID	Function Allocation Time	PCI Load Operations Rate	PCI Store Operations Rate	PCI Store Block Operations Rate	Refresh PCI Translations Operations Rate	DMA Address Space Count	Read Transfer Rate	Write Transfer Rate
0029	01BC	Hardware Accelerator	Allocated	FPGHWAM	0013	362	0	515	0	87.8	1		
0039	027C	Hardware Accelerator	Allocated	FPGHWAM	0013	362	0	514	0	87.8	1		

Expand or Collapse SegmentHardware Accelerator Activity

Function ID	Time Busy %	Request Execution Time	Std Dev for Request Execution Time	Request Queue Time	Std Dev for Request Queue Time	Request Size	Transfer Rate Total
0029	27.7	250	0.523	207	377	541	599
0039	27.7	251	0.523	208	378	541	599

Expand or Collapse SegmentHardware Accelerator Compression Activity

Function ID	Compression Request Rate	Compression Throughput	Compression Ratio	Decompression Request Rate	Decompression Throughput	Decompression Ratio	Buffer Pool Size	Buffer Pool Utilization
0029	1106	571	20.6	0	0		16	0
0039	1106	571	20.6	0	0		16	0

Figure 3-2 Sample RMF PCI activity report

Information about zEDC processing for LOB compression and decompression is also recorded in Db2 accounting records. Accounting class 3 uses two new LOB COMPRESSION counters: QWAX_LOBCOMP_WAIT and QWAX_LOBCOMP_COUNT. These counters record the wait time and the number of times the thread waited for LOB compression to be performed by the zEDC card. An excerpt of the OMPE accounting report class 3 wait information (including the new LOB compression wait information) is shown in Example 3-2.

Example 3-2 Db2 accounting class 3 LOB compression wait time

```
CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT  TIME/EVENT
--------------------  ------------  --------  ----------
LOCK/LATCH(DB2+IRLM)      0.003751      0.84    0.004456
 IRLM LOCK+LATCH          0.000083      0.02    0.003970
 DB2 LATCH                0.003667      0.82    0.004468
SYNCHRON. I/O             0.003553      1.00    0.003542
 DATABASE I/O             0.000009      0.00    0.002945
 LOG WRITE I/O            0.003544      1.00    0.003544
OTHER READ I/O            0.000002      0.00    0.003301
OTHER WRTE I/O            0.000030      0.01    0.004748
SER.TASK SWTCH            0.000139      0.00    0.072344
 UPDATE COMMIT            0.000000      0.00         N/C
 OPEN/CLOSE               0.000010      0.00    0.091700
 SYSLGRNG REC             0.000000      0.00         N/C
 EXT/DEL/DEF              0.000127      0.00    0.074100
 OTHER SERVICE            0.000002      0.00    0.022306
ARC.LOG(QUIES)            0.000000      0.00         N/C
LOG READ                  0.000000      0.00         N/C
DRAIN LOCK                0.000000      0.00         N/C
CLAIM RELEASE             0.000000      0.00         N/C
PAGE LATCH                0.000002      0.00    0.003901
NOTIFY MSGS               0.000000      0.00         N/C
GLOBAL CONTENTION         0.000000      0.00         N/C
COMMIT PH1 WRITE I/O      0.004439      1.00    0.004439
ASYNCH CF REQUESTS        0.000000      0.00         N/C
TCP/IP LOB XML            0.003463     16.00    0.000216
ACCELERATOR               0.000000      0.00         N/C
```

```
AUTONOMOUS PROCEDURE        0.000000      0.00          N/C
PQ SYNCHRONIZATION          0.000000      0.00          N/C
LOB COMPRESSION             0.003906      1.02     0.003829
FAST INSERT PIPE            0.000000      0.00          N/C
TOTAL CLASS 3               0.019285     19.87     0.000970
```

Select activity

The results from the measurements of the workload that performs SELECT operations against compressed and non-compressed LOBs with different sizes and a varying number of concurrent threads show a similar trend of large throughput benefits for SELECT operations, as shown in Figure 3-3.

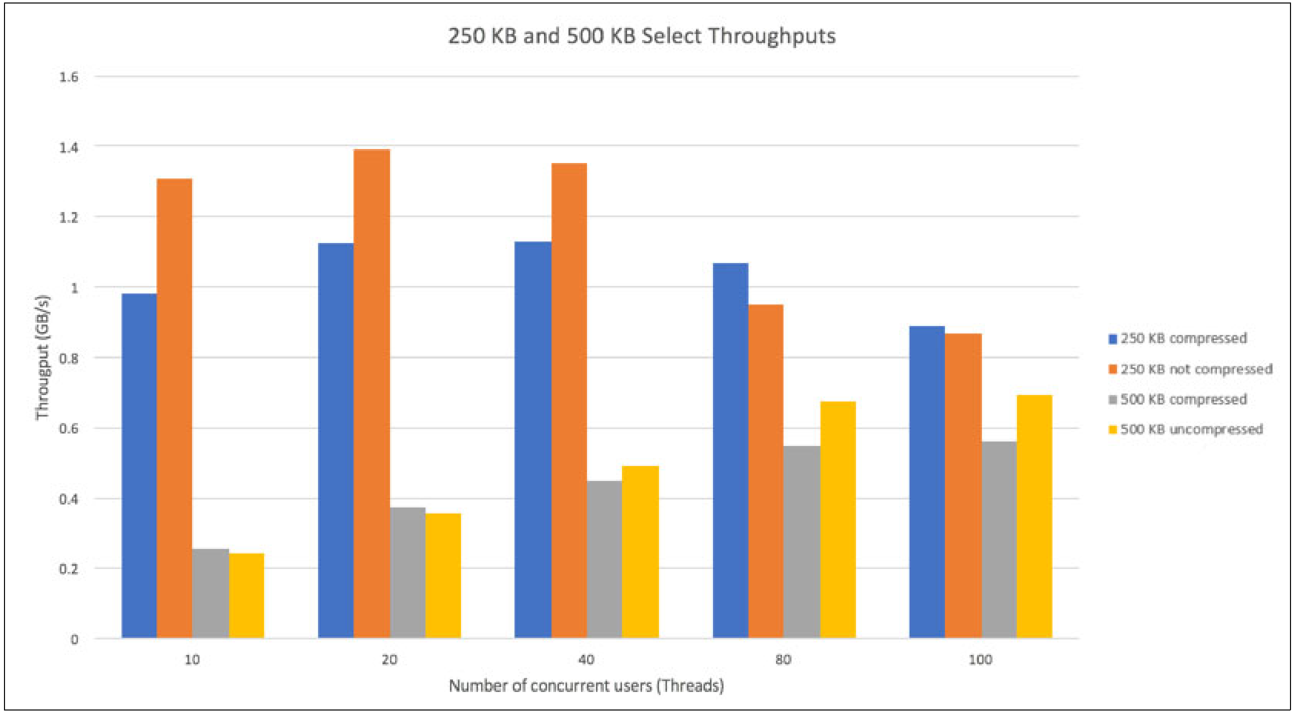


Figure 3-3   250 KB and 500 KB select throughputs comparison

It is worth noting that the overall throughput is comparatively lower for the select case when compared against the measurements that are inserting LOBs. This result is due to (in part) the random nature of the select workload compared to the sequential nature of the insert workload.

In this scenario, you can see a similar trend where the throughput rate seems to degrade after 20/40 threads for compressed LOBs.

CPU performance for insert and select with compressed LOBs

The zEDC card promises highly efficient compression of LOBs without an affect on CPU performance. This compression is reflective of the CPU performance that was measured in the same environment that was described in 3.1.2, “LOB compression considerations” on page 85.

Because the zEDC card allows Db2 to rely on the hardware for compression, we did not observe any significant increase in Db2 class 2 CPU time. This result is true for select and insert performance. We noted minor CPU time differences 0.00012 - 0.00014 seconds in the worst-case scenarios (by using 500 KB LOBs with 80 concurrent threads).

3.2 ISM that uses SMC-D

The IBM z13 GA2 and IBM z13s™ introduces an Internal Shared Memory (ISM) virtual PCI function. ISM is a virtual PCI network adapter that enables direct access to shared virtual memory, which provides a highly-optimized network interconnect for z Systems intra-CPC communications.

When ISM is used by z/OS Version 2.2 by using Shared Memory Communications - Direct Memory Access (SMC-D), the combined solution provides improved transaction rates for interactive (request and response) workloads. This improvement is due to reduced network latency and lower CPU cost for workloads with larger payloads, such as analytics, streaming, big data, or web services. The combined solution also preserves the key qualities of services that are required by z System clusters in enterprise data center networks.

Customers who have multiple z/OS instances on a single CPC with z/OS centric workloads that use TCP communications, such as Db2, IBM WebSphere® Application Server, IBM CICS®, IBM MQ, and IMS among them are the primary beneficiaries of this new support.

3.2.1 HiperSockets, SMC-R, and SMC-D

The target audience of this IBM Redbooks publication are mainly users who work in the Db2 for z/OS area. For the benefit of those users that are not deeply involved in the communication server and hardware aspects, we provide some background in this section to where SMC-D fits into the larger picture.

IBM z Systems HiperSockets

Co-location of multiple tiers of a workload onto a single z Systems platform allows for the use of IBM z Systems internal communications. IBM z Systems HiperSockets™ is an internal LAN technology that provides low-latency connectivity between virtual machines within a physical z Systems central processing complex (CPC).

HiperSockets is logically represented to the host as a standard network interface controller (NIC) with access to an IEEE standard LAN. It is implemented fully within z Systems firmware, so it requires no physical cabling or external network connection to purchase, maintain, or replace.

The performance benefits of HiperSockets when compared to the use of an external LAN connection over Open Systems Adapter-Express (OSA-Express) are achieved in large part because the physical layer (Ethernet) processing is avoided.

However, for the plethora of applications that use TCP/IP for network communications, the communications and packet-processing aspects of the software stack remain. Data is formed into, transmitted, and received in fully compliant TCP/IP packets. Traditional IP routing is also used.

All standards that are relevant to TCP/IP processing, such as checksum, segmentation, flow control, reordering, and retransmission, are used when communicating over HiperSockets. This processing occurs at the appropriate operating system communication stack software layers. A significant amount of overhead is also incurred with the associated interrupt processing, memory management, context switching, and thread dispatching.

Shared Memory Communications over RDMA

The IBM zEnterprise EC12 system, IBM zEnterprise BC12 system, and IBM z/OS V2.1 introduced a new optimized communications protocol, Shared Memory Communication (SMC) over Remote Direct Memory Access (RDMA) (SMC-R), which works with the IBM 10GbE RoCE Express feature. SMC-R maintains the TCP/IP socket application programming interface (API) for applications while significantly reducing the central processing unit (CPU) overhead of the networking software stack.

Application data is transferred by using direct memory-to-memory communication. SMC-R uses TCP to establish, manage, and end connections along with connection monitoring functions, such as traditional keep-alive processing. The TCP connection remains active while dynamically transitioning to exchange data out-of-band by using (RDMA), in a manner that is not apparent to applications.

It also does not require any IP topology changes or other definitions because the same IP interface can be used to support standard TCP/IP and SMC-R. This model preserves critical operational and network management features of TCP/IP, while providing significant latency, throughput improvements, and CPU savings.

Although SMC-R and RoCE Express can also be used for logical partitioning (LPAR) communication (LPAR-to-LPAR), this solution was aimed primarily at optimizing communication between distinct physical servers. SMC-R provides HiperSockets-like performance between servers.

Although HiperSockets is widely accepted and deployed because of the latency and throughput benefits that are realized in comparison to communications over OSA, the chief inhibitor to further deployment and the greatest criticism is the CPU overhead. Data movement, stack packet processing, and driver processing that is related to fully implementing the internal queued direct input/output (iQDIO) architecture are all incurred under the host's processors.

Technologies, such as RDMA, and protocols, such as SMC-R, closed the gap, which virtually eliminates the advantages that are gained in transaction and throughput rates that were once seen when HiperSockets were used on the same CPC as compared to communicating off-platform between physical servers.

SMC over Direct Memory Access

If SMC-R that uses RDMA can improve the transaction throughput and CPU consumption rates (which was a major drawback when HiperSockets were used) for unchanged TCP/IP applications for cross-CPC workloads, it makes logical sense that a similar protocol that is based on Direct Memory Access (DMA) can provide comparable benefits for inter-processor (LPAR to LPAR) communications.

The approach of the use of the SMC protocol (bypassing TCP/IP for data movement) without requiring the cost and complexity of more hardware solidifies the advantages of co-locating multiple tiers of a workload onto a single z Systems platform. Therefore, the IBM z13 (and z13s) platform brings new ISM virtual PCIe (vPCIe) devices.

ISM architecture enables optimized cross-LPAR TCP communications utilizing a new sockets-based DMA protocol named Shared Memory Communication - Direct Memory Access (SMC-D). ISM functions must be associated with another channel (CHID); that is, a HiperSockets channel (IQD) or an OSA Express (OSD) channel. SMC-D maintains the socket-API transparency aspect of SMC-R so that applications that use TCP/IP communications can benefit immediately without requiring any application software or IP topology changes.

SMC-D tightly couples the client and server sockets, which eliminates the traditional stack processing. It also provides for zero-copy outbound, moving data directly from user space to the target virtual server's memory.

SMC-D completes the overall SMC solution, which provides synergy with SMC-R. Both protocols use shared memory architecture, which eliminates TCP/IP processing in the data path, yet preserves TCP/IP quality of service for connection management, load balancing, and security purposes. SMC-R and SMC-D can be used concurrently or independent of each other. The SMC connection protocol can dynamically determine the most optimal option.

A sample SMC configuration that uses SMC-R (by using RoCE) between the two CECs, and SMC-D (by using ISM) between the two LPARs that are on the same CEC is shown in Figure 3-4.

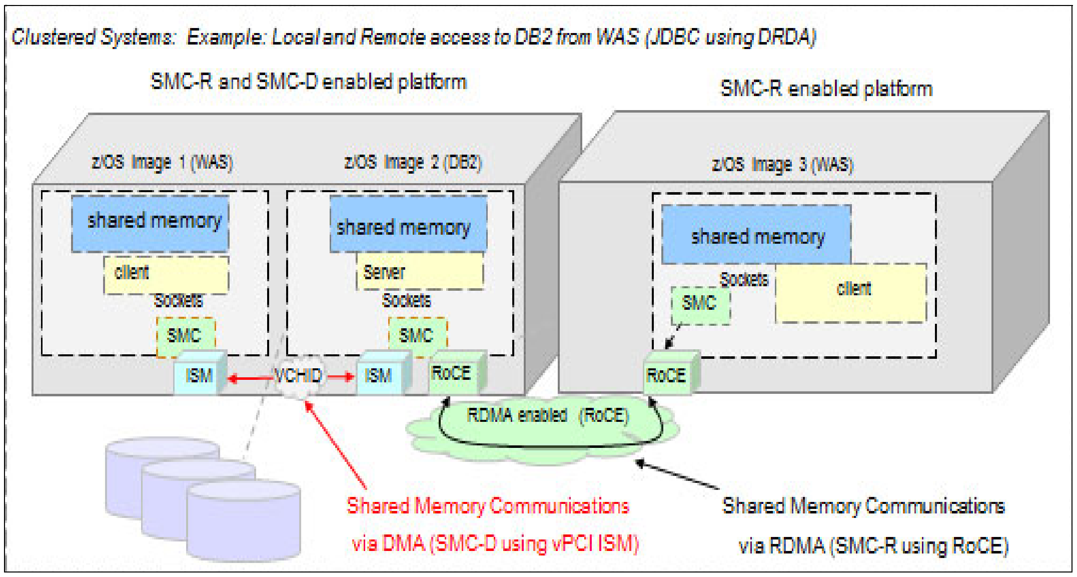

Clustered Systems: Example: Local and Remote access to DB2 from WAS (JDBC using DRDA)
SMC-R and SMC-D enabled platform
SMC-R enabled platform
z/OS Image 1 (WAS)
z/OS Image 2 (DB2)
z/OS Image 3 (WAS)
shared memory
client
Sockets
SMC
shared memory
Server
Sockets
SMC
shared memory
Sockets
client
SMC
ISM
VCHID
ISM
RoCE
RoCE
RDMA enabled (RoCE)
Shared Memory Communications
via DMA (SMC-D using vPCI ISM)
Shared Memory Communications
via RDMA (SMC-R using RoCE)

Figure 3-4 Shared memory communication

Both forms of SMC can be used concurrently to provide a highly-optimized solution. The following SMCs are available by using System z PCI architecture:

- RDMA (SMC-R for cross platforms by using RoCE)
- DMA (SMC-D for same CPC by using ISM)

3.2.2 Request/response workload performance summary comparison of HiperSockets and OSA with and without SMC-D

The following measurements are not Db2 specific but are included to demonstrate the value of the SMC-D feature.

Performance results comparing SMC-D and ISM to HiperSockets

Internal IBM benchmark testing was performed to demonstrate the throughput, latency (response time), and CPU consumption advantages that use the SMC-D protocol across ISM devices has, when compared to using the traditional TCP/IP protocol across HiperSockets.

Both interactive (request/response) workloads with small to medium message sizes were tested with varying numbers of concurrent TCP connections or threads active. In each case, a significant improvement was seen in the throughput and latency (response time) of the transaction.

Substantial improvements were also seen in the amount of CPU that was used per transaction bidirectionally (on the client and server sides). When SMC-D is used, the throughput doubled while latency and CPU usage were cut in half as compared to the use of traditional TCP/IP over HiperSockets. As the number of bytes transferred grew, even greater results were achieved.

The configuration of the environment that was used during these tests was a z13 platform with four CPs (all CPs in the same drawer) running z/OS V2R2.

The IBM internal (micro) benchmarks that were achieved by using standard tools and data points are shown in Figure 3-5.

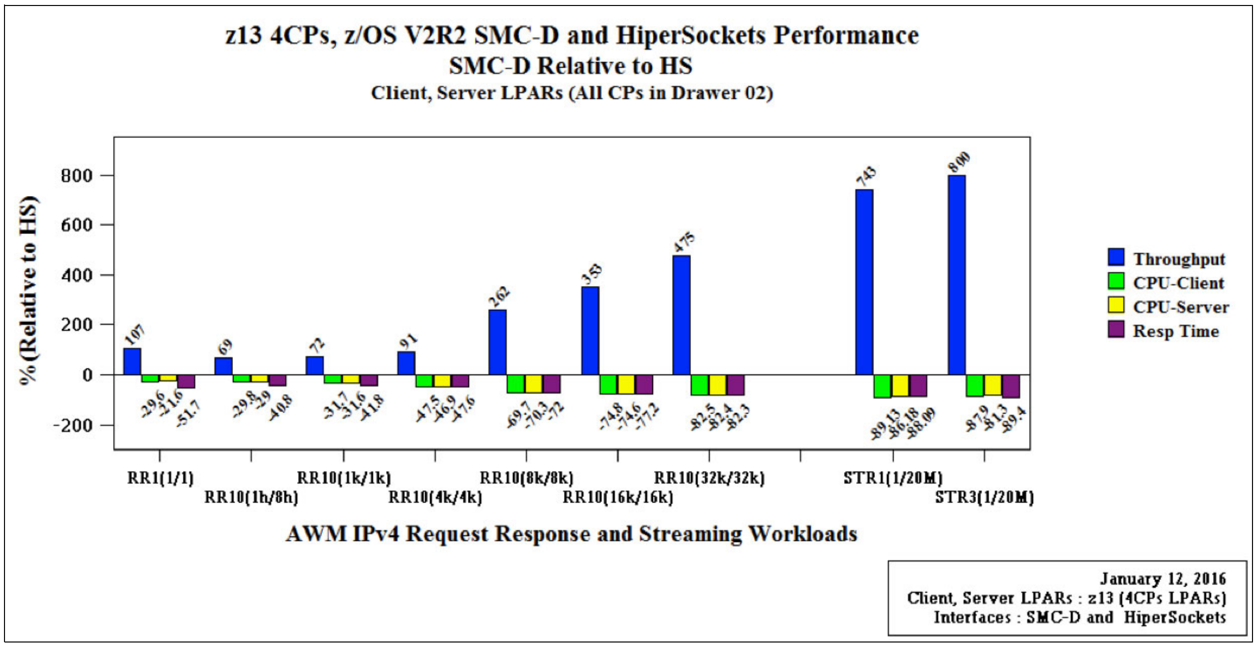


Figure 3-5 SMC-D and ISM versus HiperSockets

Each transactional data point is denoted with RRN(size) where RR indicates a request/response pattern, N indicates the number of concurrent connections, and threads and the payload size is indicated within the parenthesis. When moving left to right as the payload size grows, the benefits increase, which demonstrates the scalability of the technology.

The last two data points that are shown on the right in Figure 3-5 are related to streaming workloads. This type of benchmark testing was also performed on workloads with streaming data patterns (bulk data being transferred). In the streaming tests, up to 88% CPU savings was seen, and up to an almost 9 times increase in network throughput.

For the request/response (transaction) type measurements, the following improvements were observed:

- Request/response improvement for workloads with 1 k/1 k - 4 k/4 k payloads:
  - Latency: Up to 48% reduction in latency
  - Throughput: Up to 91% increase in throughput
  - CPU cost: Up to 47% reduction in network related CPU cost

- Request/Response improvement for workloads with 8 k/8 k - 32 k/32 k payloads:
  - Latency: Up to 82% reduction in latency
  - Throughput: Up to 475% (~6x) increase in throughput
  - CPU cost: Up to 82% reduction in network related CPU cost

SMC-D and ISM to OSA performance summary

The same set of measurements testing was conducted. However, this time by using an OSA Express 4 10Gbit interface as a basis for comparison against SMC-D. The results are shown in Figure 3-6.

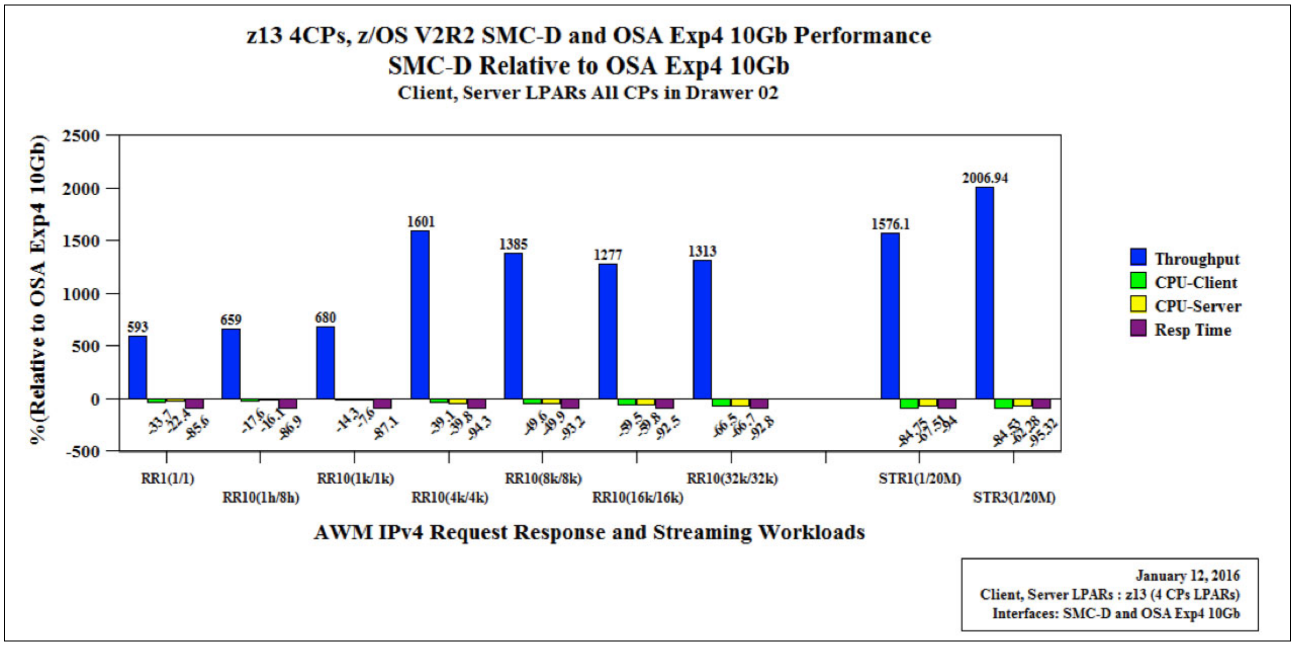


Figure 3-6 SMC-D and ISM versus OSA Express 4 10 Gb

For the request/response type work, the following observations were made:

- Request/response improvement for workloads with 1 k/1 k - 4 k/4 k payloads:
  - Latency: Up to 94% reduction in latency
  - Throughput: Up to 1601% (~17x) increase in throughput
  - CPU cost: Up to 40% reduction in network related CPU cost
- Request/response improvement for workloads with 8 k/8 k - 32 k/32 k payloads:
  - Latency: Up to 93% reduction in latency
  - Throughput: Up to 1313% (~14x) increase in throughput
  - CPU cost: Up to 67% reduction in network related CPU cost

For streaming type workload, a latency reduction of up to 95% was observed. Up to 21 times improvement in throughput and up to 85% reduction in network-related CPU cost also was observed.

3.2.3 SMC-D results for IRWW SQLJ OLTP workload

To demonstrate the value of the use of SMC-D in a Db2 environment where the requester and server are co-located on the same CEC, several measurements were taken by using the IRWW SQLJ OLTP workload. This workload includes seven different transactions in support of a warehouse ordering and delivery system. It is the same workload, in terms of the work that is run, as the standard IRWW workload. The major difference is that it is using static Java (SQLJ).

On average, each transaction consists of ~15 SQL statements (select/insert/update/delete) and their associated network messages. The workload is run under USS in one z/OS LPAR, and connects to a Db2 subsystem on another z/OS LPAR in the same CPC. The measurements use a Db2 11 for z/OS environment as Db2 12 was still under development at the time these measurements were taken. Similar results in terms of network latency improvements are expected with Db2 12 for z/OS.

The following performance measurements compare HiperSockets with SMC-D, to HiperSockets without SMC-D (that is, "old style" HiperSockets), and OSA 10 Gb without SMC-D enabled. The results are shown in Figure 3-7.

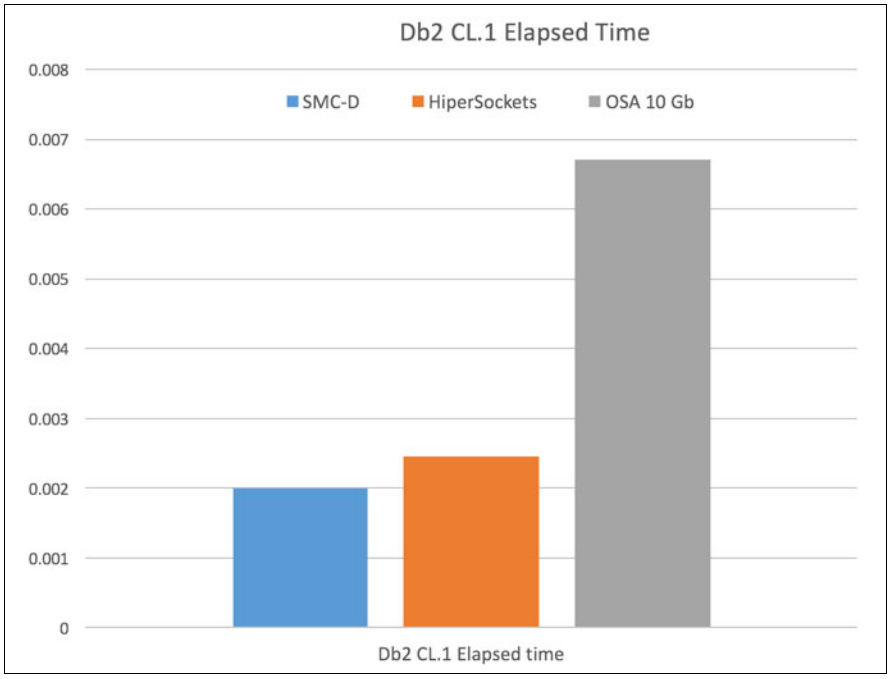


Figure 3-7 Db2 Class 1 elapsed time

From a Db2 class 1 elapsed time perspective, a 19.17% decrease in elapsed time was observed when SMC-D was used (compared to HiperSockets), and a 70.36% decrease in elapsed time when compared to the use of OSA 10 Gb.

The throughput rate in transactions per second for these three measurements is shown in Figure 3-8.

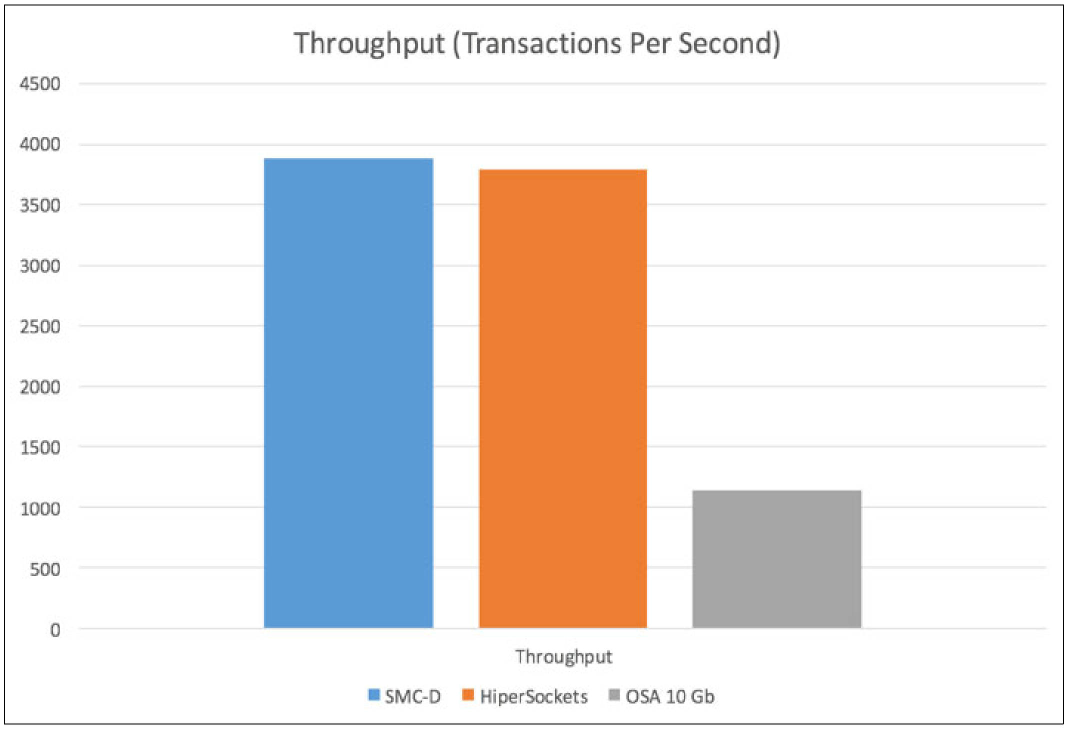


Figure 3-8 Throughput in transactions per second

The throughput rate (in transactions per second) increased by 24.77% when SMC-D was used compared to HiperSockets. A 239.75% increase in throughput rate was observed when compared against OSA 10 Gb.

The workload measurement results from a normalized throughput perspective are shown in Figure 3-9. Normalized throughput is the throughput rate (in transactions per second) that are normalized out to 100% CPU usage, which is also known as the Internal Throughput Rate (ITR).

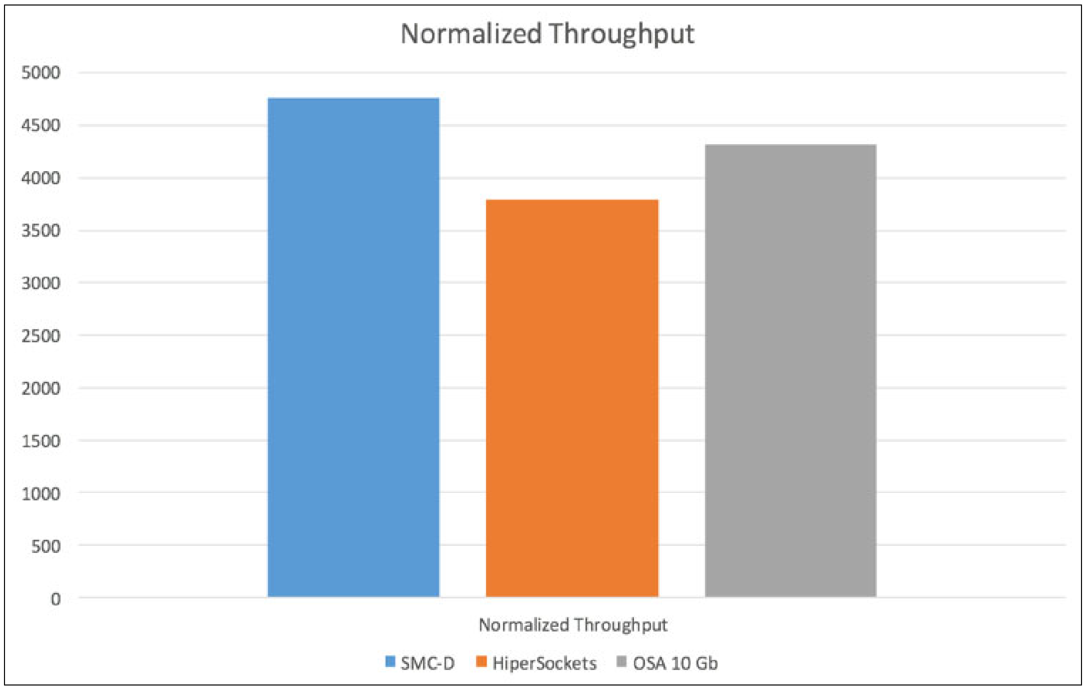


Figure 3-9 Normalized throughput

The normalized throughput rate shows a 25.64% increase when SMC-D is used compared to HiperSockets, and a 10.23% increase compared to OSA 10 Gb. This results provides some insight into CPU time savings on per-transaction basis.

Customers that use an BM z13 GA2 or z13s and z/OS Version 2.2 that have Db2 workloads (such as SQLJ IRWW) that require communication between LPARs on the same CEC, can greatly benefit from the use of SMC-D. The combined hardware and software solution provides improved transaction rates, significantly reduced network latency, and lower CPU cost.

3.3 Simultaneous multithreading

With the introduction of the z13 machine, IBM implemented simultaneous multithreading (SMT-2), which enables two active instructions per processor core. This feature can improve the processing capacity per processor core when more than one task is running. SMT-2 is available for workloads that are running on the Integrated Facility for Linux (IFL) and workloads that use the z Integrated Information Processor (zIIP). SMT-2 also requires operating system support by z/OS (for zIIP usage) and IBM z/VM® (for IFL usage).

For more information about SMT-2, see the following resources:

- *z Systems Simultaneous Multithreading Revolution*, REDP-5144
- *IBM z13 Technical Guide*, SG24-8251

Db2 workloads that are eligible for offloading work to a zIIP processor are suitable candidates for the use of this new z13 processor feature. By using SMT-2, you can expect that fewer zIIP processors are needed to process the same amount of Db2 offloaded zIIP eligible work, which can help to reduce the total cost of computing.

The feature is transparent to Db2 releases and works with both Db2 11 and 12.

Complete the following steps enable SMT-2:

1. Modify the LOADxx system configuration member to indicate the processor view of core for the duration of the IPL, as shown in the following example:

   ```
   PROCVIEW CORE
   ```

2. Specify the following parameter in IEAOPTxx member to indicate the number of active threads per zIIP core:

   ```
   MT_ZIIP_MODE=2
   ```

   Also use the default `IIPHONORPRIORITY=YES` setting in IEAOPTxx.

3. IPL the LPAR. After the IPL, verify the configuration by running the **`D M=CORE`** system command.

The output of the command that is shown in Example 3-3 shows SMT-2 (MT=2) is enabled and that the three online zIIP engines can run two threads per zIIP core (zIIP=2).

Example 3-3 D M=CORE command output

```
D M=CORE
IEE174I 11.04.09 DISPLAY M 325
CORE STATUS: HD=Y   MT=2  MT_MODE: CP=1  zIIP=2
ID    ST   ID RANGE   VP  ISCM  CPU THREAD STATUS
0000   +   0000-0001  H   0000  +N
0001   +   0002-0003  H   0000  +N
0002   +   0004-0005  H   0000  +N
0003   +   0006-0007  H   0000  +N
0004   +   0008-0009  H   FC00  +N
0005   +I  000A-000B  H   0200  ++
0006   +I  000C-000D  H   0200  ++
0007   +I  000E-000F  H   0200  ++

CPC ND = 002964.NE1.IBM.02.00000008E8B7
CPC SI = 2964.7B1.IBM.02.000000000008E8B7
         Model: NE1
CPC ID = 00
CPC NAME = SYSZD1
LP NAME = STLAB11    LP ID =  6
CSS ID  = 0
MIF ID  = 2

+ ONLINE    - OFFLINE    N NOT AVAILABLE    / MIXED STATE
W WLM-MANAGED

I        INTEGRATED INFORMATION PROCESSOR (zIIP)
CPC ND  CENTRAL PROCESSING COMPLEX NODE DESCRIPTOR
CPC SI  SYSTEM INFORMATION FROM STSI INSTRUCTION
CPC ID  CENTRAL PROCESSING COMPLEX IDENTIFIER
CPC NAME CENTRAL PROCESSING COMPLEX NAME
LP NAME  LOGICAL PARTITION NAME
```

```
   LP ID    LOGICAL PARTITION IDENTIFIER
   CSS ID   CHANNEL SUBSYSTEM IDENTIFIER
MIF ID   MULTIPLE IMAGE FACILITY IMAGE IDENTIFIER
```

3.3.1 SMT-2 Performance in a one-way data sharing environment

To evaluate this feature, a complex OLTP IBM Brokerage workload was run by using a machine with eight processor cores. The workload was run on zIIP processors with multithreading (SMT-2) and without multithreading (SMT-1).

Different experiments were conducted by using different combinations of general CPs and zIIP cores. The increased processing capacity that used SMT-2 versus SMT-1 was evaluated by measuring the throughput for different combinations of general CPs and zIIP engines.

The bars that are shown in Figure 3-10 show the internal throughput rate (internal throughput rate is the throughput rate normalized out to 100% CPU utilization) for different processor configuration with SMT-1 zIIPs and SMT-2 zIIPs. The internal throughput is always higher when SMT-2 zIIP processors are used.

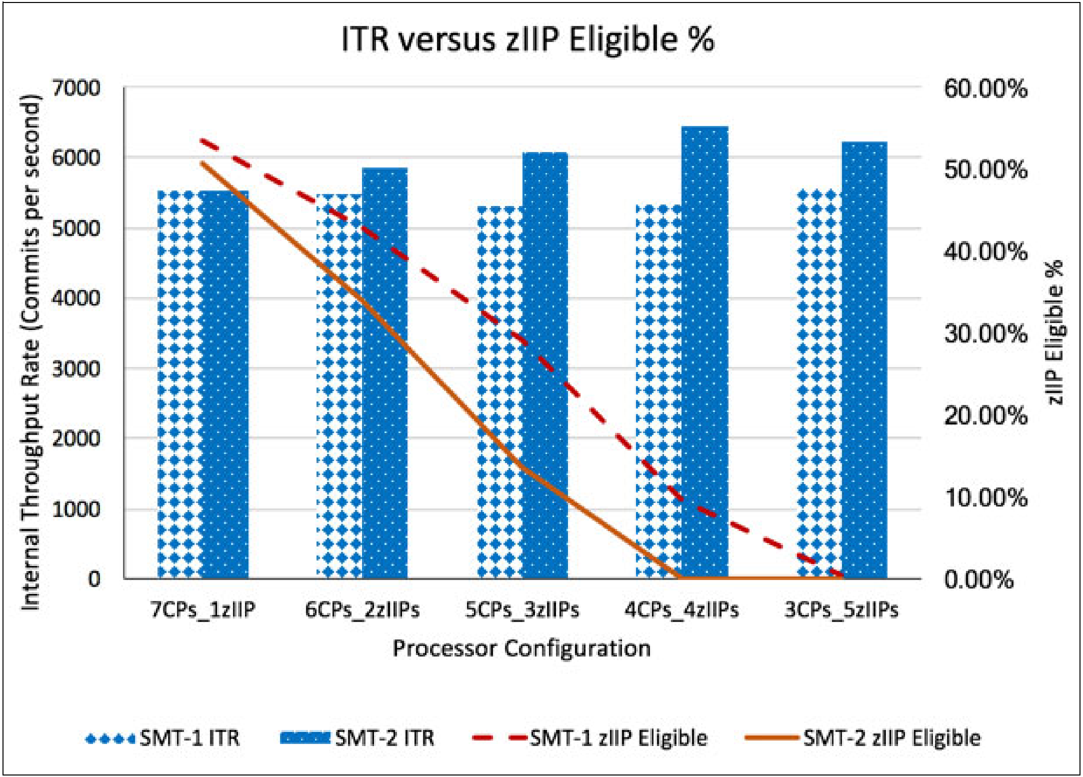


Figure 3-10 ITR versus eligible percentage comparison

The perforated line on the chart in Figure 3-10 indicates how much of the general CP processing is eligible for offload to zIIP (but ran on a CP) with SMT-1. The solid line on the chart shows how much of the CP processing is eligible for offload to zIIP by using SMT-2 (but ran on a CP). The data shows that all eligible zIIP processing is satisfied with four zIIPs when SMT-2 is used, whereas it takes five zIIP processors to offload all eligible zIIP work to run on zIIP engines for SMT-1.

Figure 3-11 shows the measured CPU times, general CP (bar in bold) and zIIP (bar that uses the pattern) for different combinations of processors for SMT-1 and SMT-2 scenarios that use the same workload.

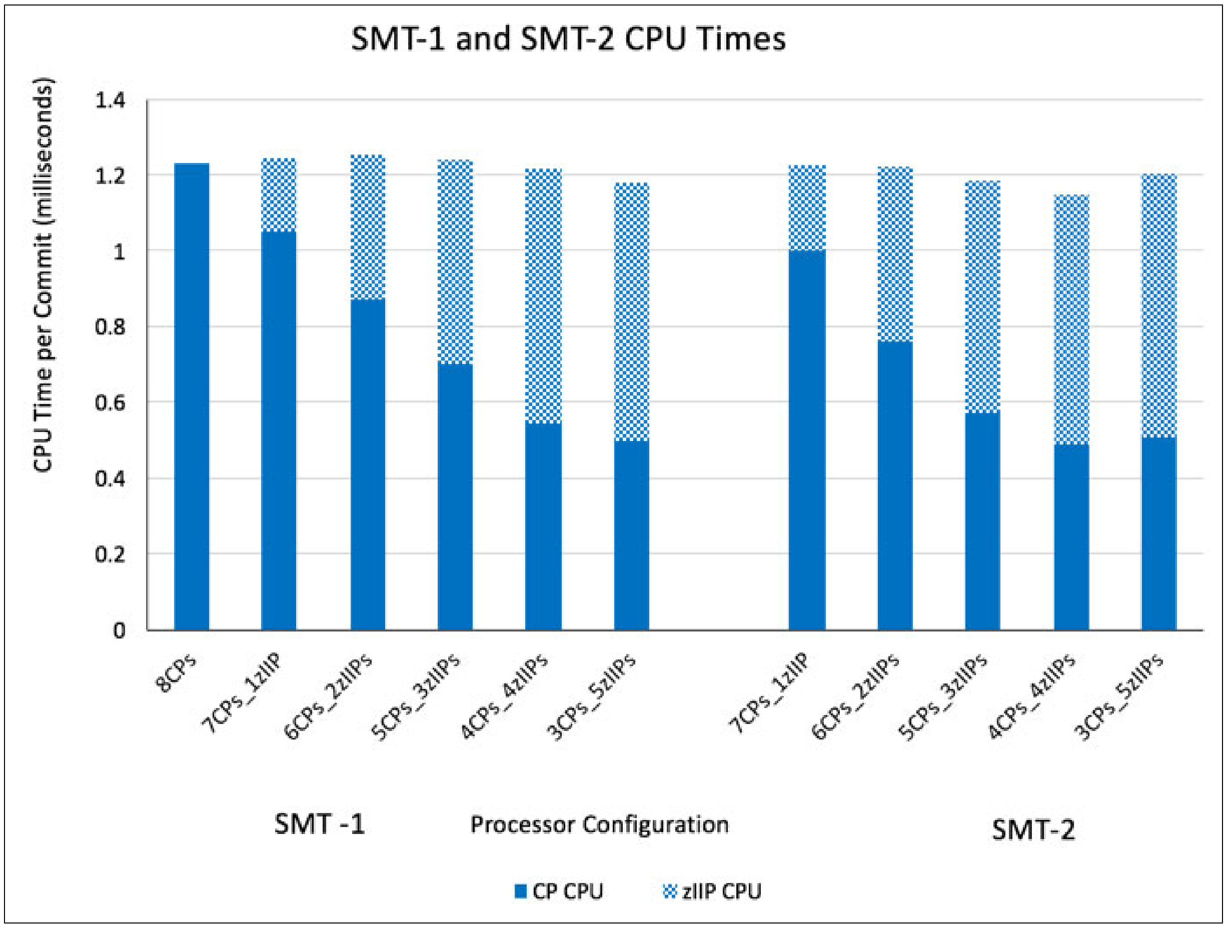


Figure 3-11 SMT-1 and SMT-2 CPU times comparison

For configurations that used 1 - 4 zIIPs, the CP CPU time (which is chargeable) is less when SMT-2 is enabled for the zIIP engines. After all zIIP eligible processing is offloaded for SMT-1, no difference in CP CPU time between SMT-1 and SMT-2 was observed, as expected.

The effect of SMT-1 and SMT-2 on the Db2 elapsed time is shown in Figure 3-12. The solid bar in the chart shows the average class 2 elapsed time when all eight general purpose processors are used. The other two types of bars show the average Db2 elapsed time with different combinations of general CPs and zIIP processors with SMT-1 and SMT-2.

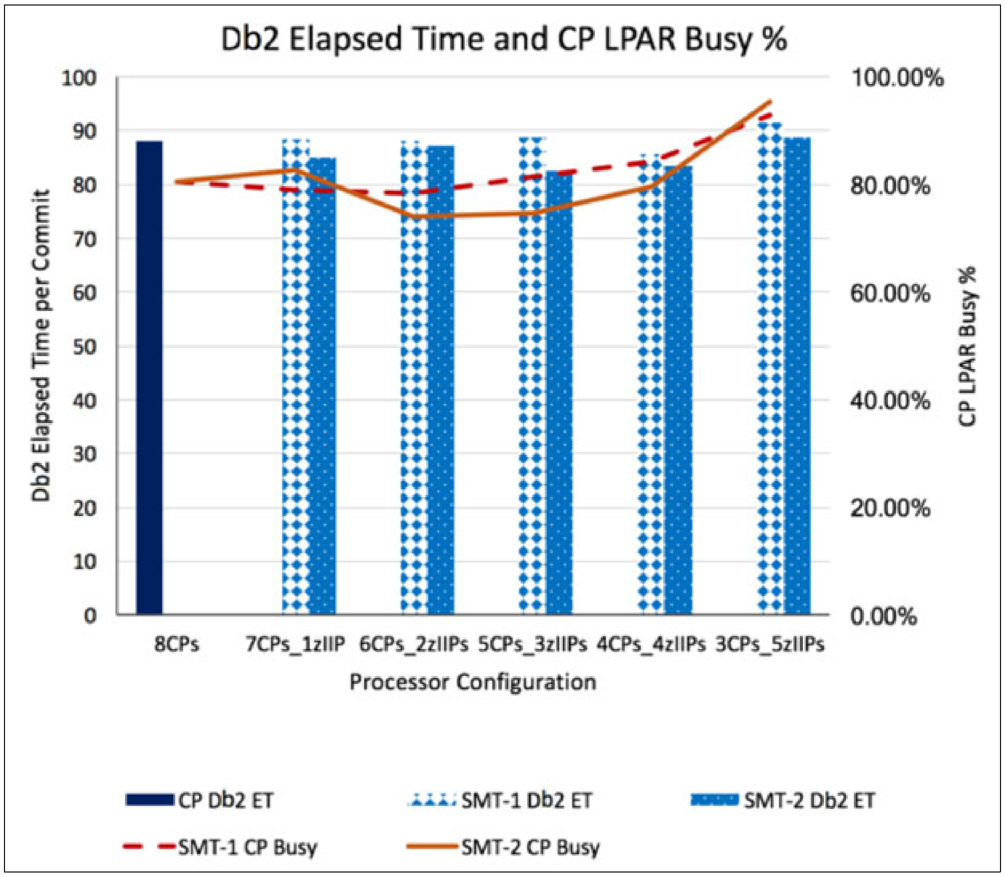


Figure 3-12 Db2 elapsed time and CP LPAR busy percentage comparison

The perforated line indicates LPAR Busy % for general purpose processors (CP) with SMT-1, and the solid line shows LPAR Busy % for general purpose processors (CP) with SMT-2. We did not observe any increase in elapsed time when SMT-1 or SMT-2 was used until we reduced the number of general purpose processors to the point where they were over 80% used utilized and no other work was eligible for zIIP offload.

3.3.2 SMT-2 performance in a two-way data sharing environment

The IBM Brokerage OLTP workload was also run in a two-way data sharing environment to observe the performance affect of the use of SMT-2. The workload ran in a parallel sysplex that consisted of two LPARs and one internal CF. Each LPAR has three CPs and three zIIPs. The internal CF had three dedicated CPs and eight ICP links that were shared between LPARs and the CF.

The workload was run by using zIIP processors with multithreading (SMT-2) enabled, and without multithreading (SMT-1). As with the one-way data sharing tests that are described in 3.3.1, “SMT-2 Performance in a one-way data sharing environment ” on page 102, the increased processing capacity comparing SMT-2 to SMT-1 was measured by the workload internal throughput rate (ITR). The effect on the total cost of computing for the workload was measured by the increased amount of zIIP CPU time or reduced CPU time on general CPs. In addition, the workload overhead when a two-way data sharing overhead was used was analyzed between SMT-1 and SMT-2.

The increased processing capacity when SMT-2 was used is shown in Figure 3-13. For this workload, as much as 15% more processing capacity was observed with SMT-2 enabled.

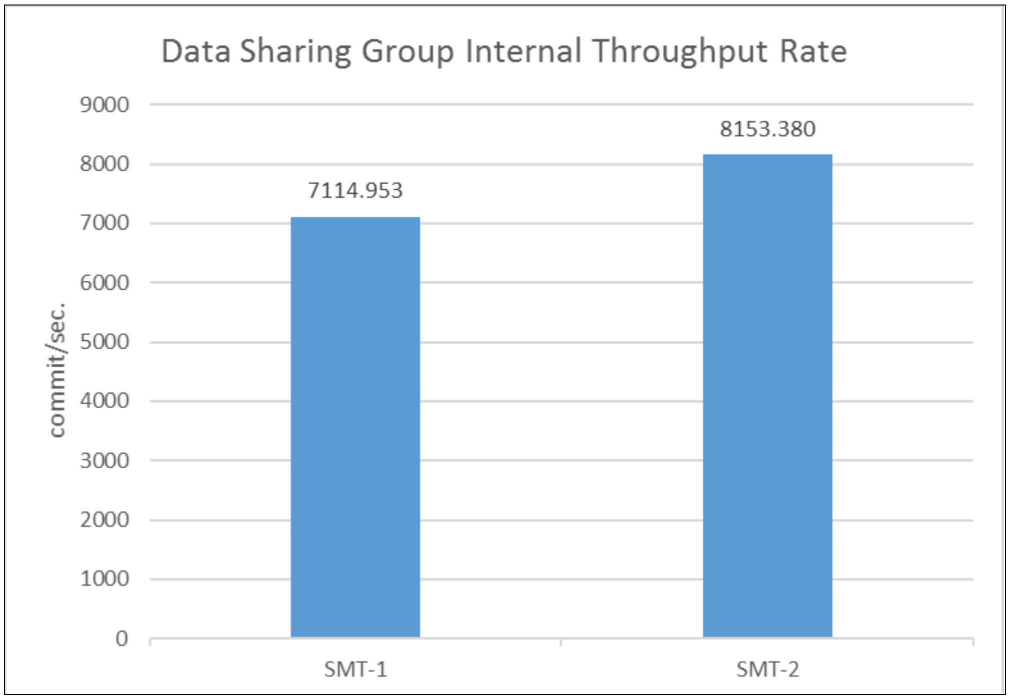


Figure 3-13 Data sharing group internal throughput rate comparison

The distribution of the transaction’s total Db2 CPU time between general CP and zIIP engines is shown in Figure 3-14 on page 106. Transaction total Db2 CPU time includes Db2 class 2 CPU time, and MSTR, DBM1 and IRLM CPU time.

As shown in Figure 3-14 on page 106, 55.8% of the total Db2 CPU time was on zIIP (free of charge) when SMT-1 was used. With SMT-2, 56.9% of the total CPU time was on zIIP; a full percentage point more than the SMT-1 case. Therefore, the total cost of computing was reduced when SMT-2 is enabled.

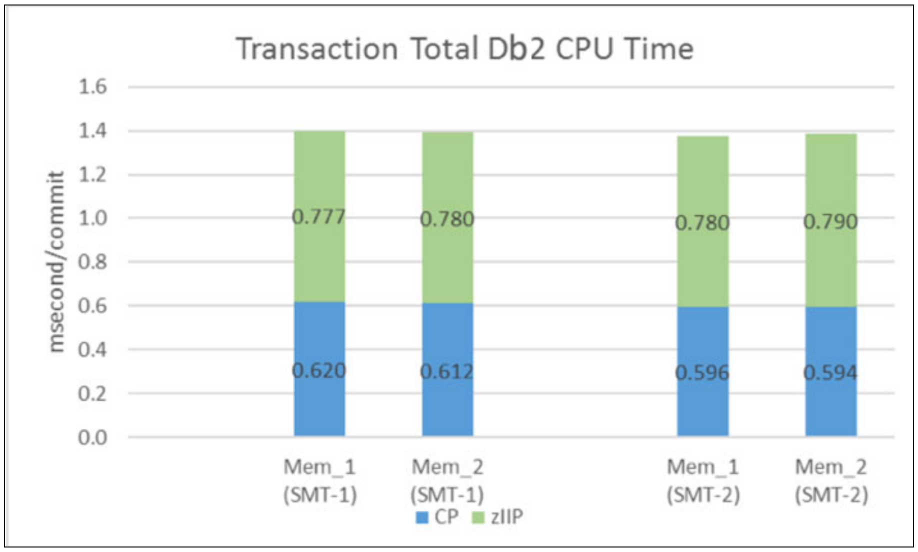


Figure 3-14 Transaction total Db2 CPU time distribution comparison

An interesting observation from this exercise is that the data sharing overhead in this two-way configuration dropped a full percentage point with SMT-2 enabled. The two-way data sharing overhead is calculated as the ratio of the data sharing group ITR and twice the ITR of the one-way system. This cost is the "start-up" cost when moving to data sharing and actively sharing objects in read/write or write/write mode between the two Db2 members. Adding more Db2 data sharing members costs significantly less and a Db2 data-sharing group scales well. The lower the data sharing overhead is, the better.

The transaction Db2 elapsed time per commit for SMT-1 and SMT-2 in a two-way data sharing environment is shown in Figure 3-15.

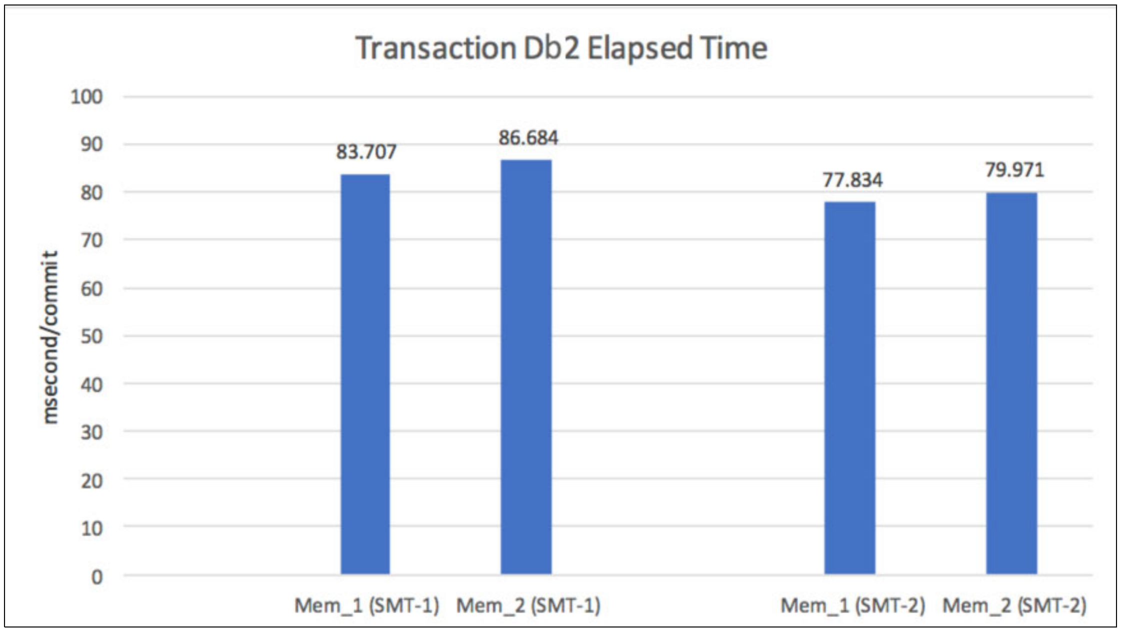


Figure 3-15 Transaction Db2 elapsed time comparison

The measurements show a response time improvement of more than 7% when SMT-2 is enabled.

The IBM Brokerage OLTP workload that is running in a two-way data sharing environment performs better with SMT-2. We observed increased workload processing capacity, lower cost of computing, and less data sharing overhead.

3.4 Performance effect of the use of 2 GB frames

With the support of 64-bit addressing, more virtual (and real) storage is available to Db2 for z/OS. In addition to allowing Db2 to use more storage for internal storage areas, 64-bit addressing enables customers to use much larger local buffer pools than in the past.

Every access to a memory address must go through a virtual-to-real address mapping by using information that is in page table entries (PTEs). If the mapped address is cached on the translation look-aside buffer (TLB), which is a hardware cache on the processor, the mapping process is efficient. If there is a TLB cache miss, processing must stop and wait for a main memory search to locate the entry and place it on the TLB.

The system TLB sizes have remained relatively the same because of the requirement of quick access and limitations on hardware space. When the storage size increases with a similar TLB cache, the hit ratio of TLB is likely to drop, which means a larger number of TLB misses. When this issue occurs, Db2 can suffer a significant performance penalty.

The use of large page frames (1 MB or 2 GB) instead of traditional 4 KB frames, allows a single TLB entry to fulfill many more address translations. Therefore, better coverage is achieved without expanding TLB size, the TLB hit ratio is improved, and Db2 can benefit from the improved TLB hit ratio.

The use of 2 GB frames requires a zEC12 or above processor. For a buffer pool to be able to use a FRAMESIZE of of 2 GB in Db2, it must also be defined with the PGFIX(YES) attribute. This definition makes it non-pageable.

Performance measurements at the IBM Silicon Valley Laboratory that used a zEC12 processor showed equivalent CPU usage for the internal workload when 2 GB frames were used compared to 1 MB frames. Therefore, no significant performance benefits were observed for the workload that was being tested for 2 GB frames. With the availability of the z13, the same measurements were conducted by using the latest hardware. Improvements were observed with these measurements with 2 GB frames compared to 1 MB frames.

3.4.1 Performance measurements

By using the z13 hardware, a 4% reduction in Db2 total CPU time (transaction CPU time plus CPU time used by the Db2 system address spaces) was observed when 2 GB frames were used compared to 1 MB frames. The workload ITR was enhanced by 1.7%. An analysis of z13 hardware sampling data revealed an 8.5% reduction in TLB misses when using 2 GB frames.

The measurement results of our tests are listed in Table 3-1. The allocated LFAREA that was used during both measurements was the same. There were 68,009 1 MB frames allocated in the run that used 1 MB frames (EXP_1M), while 33 2 GB and 425 1 MB frames were used during the run that used 2 GB frames (EXP_2G). The results showed 33 * 2048*1 MB + 425*1 MB frames = 68,009 1 MB frames allocated; therefore, an equal number of storage was used as LFAREA in both cases.

Table 3-1 Z13 1MB versus 2 GB frame measurement results

Measurement ID	**1 MB Frame size (EXP_1M)**	**2 GB Frame size (EXP_2G)**	**Delta (EXP_2G/EXP_1M)**
CPU	2964(z13)	2964(z13)	
# General CPs	12	12	
z/OS	z/OS 2.1	z/OS 2.1	
LFAREA allocated	2 GB pages = 0 1 MB pages = 68009	2 GB pages = 33 1 MB pages = 425	
Db2	V11 PTF (07/22/2015)	V11 PTF (07/22/2015)	
Workload			
ETR (commits/sec.)	5055.523	5052.434	
CPU (%)	60.98	59.91	
ITR (commits/sec.)	8290.461	8433.374	+1.7%
CPU time (MBps/commit)			
Class 2 CPU	1.176	1.130	
MSTR	0.005	0.004	
DBM1	0.013	0.012	
IRLM	0.003	0.003	
Total Db2 CPU (member scope)	1.197	1.149	-4.0%

Although this measurement is not Db2 12 specific and was performed by using a Db2 11 environment, you can expect similar results when Db2 12 on a z13 is used with 2 GB page frames.

3.4.2 Usage guidance and considerations

The IEASYSxx LFAREA keyword specifies the amount of real storage to be made available for 1 MB and 2 GB large frames. All 1 MB and 2 GB frames are backed by contiguous 4 KB real storage frames, and are allocated from real storage as specified by the LFAREA keyword, as shown in the following example:

```
LFAREA=(1M=88064,2G=40)
```

As a result of this command, 88,064 1 MB frames and 40 2 GB frames (166 GB real storage) are allocated in the LFAREA at IBM MVS™ IPL time.

The output of the `DISPLAY VIRTSTOR,LFAREA` MVS system command is shown in Example 3-4.

Example 3-4 DISPLAY VIRTSTOR,LFAREA command output

```
D VIRTSTOR,LFAREA
IAR019I  12.13.32 DISPLAY VIRTSTOR 843
 SOURCE =  20
 TOTAL LFAREA = 88064M , 80G
 LFAREA AVAILABLE = 87639M , 14G
 LFAREA ALLOCATED (1M) = 425M
 LFAREA ALLOCATED (4K) = 0M
 MAX LFAREA ALLOCATED (1M) = 456M
 MAX LFAREA ALLOCATED (4K) = 0M
 LFAREA ALLOCATED (PAGEABLE1M) = 0M
 MAX LFAREA ALLOCATED (PAGEABLE1M) = 0M
 LFAREA ALLOCATED NUMBER OF 2G PAGES = 33
 MAX LFAREA ALLOCATED NUMBER OF 2G PAGES = 33
```

Specifying the FRAMESIZE(2G) option for the Db2 buffer pool allows 2 GB frame to be allocated, if available. The IBM Db2 **`-DISPLAY BPOOL`** command output shows the preferred frame size and how many buffers that are using each frame size are allocated, as shown in Example 3-5.

Example 3-5 DISPLAY BUFFERPOOL output

```
-DIS BPOOL(BP1) DETAIL(INTERVAL) LSTATS(ACTIVE)
DSNB401I  -CEB2 BUFFERPOOL NAME BP1, BUFFERPOOL ID 1, USE COUNT 65
DSNB402I  -CEB2 BUFFER POOL SIZE = 524288 BUFFERS  AUTOSIZE = NO
           VPSIZE MINIMUM = 0 VPSIZE MAXIMUM = 0
             ALLOCATED       =   524288   TO BE DELETED   =       0
             IN-USE/UPDATED  =    75018
DSNB431I  -CEB2 SIMULATED BUFFER POOL SIZE = 0 BUFFERS -
             ALLOCATED       =        0
             IN-USE          =        0   HIGH IN-USE     =       0
             SEQ-IN-USE      =        0   HIGH SEQ-IN-USE =       0
DSNB406I  -CEB2 PGFIX ATTRIBUTE -
             CURRENT = YES
             PENDING = YES
           PAGE STEALING METHOD = FIFO
DSNB404I  -CEB2 THRESHOLDS -
            VP SEQUENTIAL    =  0   SP SEQUENTIAL   =  0
            DEFERRED WRITE   = 30   VERTICAL DEFERRED WRT  =  5,  0
            PARALLEL SEQUENTIAL =50   ASSISTING PARALLEL SEQT=  0
DSNB546I  -CEB2 PREFERRED FRAME SIZE 2G
        524288 BUFFERS USING 2G FRAME SIZE ALLOCATED
```

If the VPSIZE of a 4 KB page size buffer pool is 600,000, Db2 allocates one 2 GB frame and many 1 MB frames. When the buffer pool size is close to multiples of 2 GB, Db2 rounds up the allocation and allocates another 2 GB frame to avoid wasting storage.

3.5 Buffer pool simulation

Large buffer pools that are backed by real storage can help eliminate I/O and reduce CPU usage of Db2. However, the benefit from larger buffer pools depends on the access frequency of the same data and the workload's data access pattern.

If the data in the buffer pool is never re-referenced, there is no benefit in increasing the buffer pool size. If the data access pattern is skewed, there might not be much benefit from making the buffer pool larger.

It is often hard to determine the optimum size of a buffer pool. The tools that are available to predict the optimum size of the buffer pool usually require extensive tracing in production environments and are not always easy to use. To address the problem and accurately predict the buffer pool size with limited intervention in a production environment, buffer pool simulation was introduced in Db2 12. This feature is also available in Db2 11 with APAR PI22091.

Buffer pool simulation aims to help customers determine the correct size of buffer pools. Buffer pool simulation can be used to determine how many synchronous I/Os can be avoided if the simulated buffer size is added to the current buffer pool size. The simulated buffer pool, unlike the real buffer pool, uses much less storage and CPU compared to an actual buffer pool of that size, because the data is not stored in the simulated buffer pool. Only approximately 70 bytes of storage are required for each simulated buffer. For example, a simulated buffer pool of 4 MB require only approximately 70 KB of extra storage.

During our tests that used IBM internal workloads, only a 1% CPU overhead per simulated buffer pool was observed. Given the small amount of extra storage that is required and the small CPU overhead that is incurred with buffer pool simulation, customers can run buffer pool simulation in a production environment and more accurately estimate the right size for a buffer pool.

3.5.1 Performing buffer pool simulation

Complete the following steps to simulate a buffer pool:

1. Issue the **`-ALTER BUFFERPOOL`** command and specify the SPSIZE (simulated pool size) and SPSEQT (sequential prefetch threshold to be used for the simulated pool parameters for the buffer pool to be simulated. (The sum of SPSIZE plus VPSIZE is the buffer pool size that is going to be simulated.)

   Although specifying SPSEQT is optional, it is highly recommended to specify it because the sequential threshold should be adjusted when you have a larger buffer pool size. If SPSEQT is not specified, the current VPSEQT value for the simulated buffers is used in the simulation, as shown in the following command:

   ```
   -ALTER BUFFERPOOL(BP1) .. SPSIZE (2000) SPSEQT(20)
   ```

   This command simulates an increase in the size of BP1 by 2000 pages by using a sequential threshold of 20% for these extra pages in the simulated pool.

2. Start the collection of Db2 statistics data or issue the **`-DISPLAY BUFFERPOOL(BP1) DETAIL`** command to reset the buffer pool statistics at the beginning of the simulation.
3. Run the workload for at least two hours.
4. At the end of the measurement period, collect the Db2 statistics data or issue another **`-DISPLAY BUFFERPOOL(BP1) DETAIL`** command to obtain the buffer pool simulation results.

5. Issue the **-ALTER BUFFERPOOL (BP1)…. SPSIZE (0)** command to disable buffer pool simulation.

If you issue another **-ALTER BUFFERRPOOL ... SPSIZE(yy)** command, where yy is smaller than the size that is being simulated, the counters are reset to zero and restarts.

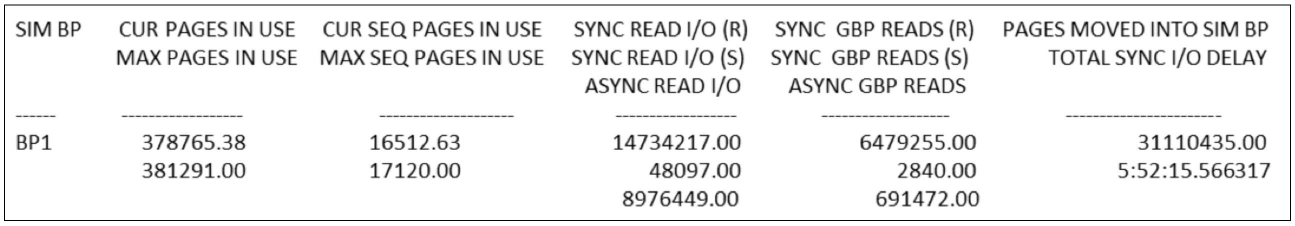
The part of the output of the **-DISPLAY BUFFERPOOL DETAIL** command that can help in determining the right size of the buffer pool is shown in Example 3-6.

Example 3-6 -DIS BP DETAIL command output

```
DSNB411I  -CEA1 RANDOM GETPAGE    = 138590374
                SYNC READ I/O (R) =  25466501
                SEQ.   GETPAGE    =   4056005
                SYNC READ I/O (S) =     81441
                DMTH HIT          =         0
                PAGE-INS REQUIRED =         0
                SEQUENTIAL        =       491
                VPSEQT HIT        =         0
                RECLASSIFY        =   3909397
DSNB431I  -CEA1 SIMULATED BUFFER POOL SIZE = 494000
                BUFFERS -
                 ALLOCATED       =   494000
                 IN-USE          =   371382
                 HIGH IN-USE     =   371436
                 SEQ-IN-USE      =     5255
                 HIGH SEQ-IN-USE =     6020
DSNB432I -CEA1 SIMULATED BUFFER POOL ACTIVITY -
               AVOIDABLE READ I/O -
                SYNC READ I/O (R)  =  25463982
                SYNC READ I/O (S)  =     81181
                ASYNC READ I/O     =  15470503
                SYNC GBP READS (R) =  11172099
                SYNC GBP READS (S) =      4601
                ASYNC GBP READS    =   1181076
PAGES MOVED INTO SIMULATED BUFFER POOL = 53668641
TOTAL AVOIDABLE SYNC I/O DELAY = 35321543 MILLISECONDS
```

Buffer pool simulation information is also available in the IFCID 2 statistics record in a new data section that is called DSNDQBSP.

IBM Tivoli® OMEGAMON® XE for Db2 Performance Expert on z/OS provides this buffer pool simulation information in the Db2 statistics report. An example is shown in Figure 3-16.



SIM BP	CUR PAGES IN USE MAX PAGES IN USE	CUR SEQ PAGES IN USE MAX SEQ PAGES IN USE	SYNC READ I/O (R) SYNC READ I/O (S) ASYNC READ I/O	SYNC GBP READS (R) SYNC GBP READS (S) ASYNC GBP READS	PAGES MOVED INTO SIM BP TOTAL SYNC I/O DELAY
------	-------------------	---------------------	-------------------	--------------------	------------------------
BP1	378765.38	16512.63	14734217.00	6479255.00	31110435.00
	381291.00	17120.00	48097.00	2840.00	5:52:15.566317
			8976449.00	691472.00	

Figure 3-16 OMPE Statistics report - BP simulation output

The data that is shown in Example 3-6 on page 111 and Figure 3-16 on page 111 was taken during the same measurement run. However, the statistics information was captured first. The output from the **-DIS BPOOL** command was captured later during the run, while SPSIZE was still greater than zero (simulation still active), which explains why the numbers look different.

You can use the output of the **-DISPLAY BUFFERPOOL DETAIL** command after the simulation, or the buffer pool simulation information in IFCID 2 to determine the size of the expanded buffer pool. To calculate what percentage of I/O is avoided by the simulated buffer pool, review the number of getpages that result in I/O and how many of these pages can be avoided because they were found in the (extra) simulated pool.

For example, message DSNB411I that is shown in Example 3-6 on page 111 reports the total number of random getpages that resulted in an I/O (25466501). Message DSNB432I reports the number of avoidable I/Os from random getpages (25463982). By using this info, we can calculate the percentage of random synchronous I/Os that can be avoided by using the simulated buffer pool. In this example, almost all the I/Os, approximately 99.9% (25463982/ 25466501) can be avoided by changing the buffer pool size to total simulated buffer pool size.

Message DSNB432I also reports the GBP reads that can be avoided because the pages are found in the simulated buffer pool. It is important to note that buffer pool simulation assumes that the group buffer pool is large enough to accommodate the increased size of the local simulated buffer pool.

Message DSNB431I indicates how much of the simulated buffer pool is used during the measurement period to achieve the I/O savings. The HIGH-IN-USE buffers counter is incremented as buffers are used in simulated buffer pool. Therefore, add the HIGH IN-USE buffers of the simulated buffer pool to the real (current) buffer pool size to determine the size of the (new) expanded buffer pool.

If the allocated size of the simulated pool is considerably more than the high in-use value (assuming that the simulation was active long enough to be representative), we can infer that the simulated buffer pool is larger than needed to avoid I/O. If the HIGH-IN-USE is the same as the size of the allocated simulated buffer pool and the avoidable I/O from the current simulated size is limited, run a buffer pool simulation with a larger SPSIZE.

We can also calculate the expected average elapsed time savings from avoidable I/O by using the larger buffer pool size.

The AVOIDABLE READ I/O fields in the DSNB432I message are the number of I/Os that can be avoided because the pages were found in the simulated buffer pool.

The TOTAL AVOIDABLE SYNC I/O DELAY is the delay that is caused by I/O because they are not found in the real buffer pool, but were found in the simulated pool. The TOTAL AVOIDABLE SYNC I/O DELAY divided by the total of all AVOIDABLE READ I/O fields gives you the average delay for an I/O.

For example, by using the information from the DSNB432I message that is shown in Example 3-6 on page 111, we can calculate that the average delay of an avoidable I/O is 862 microseconds (35321543/41015666[1]) when the pages are found in the simulated buffer pool. This saving is the average saving in elapsed time from an I/O if the current buffer pool size (VPSIZE) is increased by the simulated pool size (SPSIZE).

A reduction in the number of synchronous I/Os can translate into reduced elapsed and CPU time if the current buffer pool size (VPSIZE) is expanded.

[1] The 53373442 is the sum of SYNC READ I/O (R) + SYNC READ I/O (S) + ASYNC READ I/) in message DSNB432I.

3.5.2 Performance measurement results

To validate the accuracy of the buffer pool simulation statistics, the IBM Brokerage workload was used. First with an original total buffer pool size of 14 GB and simulating one of the buffer pools by specifying 4 GB for SPSIZE. This simulation showed a large reduction in synchronous I/O. Then, to verify the results, the size of the simulated buffer pool was expanded by 4 GB and the benchmark workload was run again. This test resulted in a substantial reduction in synchronous I/O and CPU time, as predicted by the buffer pool simulation.

The number of synchronous I/Os per second with buffer pool BP1 for the original size of the buffer pool, the synchronous I/Os avoided as predicted by the simulated buffer pool, and the actual number of synchronous I/Os measured after increasing the buffer pool by the simulated buffer pool size is shown in Figure 3-17.

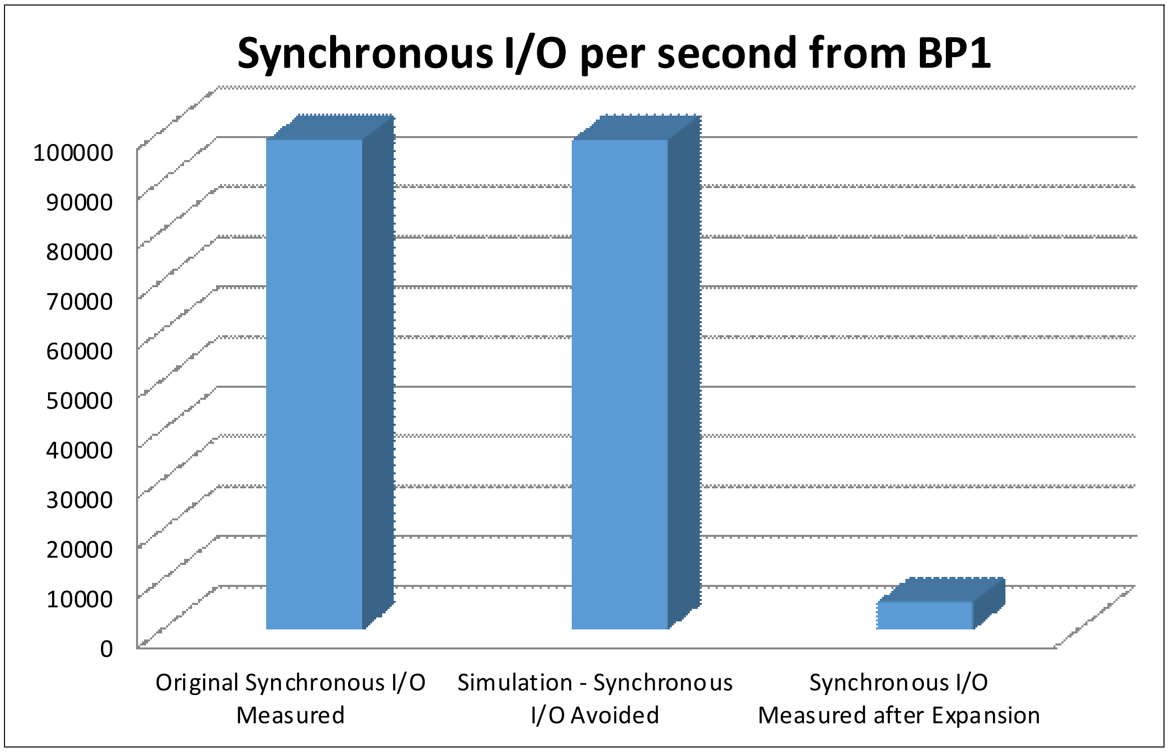


Figure 3-17 Buffer pool simulation - Sync I/O per second

The actual performance improvement that was achieved by expanding the buffer pool size based on the results of the buffer pool simulation is shown in Figure 3-18. The bar in the chart is the number of synchronous I/Os per commit and the line is the total CPU time per commit measured for the brokerage workload.

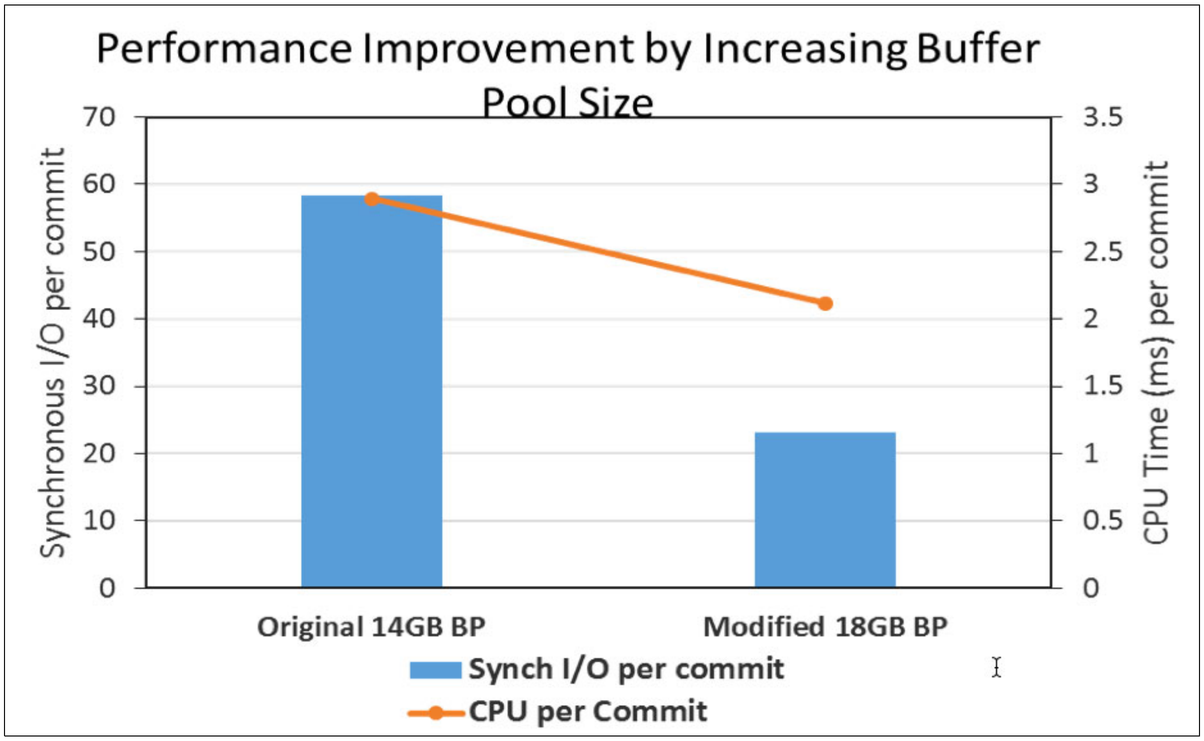


Figure 3-18 Performance improvement by increasing buffer pool size

3.5.3 Buffer pool simulation usage considerations

Consider the following points when buffer pool simulation is used:

- Buffer pool simulation works only for buffer pools that use the PGSTEAL (LRU) attribute.
- Simulation is performed for read I/O operations only.
- SPSIZE must be greater than zero to enable simulation, which implies that simulating a smaller buffer pool than the original buffer pool is not possible.
- Specifying SPSIZE (0) disables simulation and deletes the simulated buffer pool.
- The sum of the VPSIZE and SPSIZE of all buffer pools combined cannot exceed the Db2 maximum total buffer pools size (1 TB in Db2 11 and 16 TB in Db2 12).
- In a data sharing environment, buffer pool simulation assumes that the group buffer pool can handle the increased local buffer pool size. Therefore, you might want to verify whether the group buffer pool size also must be adjusted if the local buffer pool is expanded.
- If SPSEQT is not specified, the simulation assume VPSEQT as the sequential threshold for the simulated pool. You might want to consider decreasing the SPSEQT threshold if a large SPSIZE is used based on sequential I/O buffer pool statistics.

- PAGES MOVED INTO SIMULATED BUFFER POOL indicates the number of pages that must be stolen in the real buffer pool (to make room for pages that are staged in from DASD). These stolen pages from the real buffer pool are tracked in the simulated buffer pool until they are stolen from the simulated buffer pool.

3.6 Large group buffer pool usage

The closer the data is to the processor the better system performance is going to be, according to a general consensus. This belief also applies to Db2 for z/OS. For more information about the benefit of the use of large local buffer pools, section 3.7, Exploitation of Large Memory for Buffer Pools, in *IBM DB2 11 for z/OS Performance Topics*, SG24-8222.

In this section, we describe the performance benefits of having large Group Buffer Pools (GBP).

When a getpage request for a GBP-dependent object occurs in a Db2 data sharing environment, Db2 first checks whether the request can be satisfied by using the member's local buffer pool.

If the page is not found in the local buffer pool or the page is cross invalidated, Db2 attempts to satisfy the request by looking in the associated group buffer pool. Only when the page is not present in the GBP is a synchronous read required to bring the page into the local buffer pool from disk so it can be used by the transaction that requested it.

However, a synchronous read I/O to retrieve a page from disk is a costly operation. Moreover, the time to access a page in the GBP is in the order of microseconds, whereas accessing DASD is in the order of milliseconds. The difference in access time is a good reason to consider the use of large GBPs because reducing the number of times a synchronous I/O to disk is required.

The objective of this series of experiments is to observe the performance effect of the use of large GBPs by using a brokerage online transaction workload.

3.6.1 Performance measurements

A brokerage online workload that is running in a Db2 2-way data sharing environment is used to evaluate the performance affect of the use of large GBPs. The size of the GBPs is varied while the size of the local buffer pools of each member is fixed at 5 GB. The different runs use a total GBP size of 5 GB, 10 GB, 62 GB, 105 GB, 151 GB, and 566 GB. The combined size of all objects accessed during these measurements is approximately 900 GB. The same object size is used for all measurements that are described hereafter.

Figure 3-19 shows the effect of the use of larger GBPs on the number of synchronous reads for GBP-dependent objects. The larger the GBPs are, the less synchronous read I/Os are needed.

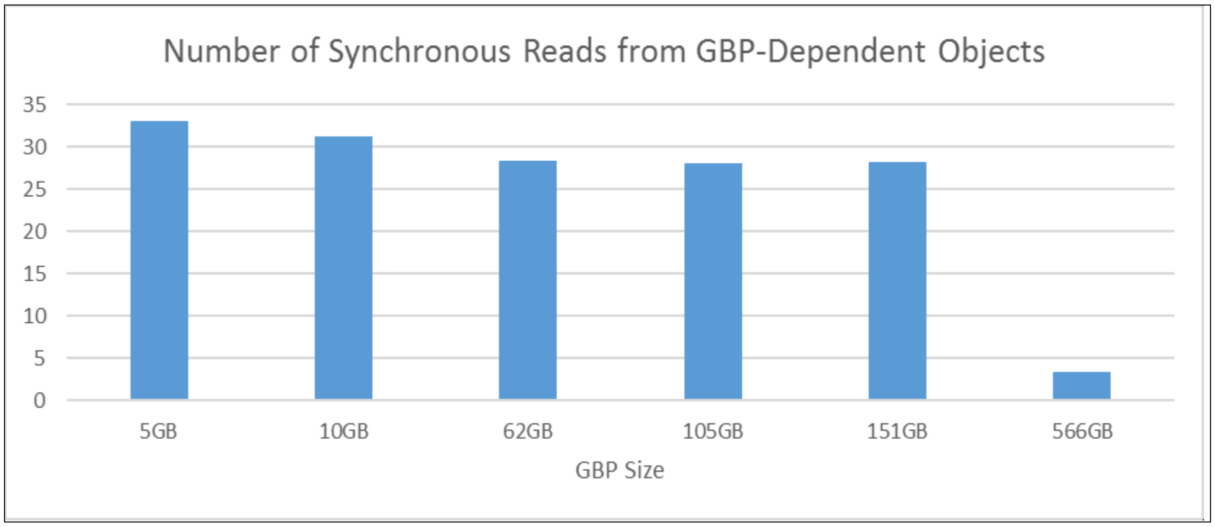


Figure 3-19 Sync I/Os for GBP-dependent objects

When the GBP size increases from 62 GB to 105 GB and 151 GB, hardly any reduction in the number of synchronous reads for GBP-dependent objects is observed. This result occurs because some GBP-dependent objects are large, their access pattern is much more random, and the increased GBP size did not make a significant difference in the number of I/O operations that is needed. Therefore, Db2 class 3 suspension time and throughput rate remain relatively the same, despite the increased size of the GBPs.

The runs with a total GBP size of 5 GB, 10 GB, 62 GB, 105 GB, and 151 GB use the (default) GBPCACHE CHANGED option, which indicates only changed pages are to be cached in the group buffer pool.

During the measurement that used a GBP size of 566 GB, the GBPCACHE ALL option was specified for several GBP-dependent objects, which allows the pages for those objects to be cached in the GBP as they are read in from DASD the first time. Subsequent access of those pages is from the GBP, which avoids synchronous reads.

When the number of synchronous I/Os decrease as the GBP size increases, a reduction in the Db2 class 3 suspension time is also expected, as shown in Figure 3-20.

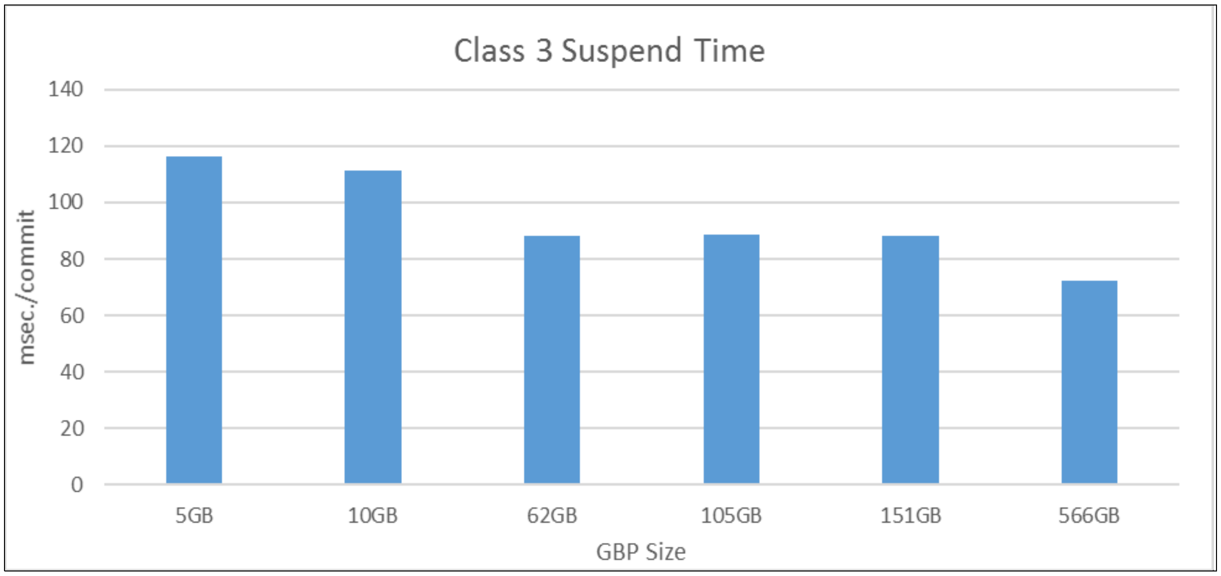


Figure 3-20   Class 3 suspension time with increasing GBP sizes

This workload is built in such a way that the faster the transactions are processed, the more transactions are triggered; that is, there is no “think time”. Therefore, shorter class 3 suspension times lead to a reduction in transaction elapsed time, which results in higher throughput, as shown in Figure 3-21.

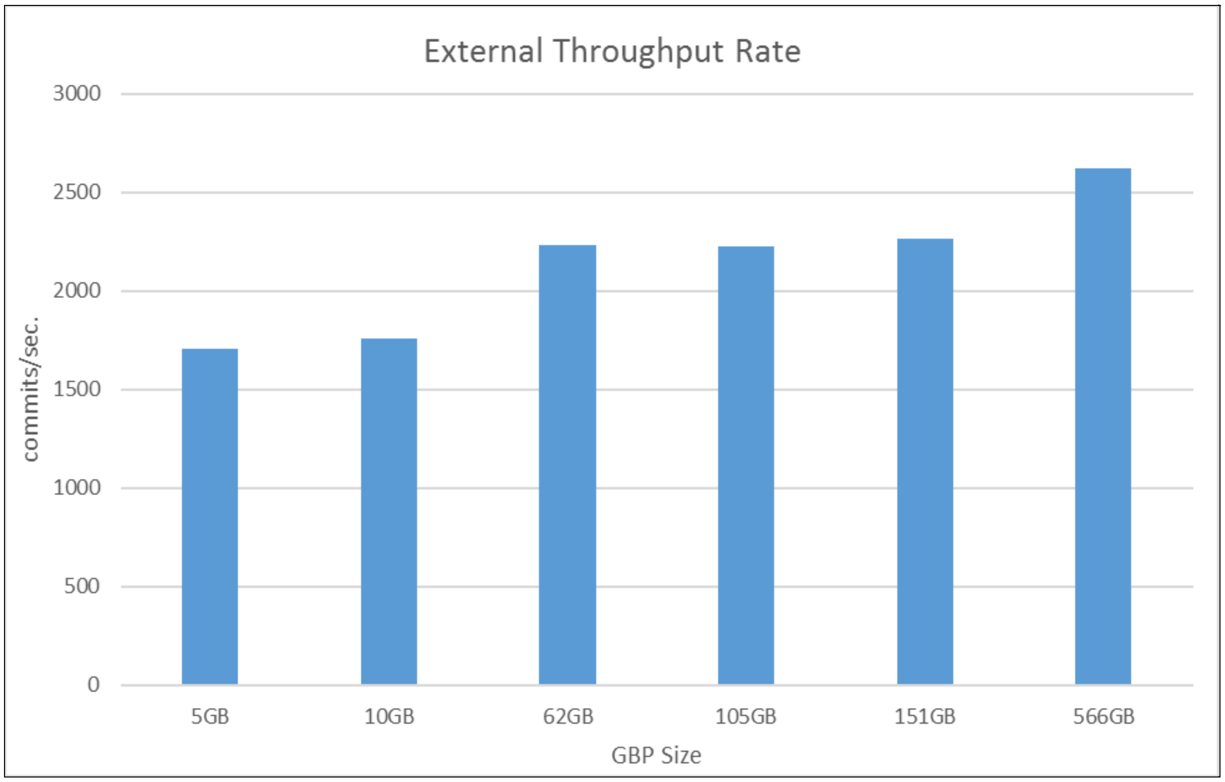


Figure 3-21   External throughput rate with increasing GBP sizes

As the size of the GBP becomes larger, less access to DASD is observed, while access to the group buffer pool in the Coupling Facility (CF) increases. Because of the zero think time model, the workload drives a higher transaction rate when the transactions run faster, hereby increasing the GBP activity.

In addition, because more pages are found in the GBP, the data must be transferred back from the GBP to the local buffer pool. Such a data request is more expensive than just checking whether a page is in the GBP. This process puts more pressure on CF CPU usage and CF links. The growing CF request rate as the GBP size increases is shown in Figure 3-22.

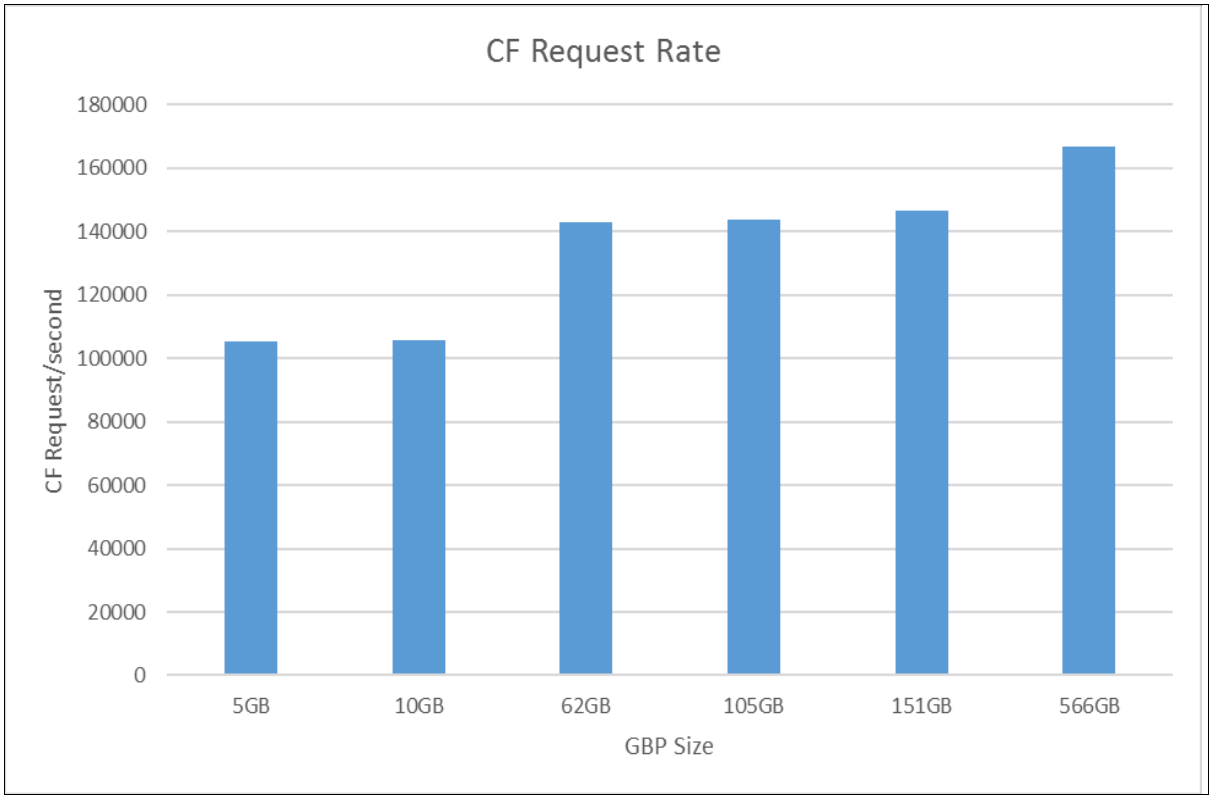


Figure 3-22 CF request rate with increasing GBP size

As a result, the CF service time becomes longer as the CF request rate increases and the number of times data is transferred back from the GBP to the local buffer pool also increases, as shown in Figure 3-23.

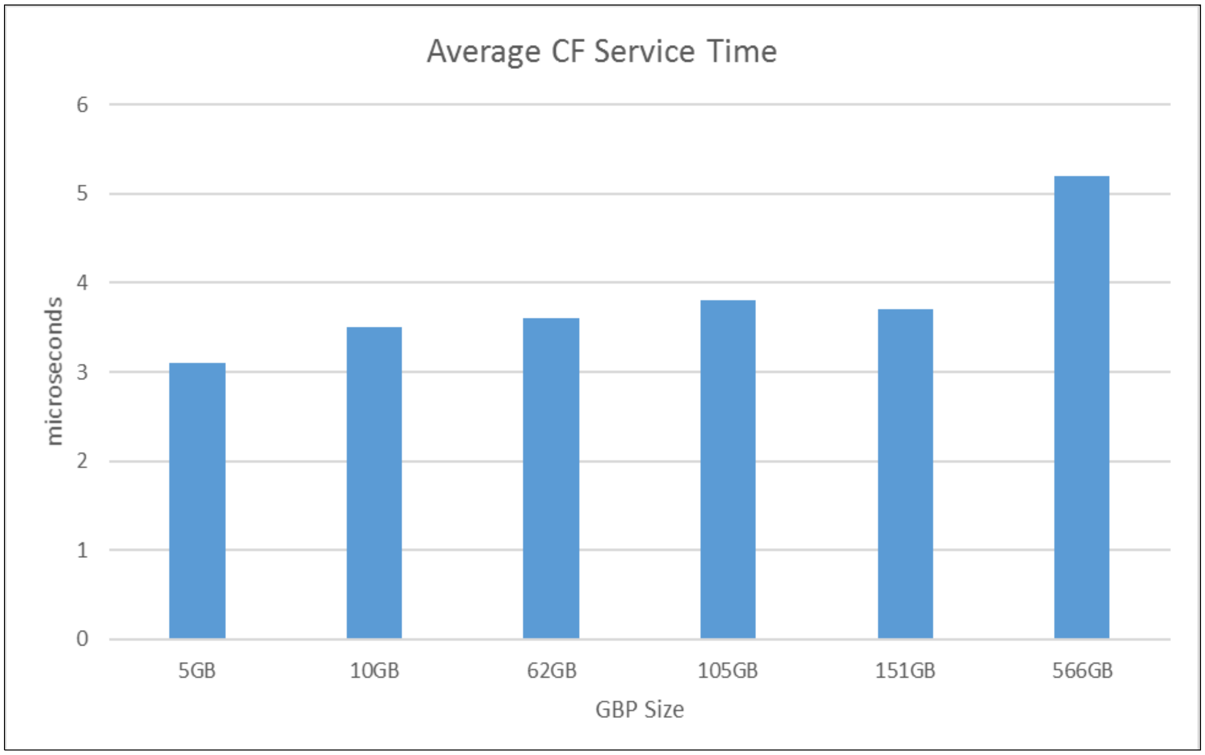


Figure 3-23 CF request service time with increasing GBP size

Longer CF service time has a direct effect on the transaction CPU time. As most CF requests are synchronous requests, the CPU is "spinning" (waiting for the request to complete). Therefore, the longer CF service time is, the more CPU time transactions use.

This process is an exercise of trading synchronous I/Os for GBP access to obtain better performance. The number of synchronous I/Os per transaction hovers from the high-20s to the mid-30s for GBP sizes ranging 5 GB - 151 GB. During those measurements, there is no clear relationship between CPU time and increased GBP size, as shown in Figure 3-24. However, when a 566 GB group buffer pool is used, synchronous I/Os are down to less than 5 per transaction, and there is clear performance advantage in terms of class 2 CPU time. At this point, a large group buffer pool clearly benefits this workload.

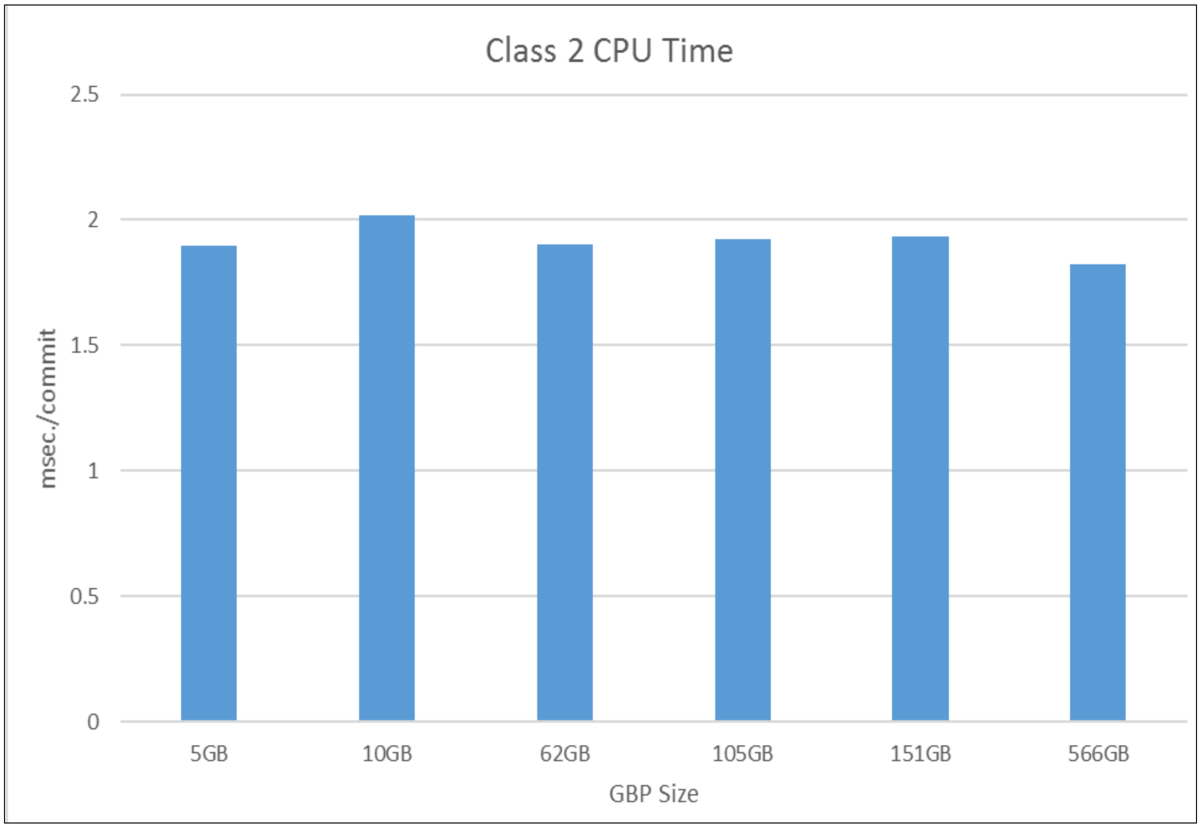


Figure 3-24 Db2 Class 2 CPU time with increasing GBP size

3.6.2 Summary

Large group buffer pools can help to minimize the number of synchronous reads for GBP-dependent objects. The access time to a group buffer pool is about three orders of magnitude faster than access to disk. This difference shortens the transaction's class 3 suspension time, which results in faster transaction response time and improves overall workload throughput.

The reduction in the number of synchronous reads for GBP-dependent objects should also benefit transaction CPU time. However, in our observation, access shifting from DASD I/O to CF requests results in longer CF service times in our environment and negatively affects transaction CPU time. It is not until synchronous I/Os are drastically reduced (566 GB measurement) that we begin to observe improved transaction class 2 CPU time.

3.7 Data set open/close performance

Currently, a maximum of 200,000 open data sets is allowed per Db2 subsystem. Opening a large number of data sets can be time consuming. z/OS release Version 2 Release 2 made some improvements to data set open (and close) processing to enhance data set allocation/deallocation time.

3.7.1 DFSMS changes in z/OS V2R2

DFSMS was enhanced to allow more and faster data set open (and close) processing.

Faster search for Access Method Block List

The Access Method Block List (AMBL) is a VSAM internal control block that is associated with opened clusters. Within VSAM, the AMBLs are organized in sequential chains. These chains can be long when many clusters are opened. Then, going through the chain to locate an AMBL (mainly during close processing) takes a long time and performance can be affected.

z/OS V2R2 addresses this issue with large AMBL chains and improved performance for Db2 linear clusters. VSAM now builds a binary tree to track the opens and closes for each cluster. In a sense, it looks similar to the volume table of contents (VTOC) Index, which is a binary tree that is pointing to VTOC. With this binary tree, a faster search can be done to determine the last open processing. With many open data sets, a significant improvement can be expected, though performance is not expected to improve when there are only a few open clusters.

Moving VSAM control blocks above the bar

Although the limit of open data sets in Db2 is 200,000, the practical number can be much less, depending on the virtual storage that is available below the bar. Having many open VSAM data sets (in the DBM1 address space) uses a large amount of virtual storage that must be in 31-bit addressable private storage in z/OS V2R1.

In DFSMS V2R2, some of the control blocks for linear clusters (VSAM LDS) are moved above the bar, if requested so by the Media Manager caller. This new feature can free up virtual storage that is allocated below the bar and allows media manager interface users to open more clusters than they did previously because the required storage is now allocated in 64-bit addressable memory.

The Media Manager interface for this improvement is not an externally documented interface. Only authorized callers (such as Db2) can request to use this feature.

3.7.2 Db2 for z/OS changes

Db2 12 uses this enhancement to allow more control blocks to be allocated above the bar.

With the PTF for APAR PI73101, changes were also made in Db2 12 to reduce the storage cushion to 10% of the region size and to reduce the amount of storage that is reserved for MVS usage to allow for more usable storage below the bar. This enhancement is also available in Db2 11 with APAR PI71764.

Use z/OS Version 2 Release 2 or higher with Db2 12 to take advantage of these performance enhancements.

Note: Functions that are provided in older releases can also be used to improve allocation performance, including S99DASUP and MEMDSENQMGMT.

3.7.3 Performance evaluation

Several measurements were conducted to observe the storage, CPU, and open/close data set performance effect with Db2 V12 by using z/OS 2.2 on a z13 processor with eight CPs. The measurements also had MEMDSENQMGT enabled, GRS STAR was used, and enhanced catalog sharing was enabled.

With 20 concurrent SQL jobs that were running to open 200,000 data sets, it took 465 seconds to open all the data sets. The measured total CPU time to open 200,000 data sets with z/OS 2.2 on z13 processor was 384 seconds. Storage usage peaked at 942.12 MB of 31-bit storage and 1151 MB of 64-bit storage for a total of 2093.12 MB of real storage used.

A DSMAX DSNZPARM value of 200,000 is the maximum value allowed in Db2 11 and 12.

Using the same workload, DSMAX was changed to 100,000 (by using the `-SET SYSPARM` command) to force the closing of 100,000 data sets. It took 6 minutes to close and reopen those data sets.

3.8 zHyperWrite

In update-intensive applications, the wait time for Db2 log write I/O (typically at commit time) often makes up a significant portion of the transaction latency, especially when disk synchronous mirroring is enabled.

zHyperWrite is designed to speed up the latency of log writes in a synchronous peer-to-peer remote copy (PPRC) environment. With zHyperWrite enabled, Db2 triggers a log write I/O to the secondary disk subsystem in parallel to the log write I/O to the primary disk subsystem. By overlapping the two I/Os (which are run serially without zHyperWrite enabled), a significant reduction in Db2 log write I/O time is possible.

zHyperWrite technology was introduced in the IBM DS8870 7.4 firmware release and requires corresponding software usage os PTFs for z/OS V2.1 (APARs OA45662, OA45125 and OA44973) and z/OS 2.2 base code. Db2 use of this feature was developed in Db2 12 and retrofitted back to Db2 10 and Db2 11 and is available through Db2 APAR PI25747. A new Db2 system parameter REMOTE_COPY_SW_ACCEL is introduced to enable or disable the zHyperWrite feature.

Although improved active log write performance can result in improved response time for Db2 operations, increased SRB time in the Db2 MSTR address space might be observed because the extra (Media Manager) I/O requests are handled under MSTR SRB process when zHyperWrite is enabled.

The benefit of zHyperWrite reduces as the distance of replication increases.

Performance measurements with zHyperWrite

A number of performance measurements were conducted to evaluate zHyperWrite by using a workload with concurrent batch inserts triggered by two members of a Db2 data sharing group. In this environment, PPRC is configured without any distance between the storage boxes.

As shown in Figure 3-25, a 40% elapsed time reduction was observed by using zHyperWrite, which resulted in a 22% overall transaction response time improvement.

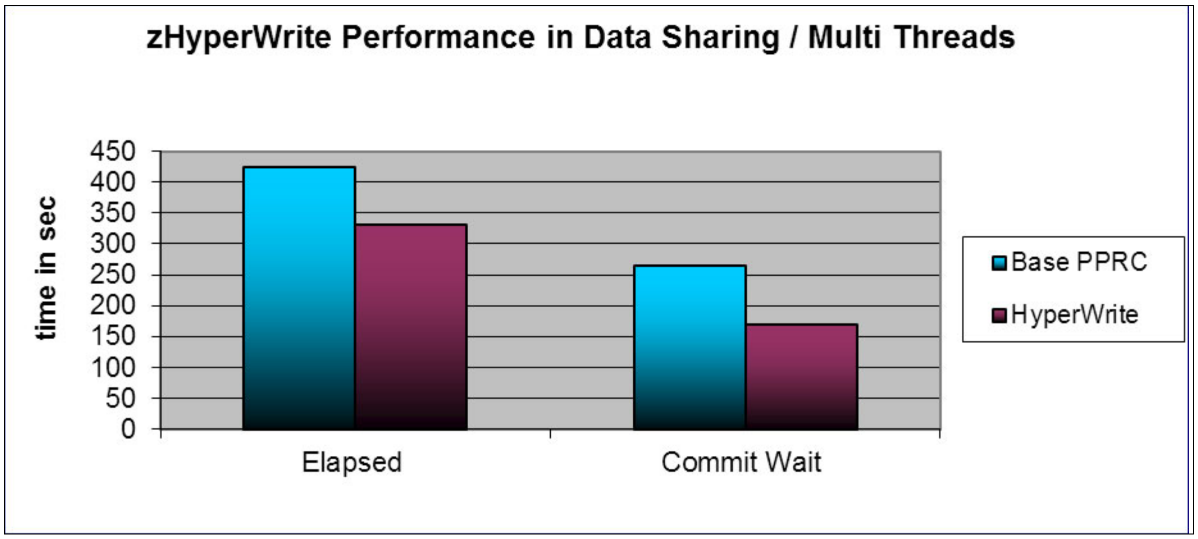


Figure 3-25 zHyperWrite performance

A minor CPU increase was observed in the Db2 MSTR address space to trigger the extra I/Os using zHyperWrite. The SRB time in MSTR address space is zIIP eligible starting with Db2 11.

In some cases, another indirect benefit of zHyperWrite can be observed. The reduced log write wait time (from zHyperWrite) can lead to reductions in other types of contentions, which in turn can result in more CPU time savings because Db2 has fewer contentions (suspend/resume activities) to manage.

3.9 Active log data set striping

VSAM data set striping for Db2 active log data sets were commonly used to improve the throughput processing that triggers heavy update activity. With significant improvement in disk subsystem technology, some of the bottlenecks that were seen in log write I/Os shifted and in those cases, the use of active log striping does not add any benefit.

Should you strip the active log data sets? Because many factors influence whether it is beneficial to stripe the active log, a simple answer to this question is no longer available.

In this section, we describe the basic process that is used in log striping and the results from measurements that were taken in a controlled environment at the Silicon Valley Lab.

Note: This section is not meant to be a comprehensive study of active log striping. For example, the environment that we used did not include synchronous disk mirroring (Peer-to-Peer Remote Copy). Our intention is to demonstrate some of the factors and considerations that influence the performance of active log striping.

3.9.1 How Db2 log striping works

A striped Db2 active log data set uses VSAM striping to spread log data across multiple volumes. To create a striped log data set, more than one volume must be specified for the VSAM cluster, the SMS data class definition for the data set must use the extended type (EXT), and the sustained data rate must be greater than zero for the SMS storage class.

The number of volumes that is specified for allocation of the active log data set becomes the number of stripes. Allocating across two volumes results in two stripes, four volumes are four stripes, and so on. When the stripe count is greater than one, a single log I/O request from Db2 is transformed to a number of parallel I/O requests. This process is done by DFSMS and the z/OS storage management software, and is transparent to Db2.

Db2 active log striping

After log striping is enabled, Db2 log data is divided into control interval (CI) size chunks and spread across multiple volumes during the I/O operation. Because the smallest unit of I/O to a volume is a CI, striping across multiple volumes occurs only when the log I/O size is greater than one CI size. The CI size of the Db2 active log data sets is normally 4 KB (we assume that the CI size is 4 KB for the rest of the section).

The degree of I/O parallelism depends on the log I/O request size and the number of stripes. For example, if Db2 triggers a log write request of 32 KB and the number of stripes is four, the request is converted to four concurrent I/Os of two CIs each to the four volumes. In the case of using two stripes, the same log write request of eight CIs is converted to two concurrent I/Os of four CIs each to two volumes.

In both cases, a log write request should complete faster with striping when compared to non-striping, as the I/O to multiple volumes are done in parallel. However, the performance benefit varies depending on the log request size. If the average log request size is less than a CI size of 4 KB, no advantage is gained in striping the log.

The Db2 log I/O size is determined by the amount of logging that must be done, the frequency of commit, and the number of concurrent update transactions. The log I/O size of online transactions tends to be small, as this type of workload tends to have a smaller log size and uses frequent commits. The log I/O size of update-intensive batch jobs tends to be larger with less frequent commits.

A larger log I/O request size benefits the most from striping, as more parallel I/O can be achieved. If the log I/O request size is 128 CIs, four stripes result in four concurrent I/Os and two stripes result in two parallel I/Os. In this example, four stripes will yield less log wait time and better log throughput. However, if the log I/O request size is two CIs, four stripes result in only two parallel I/Os and two stripes also result in two parallel I/Os. In this case, four and two stripes yield similar log wait time and log throughput; therefore, no advantage is gained by using four stripes. The performance benefit of striping size depends on the average size of the log I/O and the number of stripes.

3.9.2 Performance measurements

To better understand the factors that influence log striping performance, several tests were conducted with and without striping, and the effect on the workload's performance was measured.

The active log data sets were allocated on volumes on IBM storage DS8870 with 15 KB RPM drives and 16 GB high-performance IBM FICON® channels. Each extent pool in the DS8870 had three 6+P RAID 5 arrays with rotate extents. The Db2 active log data set size was approximately 4 GB. For striped logs, each volume was from a different extent pool, which ensured that each volume was on a different set of physical disks.

OLTP and batch INSERT workloads were run with different number of stripes for the Db2 active log data sets. These tests were conducted in a controlled environment with no other work on the z13 system and the DS8870 storage control unit.

For the OLTP workload, the logs share the extent pools with user data. The OLTP workload generates 75% reads and 25% writes (10% of the write requests are from log writes) to the DS8870 control unit. The average size of a log write I/O was 2.6 CIs, or about 10.6 KB.

The results of the test are listed in Table 3-2.

Table 3-2 OLTP workload using active logs with a variable number of stripes

OLTP Workload (400 concurrent users)	**Non-striped log**	**Log stripe = 2**	**Log stripe = 4**	**Log stripe = 8**
Average transactions per second	4779	4804	4834	4820
Average database I/O time (MBps)	6.405	6.510	6.382	6.439
Average log write I/O time (MBps)	1.936	1.010	0.894	0.963
MSTR CPU time per commit (MBps)	0.0005	0.0008	0.0009	0.0100

As the number of stripes for the active logs were increased, the average log write I/O time went down up to four stripes. Because the average number of CIs per log write was 2.6, the most parallelism we observed was three I/Os. Therefore, we saw a reduction in the average log write I/O time from no stripes to four stripes, but no other advantage was observed by going to eight stripes.

An increase in the MSTR CPU time that is associated with striping was observed. This result occurred because of the associated CPU cost in the media manager from issuing more I/Os per log write request. Depending on where the log data sets are stored, the database I/O performance varied a little among the tests.

As listed in Table 3-2, although there was a reduction in the log write wait times, there was no significant affect on the overall transaction response time. Most of the transaction elapsed time was spent reading the database (synchronous I/O wait), rather than waiting for the log write to occur at commit time.

The average database I/O time in the tests is rather high. The workload uses large buffer pools (70 GB in total) and when an occasional I/O occurs, the data is often not found in the DASD controller cache and the data must be accessed from the spinning disks. The use of solid-state drives (SSDs) might help to reduce the database I/O times, but they were not used during these tests.

In addition to an OLTP workload, several batch INSERT workloads were measured with and without striping the active log data sets. The first set of batch INSERT workloads runs 200 concurrent threads with 200 inserts per commit. This run generates the maximum possible log write size of 128 CIs per log write I/O request. During these tests, each volume that was used by the striped active logs was on a different extent pool. However, the logs shared the extent pools with the user data.

The results of these measurements are listed in Table 3-3.

Table 3-3 Batch insert workload using 128 CI log write I/Os with a variable number of stripes

Batch Insert (200 isrt/cmt from 200 thds)	**Non-striped log**	**Log stripe = 2**	**Log stripe = 4**	**Log stripe = 8**
#INSERTs per second	542,000	744,000	918,000	1,091,600
Average. log write I/O time (MBps)	69.658	49.917	39.310	32.874
MSTR CPU time per commit (MBps)	0.030	0.035	0.037	0.041
LOG rate for 1 log (MBps)	219.58	301.30	372.00	442.58

With 128 CIs per log write request, the average log write time was reduced substantially when more stripes were used. Therefore, the insert rate and log write rate throughput increased with more stripes. The Db2 MSTR CPU time increased slightly as more stripes were used. However, this increase in MSTR CPU time is zIIP eligible. Therefore, this set of test results prove large log writes can benefit greatly from striping.

The second set of batch INSERT measurements used right concurrent threads with 50 inserts per commit. This configuration generated an average of 12 CIs per log write request, which is one track of data per I/O. The results of these tests are listed in Table 3-4.

Table 3-4 Batch insert workload using 12 CI log write I/Os

Batch Insert (50 isrt/commit from 8 thds)	**Non-striped log**	**Log stripe = 4**
#INSERTs per second	288,700	365,400
Avg. log write I/O time (MBps)	0.814	0.545
MSTR CPU time per commit (MBps)	0.222	0.226
LOG rate for 1 log (MBps)	116.44	147.66

With 12 CIs per log write request, we saw a similar performance benefit in terms of insert rate as with 128 CIs per log write request for a four-stripe active log, compared to a non-striped active log. However, less increase in log write throughput (27% versus 69%) was observed.

A third set of batch INSERT measurements were run, with and without striping. They both ran 10 concurrent threads with 1 insert per commit. This configuration generated an average of 1.1 CI per log write request. The results are listed in Table 3-5.

Table 3-5 Batch insert workload using 1.1 CI log write I/Os

Batch Insert (1 isrt/commit from 10 thds)	**Non-striped log**	**Log stripe = 4**
#INSERTs per second	14,900	17,800
Avg. log write I/O time (MBps)	0.465	0.357
MSTR CPU time per commit (MBps)	0.053	0.053
LOG rate for 1 log (MBps)	19.83	23.79

Even with only 1.1 CI written per log write I/O request, we still observed a benefit by using active log striping. However, when compared against the 12 CI per log write requests with four stripes scenario, the improvement lessened in log write I/O throughput (20% versus 27%).

3.9.3 Usage considerations for Db2 active log striping

The expected benefit from log striping depends on many factors, such as hardware configurations, commit frequency, the log size being written, a=nd disk and channel utilization.

Log write I/O size

Log striping tends to benefit batch workloads the most because the log write size for batch workloads is generally considerably larger compared to OLTP workloads. As demonstrated in our batch measurement tess, the larger the log write size, the more the workload benefits from log striping as the log requests are split into multiple parallel I/Os.

Number of stripes and log write I/O size

With a large log write I/O size, such as when writing 128 CIs per log write I/O, eight stripes resulted in a good benefit. With smaller log write I/O sizes, the use of eight stripes did not provide as much benefit.

Disk configuration and location of the active logs

OLTP workloads can gain benefits from the use of log striping if non-striped active logs are becoming hot spots on disks. By striping to volumes on different extent pools, we can spread the log I/O to reduce disk contention.

The performance gain from striping active log data sets can be larger when there is hardware contention. However, the benefit can be minimal when there is no disk contention, such as when dedicated control units are used for the Db2 logs. By means of demonstration, an OLTP brokerage workload was run with the Db2 logs and data that was allocated on separate storage control units, and with the Db2 log and data allocated on the same storage control unit for non-striped and striped active log data sets. The results are shown in Figure 3-26.

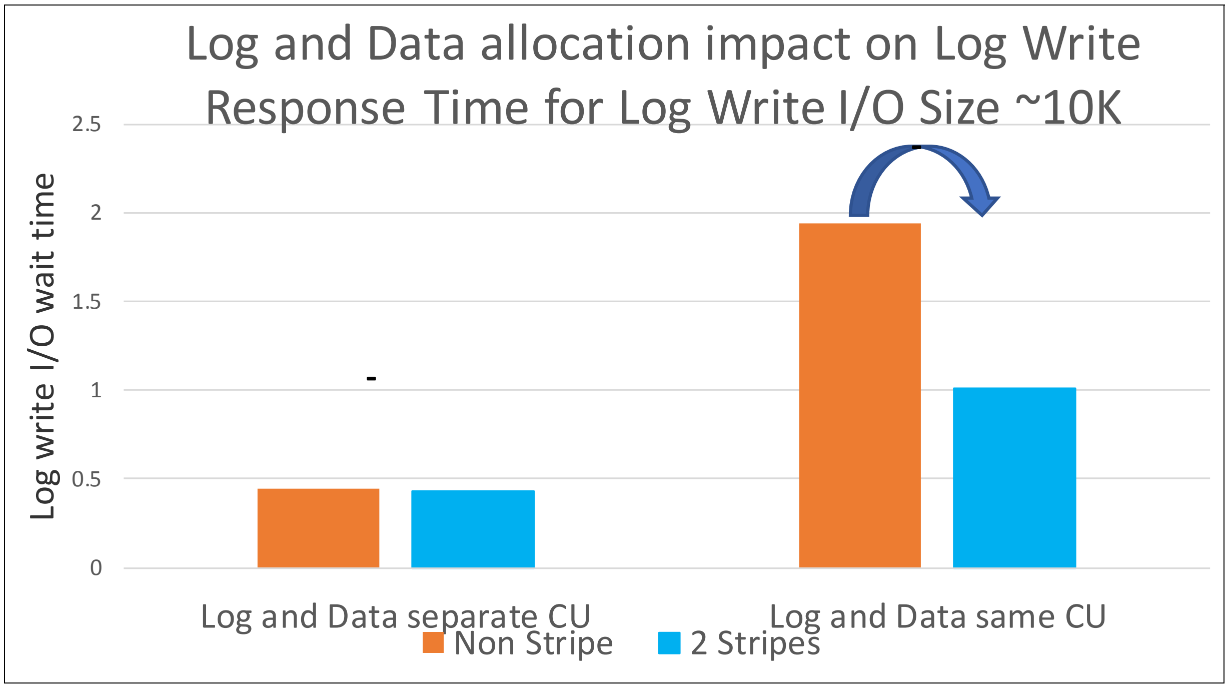


Figure 3-26 Effect of location of active logs on non-striped versus striped log data sets

When the Db2 log and the data are on separate control units, the log I/O wait time is small for the non-striped case because no contention with the data exists. Therefore, the benefit of striping when the Db2 log and the data are on separate control units is miniscule.

However, when the Db2 log and the data are sharing the same control unit, we observed a 48% reduction in log I/O wait time with striping. This issue occurs because when the log and the data are on the same control unit, hardware contention resulted without striping. With striping, hardware contention was considerably reduced by spreading the log data sets across multiple stripes.

Disk mirroring

Whether log striping is enabled, log I/O is susceptible to problems that are caused by remote DASD replication. Each I/O introduces a risk of a potential delay because log write I/O is synchronous to the Db2 transaction. Also, because striping creates more I/Os, striping might increase the risk of a delay.

Hardware configuration considerations

If possible, each log data stripe should be on a different RAID rank to avoid potential contention. If logs and data are on the same RAID rank, avoid sharing RAID ranks with update intensive tables and indexes that cause (many) asynchronous writes, such as Db2 deferred writes and castout writes. To reduce the disk constraint, allocate Db2 active logs on the fastest disk available, such as 15 KB RPM drives or even better, allocate them on SSDs.

3.10 Easy Tier directive data placement support by Db2 REORG utility

Multi-tier storage devices, such as the DS8870 and DS8880, with software, such as IBM Easy Tier®, provide management of data sets across SSDs, fast hard disk drives (HDDs), and slow disk drives that are based on usage. Data set extents that are accessed frequently are considered “hot” and Easy Tier directs the extents to be migrated to SSDs (if installed).

Conversely, data set extents that are rarely or never accessed are migrated to the fast (but expensive) HDDs or even to the slow (less expensive) HDDs (if installed).

A highly used Db2 table space or index likely is migrated, at least in part to the SSDs over time. When the table space is then reorganized, the newly created data sets likely are not allocated on SSDs but on slower or slowest HDDs. Customers then experience increased I/O time after a reorganization until Easy Tier recognizes the data sets are hot and migrates them to an SSD.

Conversely, the original data set extents are still considered hot for some time until Easy Tier determines they are no longer in use. Data sets are managed by DFSMS and the DS8870 has no knowledge that the data sets were deleted, and valuable SSD space cannot be reused immediately.

New functionality called *directive data placement* was added to the DS8K (introduced by DS8870 microcode level R7.4), and DFSMS Media Manager added new interfaces to allow Db2 for z/OS REORG SHRLEVEL CHANGE and SHRLEVEL REFERENCE utilities to copy the “temperature signature” from the old data sets to the new data sets and then, at completion of the reorganization, to set the temperature of the old data sets to “cold”.

The DFSMS Media Manager support is available in z/OS 1.13 and above. If z/OS 1/13 or z/OS 2.1 is used, PTF for APAR OA46482 must be installed.

APAR PI82904, which is still open at the time of this writing, provides the support for this enhancement in Db2 12. This functionality is available in Db2 10 and 11 for z/OS by using APAR PI35321.

4

Data sharing

In this chapter, we describe the improvements to data sharing that were introduced with Db2 12 for z/OS. The main focus of these functions is performance and availability, and most of them do not require user action to take effect.

This chapter includes the following topics:

4.1 Global commit LRSN enhancement

An IRLM lock request is a fairly expensive operation. New lock requests can conflict with existing locks and create lock contentions. Also, lock contentions typically result in longer transaction elapsed times. For a long time, Db2 was implementing *lock avoidance* techniques to avoid acquiring locks whenever the semantics of the query allow it to do so. Ideally, more effective lock avoidance means taking fewer locks, which reduces lock contentions to improve transaction elapsed times.

One of the techniques that is 0used by lock avoidance is called commit LRSN (CLSN) checking. For every page in any page set, Db2 tracks the last “time” that page was updated. The instance is an RBA (for non-data sharing) or an LSRN (for data sharing) value. To explain how commit LRSN (CLSN) checking works, we call this value (A).

Db2 also tracks the instance of the oldest uncommitted activity for every page set or partition. This value is called the commit LRSN (or CLSN value) and the value is an LRSN or an RBA value, depending whether it is data sharing. We call this value (B). Consider the following points:

- For non-GBP-dependent page sets, the CLSN is at the page set level. It is the LRSN of the oldest uncommitted unit of recovery for all the active transactions for that specific page set.
- For GBP-dependent page sets, a Global CLSN value is maintained for the entire data sharing group. It is the LRSN of the oldest uncommitted unit of recovery across all members of the data sharing group and across all pages or partitions (regardless of whether the object is GBP-dependent).

The lock avoidance that is based on CLSN checking depends on the fact that if the value (A) is less than (B), Db2 can be sure that everything on the page can be committed. Therefore, there is no need to get a lock on the page or row (if lock avoidance is allowed).

Otherwise, if value (A) is greater than (B), Db2 performs another lock avoidance check before acquiring a lock. However, that check is not relevant to the Db2 12 enhancement that is described in this section.

For more information about lock avoidance, see *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725.

Db2 11 uses a single Global CLSN value for the entire data sharing group for GBP-dependent page sets. To take better advantage of the Db2 lock avoidance logic, all applications that access data concurrently should commit frequently to advance the Global CLSN quickly. When not doing so, a single transaction that does not commit can hold back the Global CLSN, which keeps the entire data sharing group from implementing lock avoidance effectively.

Db2 12 was enhanced to overcome situations in which a single (or a few) transactions that do not commit frequently can ruin lock avoidance for all users.

Db2 12 still tracks the CLSN value at the page set or partition level as before. Also, each member uses a system agent to maintain a list of the (approximately) 500 oldest CLSN values and their corresponding page set or partition. This information is stored in the SCA structure in the coupling facility (CF) to share among the members.

When the CLSN lock avoidance check for a GBP-dependent object is done in Db2 12, the list with 500 oldest CLSNs is consulted first. If the page set that is being accessed is in the list, that CLSN is used. If the page set is not in the list, the highest CLSN value in the list is being returned as the global CLSN value. This value should be considerably better than the Db2 11 case where the lowest CLSN value is used as the global CLSN value to be checked against by the lock avoidance logic.

In a data sharing environment, each member also needs a global view if other members also are using this object. Therefore, the global list of object-level CLSN values is stored in the Db2 SCA structure for all members to check.

This feature is available in Db2 12 at function level V12R1M500 and higher.

A new LRSN keyword was added to the **`-DISPLAY DATABASE`** command. It shows the commit LRSN and read LRSN for a specific page set. For more information about the read LRSN and sample command output, see "Displaying the commit and read LRSN" on page 135.

4.1.1 Performance effect of not committing frequently before Db2 12

The chart that is shown in Figure 4-1 shows the effect a single, long-running unit of recovery (UR) can have on the number of lock and unlock requests of other transactions in the system. In this example, a single, long-running transaction performs database updates but does not issue frequent COMMITs. The transaction is holding an exclusive lock on a single object. Although the object is not used by the classic IRWW workload that is being measured here, its performance is greatly affected by its CLSN not advancing quickly.

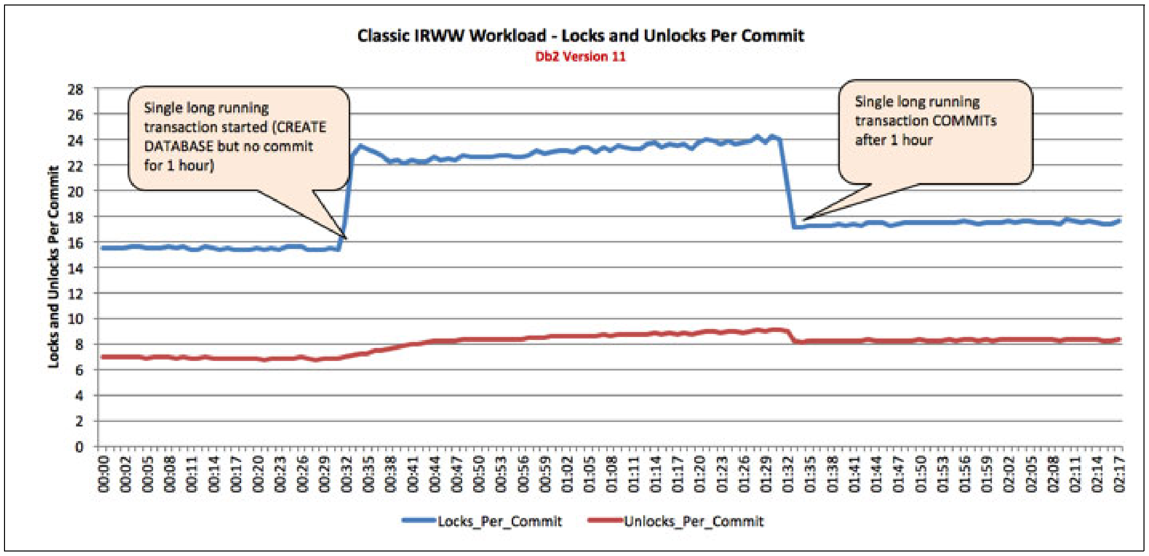


Figure 4-1 IRWW workload with a long running UR - locks and unlocks per commit in Db2 11

In a data sharing environment, Db2 uses a global (group-wide) CLSN to determine whether lock avoidance can be used. In these situations where a single UR is not committing, the global commit LSN cannot move forward and Db2's lock avoidance does not work effectively.

As a result of ineffective lock avoidance, the number of lock and unlock requests increases, which leads to an increase in Db2 CPU usage. For the two-way Classic IRWW workload that is used in this scenario, the Db2 class 2 CPU time increased by 7.6% when a single misbehaving transaction caused lock avoidance to be ineffective.

4.1.2 Performance effect of not committing frequently in Db2 12

By providing greater granularity for the commit LSN, data sharing workload performance can improve significantly by using Db2 12. Instead of keeping a single global CLSN for all GBP-dependent objects, more commit LSN values for a maximum of (about) 500 object-level (page set or partition) are tracked.

Figure 4-2 shows that the use of the new Db2 12 design, long-running “misbehaving” transactions (those transactions that do not commit frequently) no longer cause other transactions to suffer. Db2’s lock avoidance strategy works effectively, even if there are many objects with open units of recovery for a long time because Db2 now maintains a list of oldest CLSNs on a per object basis instead of having everything depend on the single global CLSN for all GBP-dependent objects.

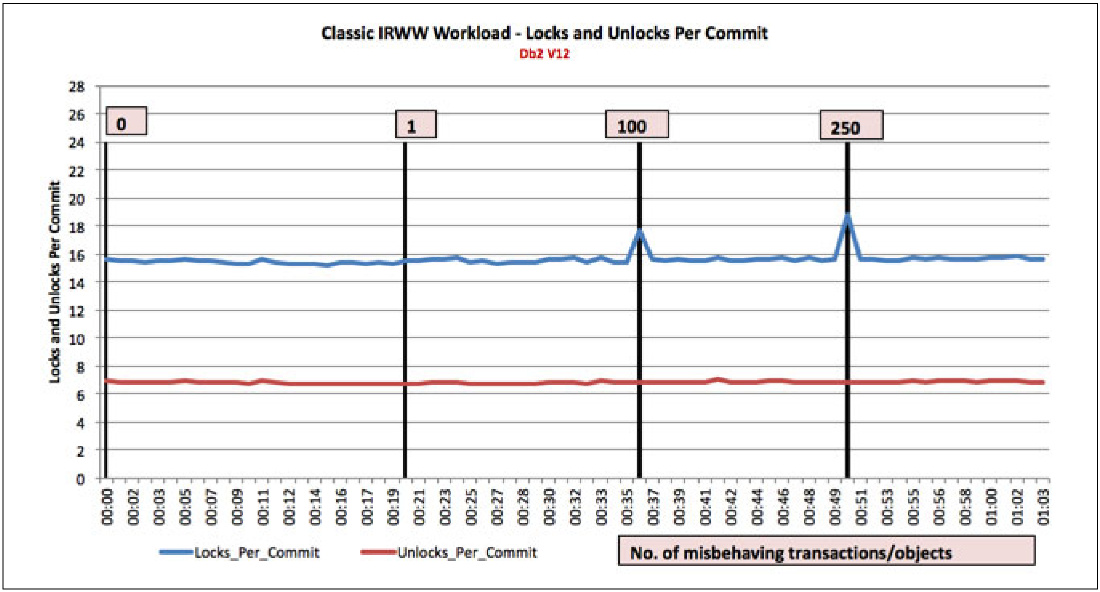
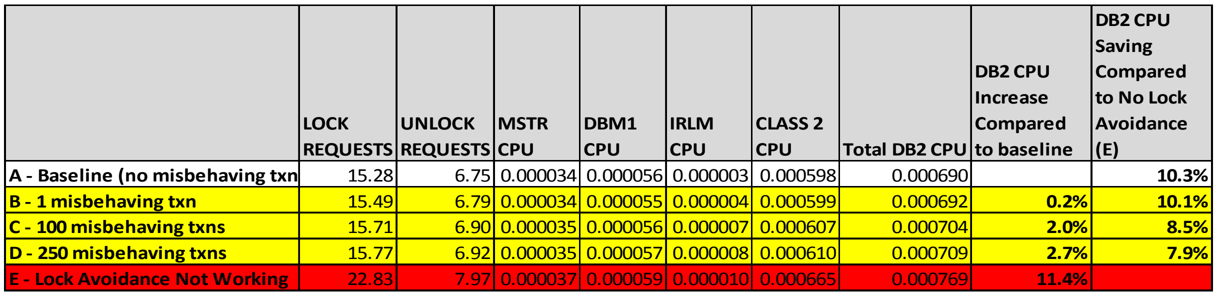


Figure 4-2 IRWW workload with a long running UR - locks and unlocks per commit in Db2 12

With this enhancement, a single misbehaving transaction that is accessing an object that is unrelated to the IRWW workload no longer has a negative influence on the amount of lock avoidance that is used by the transactions of the IRWW workload, as they did in Db2 11. Also, when more misbehaving transactions against other objects are introduced, there is no significant effect on the IRWW workload.

The detailed metrics for the test scenario are shown in Figure 4-3.



	LOCK REQUESTS	UNLOCK REQUESTS	MSTR CPU	DBM1 CPU	IRLM CPU	CLASS 2 CPU	Total DB2 CPU	DB2 CPU Increase Compared to baseline	DB2 CPU Saving Compared to No Lock Avoidance (E)
A - Baseline (no misbehaving txn	15.28	6.75	0.000034	0.000056	0.000003	0.000598	0.000690		10.3%
B - 1 misbehaving txn	15.49	6.79	0.000034	0.000055	0.000004	0.000599	0.000692	0.2%	10.1%
C - 100 misbehaving txns	15.71	6.90	0.000035	0.000056	0.000007	0.000607	0.000704	2.0%	8.5%
D - 250 misbehaving txns	15.77	6.92	0.000035	0.000057	0.000008	0.000610	0.000709	2.7%	7.9%
E - Lock Avoidance Not Working	22.83	7.97	0.000037	0.000059	0.000010	0.000665	0.000769	11.4%	

Figure 4-3 IRWW workload with a varying number of long URs: Lock, unlock, and CPU time effects

Consider the following points:

- There is no DBM1 or MSTR CPU time affect as the number of misbehaving transactions or objects increased from 0 to 250.
- A slight increase in CPU time can be observed when the number of misbehaving transactions or objects increased to 100 and 250, due to the longer list of objects to manage object-level commit LRSN values for. However, the increase is significantly lower compared to what the increase would be if lock avoidance was not effective.

4.1.3 Conclusion

The performance measurements that use the classic IRWW workload show that with the Db2 12 global CLSN enhancement, the effect of transactions that do not commit frequently on other concurrent transactions can be mitigated to a great extent.

Transactions that must access the same objects that are being updated by the misbehaving transactions still suffer because those objects' CLSN value are still kept from advancing until the bad transactions commit. Therefore, it is still important to commit frequently and avoid long running (updating) transactions.

4.2 Global read LSN enhancement

Similar to tracking the (global) commit LSN for updating transactions, Db2 also keeps track of the read interest by using the Read LSN or RLSN at the buffer pool level.

Before Db2 12, the read LSN for a BP in a data sharing environment is the minimum value across all the Db2 members for that BP. The read LSN tracks the oldest read claim against an object (or objects in a buffer pool) and is used by Db2 space management to determine whether deleted space can be reused. A good example is to determine whether it is acceptable to reuse space from deleted LOBs. Db2 must ensure that nobody is still using those deleted LOBs before the space is reused for new LOBs. In certain cases, deleted LOB space is not reused effectively because of an infrequently updated read LSN resulted from transactions that do not commit frequently, which causes the LOB space to grow prematurely.

Db2 12 improves performance and space reuse by providing greater granularity for the read LSN. These values are kept at object level (page set or partition) in each Db2 member's memory. Each member also tracks a maximum of 500 object-level oldest read LSN values to be exchanged with other data sharing members, as each member also needs a global view of which other members are using these objects.

Therefore, the 500 oldest object-level read LSN values are also stored in the SCA for all members to access, which is similar to the Db2 12 commit LSN enhancement.

Displaying the commit and read LRSN

A new LRSN keyword was added to the -DISPLAY DATABASE command. It shows the current commit LRSN and read LRSN value for a specific page set. For more information about commit LRSN, see 4.1, "Global commit LRSN enhancement " on page 132.

Issuing the `-DISPLAY DATABASE(TESTTD) SP(TESTTS) LRSN` command produces the output that is shown in Example 4-1 on page 136. It shows the 10-byte read LRSN and commit LRSN for table space TESTTS in hexadecimal, and the value that is converted into local time format.

Example 4-1 -DIS DB LRSN command output

```
DSNT360I  -DB2B ***********************************
DSNT361I  -DB2B *  DISPLAY DATABASE SUMMARY
                *  GLOBAL LRSN
DSNT360I  -DB2B ***********************************
DSNT362I  -DB2B     DATABASE = TESTTD  STATUS = RW
                    DBD LENGTH = 4028
DSNT397I  -DB2B
NAME   TYPE PART  LRSN                       TIMESTAMP
------ ---- ----- -------------------------- -------------------
TESTTS TS    0001 COMMIT 00D0C8E70FAA8BE90000 05/23/2016 01.23.37
                  READ   00D0C8E70FAA8BE90000 05/23/2016 01.23.37
******* DISPLAY OF DATABASE TESTTD   ENDED    ******************
DSN9022I  -DB2B DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

If you suspect that something is holding back the commit LRSN or read LRSN, issuing the command more than once is an easy way to verify whether the values are moving forward or stalling.

4.2.1 Performance evaluation

A customized workload was used to evaluate the performance effect of the enhanced RLSN tracking. The workload consists of two separate programs that access two separate tables that contain LOB columns. The measurements were done by using a two-way data sharing group setup. The test scenario is as follows:

- Program #1, running on Db2 member 1, performs SELECT operations from a LOB column from Table #1 every 3 seconds without committing.
- Program #2, running on Db2 member 2, first performs 400,000 DELETEs (committing every 10 rows) from Table #2 containing a LOB column, followed by 400,000 INSERTs (again committing every 10 rows) into Table #2.
- Table #1 and Table #2 and their associated LOB table spaces all use the same buffer pool.

The objective was to measure the effect on Db2 CPU time of Program #2, and to evaluate the space reuse for Program #2.

The key metrics from these tests are summarized in Table 4-1.

Table 4-1 Read LSN enhancement measurement results

Per commit	Without Read LSN enhancement baseline (A)	With Read LSN enhancement (B)	Delta % (B-A)/A
INSERT functionality			
Class 1 Elapsed time (sec)	0.063282	0.013549	-78.6
Class 2 Elapsed time (sec)	0.062400	0.012687	-79.7
Class 1 CPU time (sec)	0.040667	0.001805	-95.6
Class 2 CPU time (sec)	0.040430	0.001576	-96.1
GETPAGES	9,807.01	1,048.17	-89.3
BUFFER UPDATES	9,012.96	214.57	-97.6

EXT/DEL/DEF (sec)	0.011522	0.000009	-99.9
NACTIVE - BEFORE running workload	8,545,320	8,545,320	
NACTIVE - AFTER running workload	15,019,200	8,545,320	-43.1
DELETE functionality			
Class 1 Elapsed time (sec)	0.014918	0.014351	-3.8
Class 2 Elapsed time (sec)	0.014280	0.013709	-4.0
Class 1 CPU time (sec)	0.000834	0.000847	1.6
Class 2 CPU time (sec)	0.000645	0.000656	1.7
GETPAGES	122.18	122.18	0.0%

The following key observations can be made from the workload's performance data that is listed in Table 4-1 on page 136:

- Using the enhanced read LSN functionality resulted in complete reuse of the space during INSERT processing into the table that contains the LOB. This empty space was created by the DELETEs that preceded the INSERTs.
- The INSERTs also use significantly less CPU and the number of GETPAGES is significantly reduced because the space search algorithm is much more effective now that it has a more granular read LSN value to determine whether space can be reused.
- No significant regression for the DELETE operations was observed as read LSN is not required for delete operations.

4.2.2 Conclusion

Based on the measurements, it is safe to conclude that the new functionality that maintains a more granular read LSN value can result in performance and space management improvements for tables that contain LOB columns.

With this enhancement, isolating certain objects that are used by misbehaving transactions in a separate buffer pool to improve space reuse for other objects that are accessed by "well-behaved" transactions should no longer be required.

4.3 Asynchronous lock structure duplexing

Running some or all of the SCA, lock, and group buffer pool structures in duplex mode is one way to achieve high availability for these structures across many types of failures (including lost connections and damaged structures) in a Db2 data sharing environment.

Db2 lock structures in the CF are managed by XES and can be set up with system-managed duplexing for high availability in a CF failure. This setup is often used to avoid "double" failures, which are a failure of a Db2 member (including its IRLM) and a failure of the lock structure at the same time. An example of a double failures situation is when a machine that hosts a z/OS LPAR running Db2 and the ICF LPAR where the lock structure of that Db2 resides fails.

4.3.1 System-managed duplexing protocol

System-managed duplexing for any CF structure requires every structure update request to occur in both the primary and the secondary structures at the same time. Two identical commands are sent to both structures to be run in parallel, and both must complete successfully before the update result is returned to the exploiter, which is IRLM in Db2's case.

Figure 4-4 shows the sequence of operations that are involved in the current synchronous duplexing protocol (or system-managed duplexing) for lock structures between two coupling facilities CF1 (primary) and CF2 (secondary). Each step from the time a lock structure update request is sent by the exploiter (Db2 IRLM in this case) to XES and receiving the response back by the exploiter is shown in Figure 4-4.

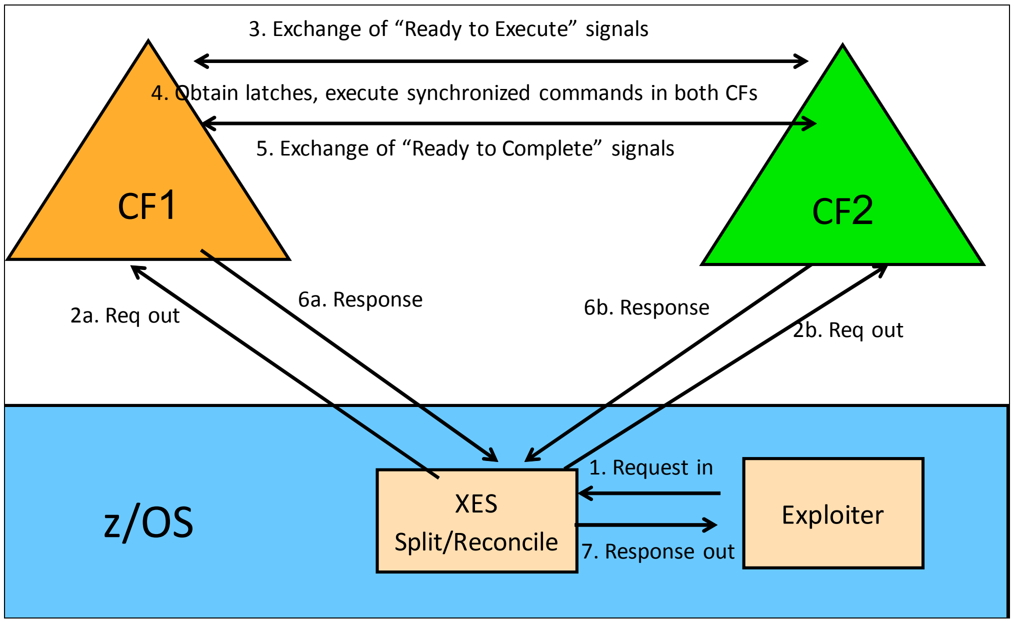

3. Exchange of "Ready to Execute" signals
4. Obtain latches, execute synchronized commands in both CFs
5. Exchange of "Ready to Complete" signals
CF1
CF2
2a. Req out
6a. Response
6b. Response
2b. Req out
z/OS
XES
Split/Reconcile
1. Request in
7. Response out
Exploiter

Figure 4-4 System-managed duplexing protocol

The following steps are shown in Figure 4-4:

1. The request is sent from the exploiter (Db2 IRLM) to XES.
2. The CF request is sent to both instances of the structure (in CF1 and CF2).
3. Each CF sends a Ready-To-Execute (RTE) signal to its peer CF, which indicates that it is ready to start processing the request. When the CF sends its RTE signal and received the corresponding signal from its peer CF, it can start running the request.
4. The necessary latches are acquired and the request is processed by each CF.
5. When each CF finishes processing the request, it sends a Ready-To-Complete (RTC) signal to its peer, which informs it that it completed processing and is ready to release the resources that are associated with that request. (Enabling the DUPLEXCF16 option in the COUPLExx FUNCTIONS statement or by using the **`SETXCF FUNCTIONS`** command allows the RTC exchange to occur asynchronous to the completion of the request and might help to reduce the affect of Step 5).
6. When the CF sends its RTC signal and received the corresponding RTC from its peer, it sends its response back to XES.
7. XES returns to the exploiter to indicate that the request completed.

Although synchronous system-managed duplexing provides a robust recovery mechanism for the CF lock structures by enabling failover to another CF in case of connectivity or other CF failures, it should be clear from Figure 4-4 on page 138 that every CF update request requires CF-to-CF communication and synchronization signals to coordinate execution.

Commands run in lock-step between the two CF structure instances (primary and secondary). There is a significant performance overhead, even with no distance between the CFs, which can result in increased utilization rates of the LPAR CPs, CF CPs, and the CF links or subchannels). This overhead increases as the distance between the primary and the secondary CF increases.

Because of these overheads, many customers choose not to use lock structure duplexing, even when that choice results in bearing the risks of exposing themselves to the single-point-of-failure of having a simplex lock structure.

4.3.2 Asynchronous duplexing protocol

With Db2 12, a new feature, *asynchronous duplexing for lock structures*, is introduced to address the performance issues that are associated with synchronous system-managed duplexing of the lock structures while maintaining the same level of availability capability of duplexing lock structures.

How a lock request is run by using the new asynchronous duplexing protocol is shown in Figure 4-5.

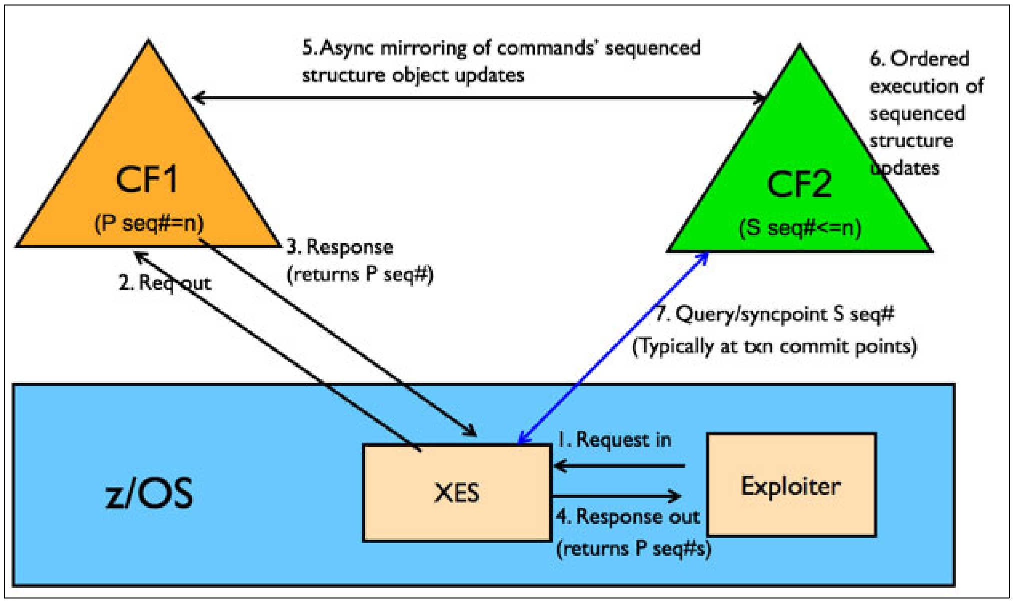

5.Async mirroring of commands' sequenced structure object updates
CF1
(P seq#=n)
CF2
(S seq#<=n)
6. Ordered execution of sequenced structure updates
2. Req out
3. Response (returns P seq#)
7. Query/syncpoint S seq# (Typically at txn commit points)
z/OS
XES
1. Request in
4. Response out (returns P seq#s)
Exploiter

Figure 4-5 Asynchronous duplexing protocol for Db2 lock structures

In this model, the secondary structure updates are performed asynchronously regarding primary structure updates. The following processing steps are performed:

1. The request is sent from the exploiter (Db2-IRLM) to XES.
2. The CF request is sent only to the primary structure (in CF1).
3. After the request is processed by the primary CF (CF1), a reply is returned to XES (and the exploiter). In addition to the usual return code (and other information), a new lock request sequence number is returned.

4. This information is returned to IRLM and Db2 and stored there and is used later during the process. The application now can continue processing; therefore, this processing is similar to the use of a lock structure in simplex mode.
5. The primary CF (CF1) sends the lock requests to the secondary CF (CF2) asynchronously and information is exchanged so that the primary is aware of the arrival and storing of the requests at the secondary structure.
6. The secondary CF received the requests and can start processing them. As the requests all have a sequence number, the secondary CF can process the lock requests in the same order as the primary.
7. To ensure integrity, at log force write time (typically at commit) Db2 must ensure that all of the lock requests from the unit of work were processed by the secondary CF as well so both contain the same information regarding this unit of recovery(UR).
8. To ensure that this process occurred, Db2 (IRLM) checks with XES to see whether the highest sequence number from the UR that is committing was processed by the secondary CF by using a new IXLADUPX request (as indicated by a blue arrow in Figure 4-5).

   If the highest lock sequence number that is requested by our UR is less than or equal to the highest lock request number that is processed by the secondary CF then, the request is completed and Db2 can continue with the commit processing. If not, the request is suspended in XES and is resumed after the highest lock request that is processed by the secondary CF catches up with the highest lock sequence number that is used by our committing UR.
9. As transactions typically do other work besides locking, it is expected that many times these IXLADUPX requests (as indicated by a blue arrow in Figure 4-5 on page 139) can complete immediately without being suspended.

This enhancement is intended to make multi-site sysplex operations and duplexing of CF lock structures at longer GDPS distances more practical by reducing the need for speed-of-light communication delays during the processing of every duplexed update operation. In case of Db2/IRLM, only at log sync times (such as at COMMIT) is the secondary structure queried to ensure that all of the CF updates were written to both primary and secondary lock structure. The expectation is that, in general, the ratio of lock structure updates to transactional sync-ups can be much higher than 1:1, and this ratio gives us a performance advantage over synchronous system-managed duplexing.

Note: Asynchronous duplexing is a duplexing protocol enhancement and it does not affect the cost of access to a remote primary (or simplex) lock structure. That cost, along with the costs of handling any sync/async conversion that occurs because of distance to the lock structure, is not reduced by this duplexing protocol enhancement.

The use of this feature employs the z/OS asynchronous duplexing support for lock structures in the CF and includes the following prerequisites:

- CF Level 21 with service level 02.16 or above
- z/OS V2.2 SPE with the PTF for APAR OA47796
- Db2 12 with the PTF for APAR PI66689
- IRLM 2.3 with the PTF for APAR PI68378

With all the prerequisites in place, asynchronous duplexing is supported starting at function level V12R1M100. However, all members must be Db2 12. That is, coexistence mode is not supported. If a Db2 11 member joins the group, asynchronous duplexing is ended and the structure is rebuilt in simplex mode.

Note: Asynchronous lock structure duplexing is only available for the Db2 lock structure, not the SCA.

4.3.3 Performance measurements

A series of measurements were conducted to evaluate the performance effect of the use of asynchronous lock structure duplexing. The Classic IRWW workload was used in a two-way data-sharing setup. The setup for the tests used CFs that were created in the same physical z13 box that also housed the LPARs for the Db2 subsystems. That is, the tests did not involve physical or simulated distance between the CFs and the LPARs. For the tests, the Db2 group buffer pools were duplexed, and SCA was not duplexed.

The following sets of measurements were performed:

- A workload was run by using a simplex setup for the lock structure.
- A workload was run by using synchronous system-managed duplexing for the lock structure.
- A workload was run by using the new asynchronous duplexing for the lock structure.

The key data points that were generated from these measurements are listed in Table 4-2.

Table 4-2 Asynchronous lock structure duplexing using IRWW workload

	RAW DATA				COMPARISONS		
(times in microseconds)	**Simplex (A)**	**Sync duplex (B)**	**Async duplex (C)**		**Sync duplex vs. simplex (B-A)/A**	**Async duplex vs. simplex (C-A)/A**	**Async duplex vs. sync duplex (C-B)/B**
Db2 Member 1							
LPAR CPU utilization	40.37%	56.78%	44.21%		16.41%	3.84%	-12.57%
ETR (txns/sec)	1,451	1,453	1,470		0.11%	1.30%	1.19%
ITR (txns/sec)	3,594	2,558	3,324		-28.82%	-7.50%	29.96%
Class 2 CPU/commit	533	958	600		79.74%	12.57%	-37.37%
MSTR CPU/commit	55	54	73		-1.82%	32.73%	35.19%
DBM1 CPU/commit	52	58	48		11.54%	-7.69%	-17.24%
IRLM CPU/commit	1	5	2		400.00%	100.00%	-60.00%
TOTAL Db2 CPU/commit	641	1075	723		67.71%	12.79%	-32.74%
Db2 Member 2							

LPAR CPU Utilization	40.37%	55.16%	43.62%		14.79%	3.25%	-11.54%
ETR (txns/sec)	1,455	1,446	1,473		-0.63%	1.24%	1.88%
ITR (txns/sec)	3,603	2,621	3,376		-27.27%	-6.30%	28.84%
Class 2 CPU/commit	527	941	590		78.56%	11.95%	-37.30%
MSTR CPU/commit	55	55	72		0.00%	30.91%	30.91%
DBM1 CPU/commit	42	44	49		4.76%	16.67%	11.36%
IRLM CPU/commit	1	5	2		400.00%	100.00%	-60.00%
TOTAL Db2 CPU/commit	625	1045	713		67.20%	14.08%	-31.77%
Coupling Facility A: Primary lock structure							
CF Utilization - ICF3A	11.30%	25.10%	24.10%		13.80%	12.80%	-1.00%
Coupling Facility B: Secondary lock structure							
CF Utilization - ICF3B	9.70%	31.50%	23.40%		21.80%	13.70%	-8.10%

The data that is shown in Table 4-2 on page 141 shows that for the classic IRWW workload, the following observations were made:

- Over a 30% reduction in Db2 CPU time per transaction can be observed when asynchronous duplexing was used, compared to the CPU usage when using the synchronous system-managed duplexing mode for the Db2 lock structure.
- Approximately a 13% increase in Db2 CPU time per transaction was measured when asynchronous duplexing was used, compared to the CPU usage when simplex mode is used for the Db2 lock structure.
- The evaluations that are described in this section were done at the IBM Silicon Valley Lab. For those measurements, no distance between the two coupling facilities existed. At the time of this writing, the IBM Systems team is planning to document several performance evaluations they conducted in which a distance between the coupling facilities does exist.

4.3.4 Conclusion

Based on the measurements in Table 4-2 on page 141, customers who use synchronous duplexing for Db2 lock structures see a significant reduction in Db2 CPU usage after switching to use asynchronous duplexing.

Customers who do not use Db2 lock structure duplexing see an increase in CPU usage if they start using asynchronous duplexing. They also should consider whether the increased cost of the extra CPU overhead outweighs the benefits of achieving increased availability by using asynchronous lock structure.

4.4 Automatic retry of GRECP and LPL recovery

Before we describe the changes in Db2 12 that are related to retrying GRECP and LPL recovery, we review the changes in this area over the last couple of Db2 versions.

4.4.1 GRECP recovery

The group buffer pool RECOVER-pending (GRECP) status is set for GBP-dependent objects when Db2 detects there is something wrong with one or more group buffer pool structures in the CF with pages that were not externalized.

When a structure failure or 100% loss of connectivity is detected, Db2 performs damage assessment (DA) to mark the GBP-dependent page sets in the affected group buffer pool or pools as GRECP. When a Db2 member is also involved in such a failure, all GBP-dependent objects are put into GRECP during group restart.

To recover from a GRECP condition, you must issue a `-STA DB(xx) SP(yy)` command. GRECP marked objects are then recovered from information in the logs. Before Db2 V5, you manually issued the `-START DB` command for the page sets in GRECP status.

Db2 V5 automatic GBP recovery

In Db2 V5, automatic recovery of GRECP objects was introduced. This feature was for a failure of the GBP and for total loss of connectivity by all members, but the Db2 systems are still available.

If AUTOREC YES is set for a group buffer pool, Db2 performs an automatic recovery. If AUTOREC NO is set or Db2 failed, the `-START DATABASE` commands still must be issued.

Db2 V9 automatic GBP recovery during restart

Although automatic GRECP is a useful feature, the Db2 V5 enhancement did not help when some or all of the Db2 members failed alongside the GBPs. This issue effectively always is the case in many disaster recovery scenarios. The Db2s then must be restarted (not taken down cleanly) and also missing their group buffer pools.

To help with this issue, Db2 V9 introduced auto GRECP recovery on Db2 restart (at the end of restart) when a GBP failure occurs. This functionality applies to a group restart and a normal restart.

Each Db2 starts only GRECP recovery for group buffer pool-dependent objects that it had an update interest in at the time this Db2 came down (R/W state). AUTOREC is used to control this functionality and must be set to YES to realize automatic GRECP recovery on restart. When Db2 starts automatic GRECP recovery after restart, it attempts to acquire a conditional drain for each GRECP object. If Db2 is successful in acquiring the drain, it performs the recovery from the log.

All the drained objects are recovered by using one **-START DB** command. The reason for this command is twofold. First, this command results in only one scan of the log. Second, and more importantly, the objects do not need to be sequenced (that is, the catalog and directory objects do not need to be done first). Fast log apply is always used, as fast log is part of restart.

If the drain of an object is not successful or GRECP recovery fails, a DSNI005I message is issued and the object remains in GRECP. For those objects where the drain fails, you must issue a manual **-START DB** command to resolve the GRECP condition.

The following exceptions apply when automatic GRECP recovery is not used:

- Db2 is restarted with the DEFER ALL option.
- Db2 is restarted in system level PITR mode.
- Db2 is started in tracker mode.
- Db2 is restarted in restart LIGHT mode.

4.4.2 LPL recovery

A page that is in error can be logically or physically in error. A page is considered logically in error if Db2 expects its problem can be fixed without redefining new disk tracks or volumes for the object. For example, if Db2 cannot write pages to disk because of a (temporary) connectivity problem, the pages are logically in error. To track this issue, Db2 inserts entries for those pages that are logically in error into a logical page list (LPL). The object's pages are put into LPL for data integrity protection.

Entries that are added to the LPL results in the following common situations:

- Transient DASD read/write errors that can be fixed without redefining the DASD
- Must complete operation could not access the GBP/DASD for read and write:
  - Apply UNDO/REDO log records during the BLR and FLR restart phase
  - Apply UNDO log records during rollback
  - Force-at-commit write failure to GBP
- During must complete activities when Db2 cannot obtain the required page P-lock or page latch for a page or set of pages
- Pages that the Db2 castout process cannot successfully cast out
- Pages from the restart DEFER objects list during the restart

Pages in LPL are unavailable for any access until the LPL condition is removed. To resolve the LPL condition for an object, you issue the **-STA DB(xx) SP(yy)** command for every object with an LPL exception condition. Db2 then reads the Db2 log and applies any changes to the page set.

The RECOVER, REBUILD, and LOAD utilities can also be used to recover LPL pages. If the **-START DATABASE** command fails to successfully recover the LPL pages, you are forced to recover the whole page set by using the RECOVER/REBUILD utility, or reloading the data.

Db2 V8 automatic LPL recovery

The manual process of resolving LPL conditions can be time-consuming if there are many objects with pages in the LPL. This process also can extend the time that some applications are unavailable. This manual process can also be error prone because some of the objects can be missed when the `-START DB` commands are issued, especially when there is a long list of objects to be recovered.

Starting in Version 8, Db2 automatically attempts to recover pages that are added to the LPL when they are added to LPL. When pages are added into LPL, Db2 issues message DSNB250E to indicate the LPL page range and the names of the database, the page set or partition, and the reason for adding the page to the LPL.

If the automatic LPL recovery runs successfully, LPL pages are deleted from the LPL and Db2 issues the message DSNI021I to indicate the successful completion of the recovery. If the automatic LPL recovery does not run successfully, pages are kept in the LPL and Db2 issues the DSNI005I to indicate the failure of the automatic LPL recovery. In this case, manually recovering the pages in LPL is required by using a `-START DB(xx) SP(yy)` command or by running the RECOVER/REBUILD utility or LOAD utility with REPLACE option.

Automatic LPL recovery is not started by Db2 in the following situations:

- DASD I/O error
- During Db2 restart or end restart time
- GBP structure failure
- GBP 100% loss of connectivity.

Db2 11 automatic LPL recovery at end of restart

Db2 11 improved the LPL recovery process by starting automatic LPL recovery of objects at the end of normal restart and restart light.

If the restart involves any indoubt or postponed abort (PA) units of recovery (URs), the LPL recovery that is associated with those URs is not automatically triggered at the end of restart. These objects cannot be automatically recovered because Db2 does not know the entire LPL log range for indoubt and PA URs until they are resolved.

The auto-LPL recovery process is not triggered for any of the following circumstances:

- If the Db2 member is started in access maintenance mode
- If the Db2 member is started in point-in-time (PIT) Recovery mode
- If the Db2 member is started at a tracker site
- If the Db2 member is involved in any type of conditional restart
- If DEFER ALL was specified on installation panel DSNTIPS
- For objects that are explicitly listed in a DEFER object list on installation panel DSNTIPS
- For table spaces that are defined as NOT LOGGED

The processing for automatic LPL recovery is similar to the auto-GRECP recovery processing that is done at the end of restart in that the auto-LPL recovery uses the messages to report LPL recovery progress, errors, and successful completion.

Db2 makes only one attempt to automatically recover LPL objects. If the auto-LPL recovery fails at restart time, the DBA can manually recover the LPL objects by issuing `-START DB` commands or regular auto-LPL recovery can perform the recovery at the appropriate time.

4.5 Retrying GRECP and LPL recover in Db2 12

Although Db2 came a long way to automate GRECP and LPL recovery, Db2 typically tries to perform auto recovery only once. Often, automatic recovery fails because of some transient condition; for example, not all objects in GRECP are auto recovered because of contention with other members during the group restart.

What typically happens is that after seeing the DSNI049I (GRECP/LPL recovery completed) message, a `-DISPLAY DB` command is issued to list all the objects that are still in GRECP/LPL, and then manual `-START DB` commands are issued for those objects. In most cases, the transient condition, such as contention, is resolved by this time and the command now succeeds in removing the GRECP/LPL condition.

Would it be useful if Db2 performs this retry if it typically has a good chance of being successful? Db2 12 can perform that retry process because it automatically retries GRECP/LPL recovery for you.

Although Db2 12 provides the capability to retry the GRECP/LPL recovery after it failed before, it does not retry indefinitely. In some cases, things can be broken, and there is no point to retry as the `-STA DB2` command continues to fail and the problem must be analyzed and addressed.

Db2 retries to automatically resolve GRECP/LPL conditions three times (from each member) before giving up. In Db2 12, Db2 checks (the DBET) and generates an (internal) report of all the page set or partitions in GRECP/LPL status every minute.

If one object's failed recovery was retried or it was retried more than 3 minutes ago, Db2 starts the GRECP/LPL recovery process for the object. The retry counts are monitored and each object has up to three retry opportunities. If the third retry still fails to recover a GRECP/LPL contention, Db2 gives up and the GRECP/LPL condition must be resolved manually by issuing the `-START DB` command or some other means.

The retry LPL/GRECP recovery tasks are scheduled as system agents in the DBM1 address space and are zIIP eligible. Retry tasks are scheduled for up to 100 objects at each one minute interval. If Db2 detects that there are already 500 LPL/GRECP recovery tasks running, it stops scheduling new tasks. The automatic GRECP/LPL recovery retry tasks use a correlation ID of DRSTRT02.

The same messages (DSNB320I, DSNB321I, DSNB322I, and DSNB323I) are used to report GRECP status, and message DSNB250E is used to report LPL status. The same DSNI049I message is used to indicate that the retry task completed.

Also, the same data manager messages are used to report the start and completion of the automatic GRECP and LPL recovery retries for individual objects (DSNI006I for the start of GRECP/LPL recovery and DSNI021I for the successful recovery). As before, check for message DSNI005I to find the reason why the GRECP/LPL recovery retry failed.

When automatic GRECP/LPL recovery is triggered at the end of restart, it recovers all GRECP/LPL object by using a single command. The Db2 12 automatic retry feature schedules recovery in the following order:

- Schedule SYSLGRNX and its indexes (DSNLLX01, DSNLLX02)
- Schedule all other DIRECTORY GRECP objects
- Schedule all CATALOG GRECP objects
- Schedule all users GRECP objects

Db2 does not schedule retry tasks when Db2 is one in the following conditions:

- Is started for Point In Time Recovery (PITR)
- Is started as Tracker Site
- Is using the DEFER ALL option
- Is started in ACCESS(MAINT) mode

For INDOUBT or POSTPHONED ABORT (PA) URs, LPL recovery can be retried after the INDOUBT or PA condition is resolved.

As this enhancement is mainly a usability and availability enhancement, no specific performance measurement were conducted to evaluate this enhancement. However, performance is expected to be similar to automatic GRECP/LPL recovery.

This functionality is available in Db2 12 at function level V12R1M100.

4.6 Db2 shutdown and restart performance improvement when many read/write objects are used

Before Db2 12, the Db2 shutdown process first takes a system checkpoint following a **`-STO DB2 MODE(QUIESCE)`** or **`MODE(FORCE)`** command, where the Db2 Buffer Manager (BM) component writes a checkpoint Page Set Control Record (PSCR) for every page set that is open-for-update to the log. Next, the BM shutdown code closes all page sets, and writes a close PSCR log records for most (if not all) the page sets that were checkpointed.

During the subsequent restart, Db2 reaches the last system checkpoint's log records first. It sees all the checkpoint PSCR log records, and allocates control blocks for each of the page sets. Then, it discards them when Db2 proceeds with restart and sees the close PSCR log records.

Starting in Db2 12, shutdown drives the data set close process first, and performs the system checkpoint after. This way, during the subsequent restart, Db2 does not see any checkpoint PSCRs for successfully closed page sets and the restart process must allocate only control blocks for objects with indoubt or postponed Units of Recovery (URs) when the Db2 was last stopped. This change results in faster Db2 shutdown and restart time and less storage is required during restart.

In Db2 12, the shutdown process performs the same tasks as in previous Db2 releases, only in a different order. But, the seemingly simple change can make subsequent Db2 restart complete in less time and use less storage.

This enhancement applies to data sharing and non-data sharing environments.

4.6.1 Performance measurements

A set of OLTP workload measurements were performed to analyze the performance effect of this enhancement on Db2 shutdown and restart elapsed time. There were 7800+ read/write objects closed during Db2 shutdown.

Db2 restart performance

With this enhancement, Db2 restart time is 5.58 seconds, compared to 6.77 seconds without the enhancement, which is 1.19 seconds faster.

A closer examination reveals that nearly all of the time savings are achieved during the status rebuild (CSR) phase. More specifically, the CSR phase took 3.25 seconds without the enhancement, while only 2.23 seconds were needed with the enhancement. This saving of 1.02 seconds out of 3.25, or more than 30% is impressive. The detailed breakdown of the Db2 restart measurement is listed in Table 4-3.

Table 4-3 Db2 restart time measurement

Event (Times in seconds)	**Without the enhancement**		**With the enhancement**		**Delta (with/without)**	
	Time between phases	**Cumulative time**	**Time between phases**	**Cumulative time**	**Time between phases**	**Cumulative time**
-CEB2 STA Db2 (begin restart)	0.00	0.00	0.00	0.00		
DSNJ099I: End of Log Initialization phase	3.18	3.18	3.02	3.02	-0.16	-0.16
DSNR004I: End of Current Status Rebuild phase	3.25	6.43	2.23	5.25	-1.02	-1.18
DSNR005I: End of Forward Log Recovery phase	0.01	6.44	0.00	5.25	-0.01	-1.19
DSNR006I: End of Backward Log Recovery phase	0.00	6.44	0.00	5.25		-1.19
DSNR002I: Restart Completed	0.33	6.77	0.33	5.58		-1.19

Db2 shut down performance

In addition to the restart time savings, a shortened Db2 shutdown time was observed during the workload test with this enhancement enabled. Most savings result from faster DBM1 shutdown, as is to be expected because DBM1 handles the Db2 page sets.

A reduction in the Db2 shutdown time of nearly 5 seconds out of 37 seconds was measured. Without the enhancement, Db2 shutdown had to write 7800+ checkpoint PSCR log records, followed by 7800+ page set close PSCR log records. With the enhancement, Db2 shutdown only had to write 7800+ close PSCR log records and no checkpoint PSCR log records. The reduced amount of logging makes a difference here. A breakdown of the Db2 shutdown time is listed in Table 4-4.

Table 4-4 Db2 shut down measurement

Event (Times in seconds)	**Without the enhancement**		**With the enhancement**		**Delta (with/without)**	
	Time between phases	**Cumulative time**	**Time between phases**	**Cumulative time**	**Time between phases**	**Cumulative time**
-CEB2 STO Db2 -CEB2 Subsystem Stopping	0.00	0.00	0.00	0.00		
DSNL005I: DDF is stopping	0.00	0.00	0.00	0.00		
DSNL006I: DDF stop complete	1.02	1.02	1.02	1.02		

DSNX922I: Begin Disconnect of SP Address Spaces from Db2	0.00	1.02	0.00	1.02		
DSNX923I: All SP Address Spaces are now disconnected	5.00	6.02	5.00	6.02		
DSNY025I: DBM1 shutdown is complete	31.70	37.72	26.87	32.89	-4.83	-4.83
DSNY025I: DIST shutdown is complete	0.01	37.73	0.01	32.90		-4.83
DSNY025I: MSTR shutdown is complete	0.87	38.60	1.46	34.36	+0.59	-4.24
DSN9022I: DSNYASCP 'STOP Db2' normal completion	0.00	38.60	0.01	34.37	+0.01	-4.23
DXR110I: IRB2002 stop command accepted (IRLM)	0.49	39.09	0.99	35.36	+0.50	-3.73
DXR121I: IRB2002 End-of-Task Cleanup Successful (IRLM)	0.36	39.45	0.45	35.81	+0.09	-3.64
DSN3100I: Subsystem CEB2 Ready for start command	1.69	41.14	1.57	37.38	-0.12	-3.76

4.6.2 Usage considerations

There are no external parameters that are associated with this enhancement. The enhancement is available in Db2 12 at function level V12R1M100.

4.7 Db2 peer recovery

Db2 12 peer recovery is a new feature that responds to customer requirements. It was developed for GDPS automation to perform Db2 peer restart to automate retained lock recovery of failed Db2 members after LPAR failures without requiring Automatic Recovery Management (ARM) or other external automations.

Peer recovery refers to a Db2 data sharing member's ability to automatically recover retained locks on behalf of a failed member. The recovery process is implemented by initiating the restart light of the failed member. That is, if a member fails, a surviving member with the proper DSNZPARM setting initiates a restart light of the failed member.

Db2 already performs peer status rebuild (to rebuild the SCA and ensure that all page set and partition P-locks are reacquired) and peer forward recovery (to ensure that all of the locks are reacquired and the lock structure is rebuilt) today during group restart if not all members are being restarted. However, those restart phases do not perform any processing to resolve retained locks of the member (or members) not being restarted. It is this stage that the new peer recovery comes in.

A new DSNZPARM, called PEER_RECOVERY, allows you to specify whether this data sharing member is to participate in data sharing peer recovery and what role the member has in the peer recovery process. The following acceptable values for PEER_RECOVERY are available:

- NONE

  This member does not participate in peer recovery. Use this option if z/OS ARM is configured to restart failed Db2 member. This value is the default value.

- RECOVER

  This member should be recovered by a peer member if it fails.

- ASSIST

  This member should attempt to start peer recovery for other failed members. When this member detects a failure, it attempts to start a LIGHT(YES) restart for the failed member (if it is not started) to recover the retained locks.

- BOTH

  Both RECOVER and ASSIST options are activated for this member.

Peer recovery is accomplished by one of the assisting members starting a restart light for the failed member. The LIGHT(YES) option is used, along with the ZPARM that was used by the failed member. The first peer assist member to obtain the lock for the failed member then attempts the restart light.

4.7.1 Performance measurements

A TPC-E two-way data sharing workload was used to evaluate the Db2 peer restart feature, focusing on the following metrics:

- The elapsed time for (peer) restart light to resolve retained locks.
- The CPU and storage usage during restart light.

The two-way data sharing members, CEB1 and CEB2, had their DSNZPARM PEER_RECOVERY set to BOTH in this test.

The test scenario is such that initially the Db2 12 data sharing members CEB1 and CEB2 are steadily running an OLTP workload on systems STLAB10 and STLAB11. Suddenly, STLAB10 fails (and it is confirmed at 11:20:27.28 by operator) and 0.82 seconds later, Db2 member CEB1 was brought up using LIGHT(YES) by CEB2. The messages concerning CEB1's restart light are listed in Table 4-5.

Table 4-5 Db2 peer recovery: Timeline and messages issued

Timestamp	Message number	Command or message text	Comments
11:20:28.10		-CEB1 START DB2,PARM(ZPCEB1), LIGHT(YES)	Restart begins
11:20:37.24	DSNJ099I	Log recording to commence with StartRBA=000000000004D2131000	End of 'log initialization' phase
11:20:39.74	DSNR004I	Restart…UR Status Counts In Commit=0, Indoubt=0, Inflight=65, In Abort=0, Postponed Abort=0	
11:20:39.74	DSNR007I	Restart…Status Table	End of 'current status rebuild' phase

11:21:26.87	DSNR005I	Restart…Counts After Forward Recovery In Commit=0, Indoubt=0	End of 'forward log recovery' phase
11:21:27.20	DSNR006I	Restart…Counts After Backward Recovery Inflight=0, In Abort=0, Postponed Abort=0	End of 'backward log recovery' phase
11:21:42.81	DSNR002I	Restart completed	
11:21:42.83	DSN9022I	DSNYASCP 'STA DB2' Normal Completion	Restart completed
11:22:00.88	DSN3100I	Subsystem CEB1 Ready for Start Command	

Using Db2 12 peer recovery, it took only 0.82 seconds for the surviving Db2 member CEB2 to attempt restarting the failed Db2 member CEB1 with LIGHT(YES) on the remaining LPAR STLAB11 from the point of the operator confirmation of STLAB10’s failure.

After approximately another 75 seconds, all of the 65 inflight transactions on CEB1 were resolved and all the retained locks were handled and freed. After the successful recovery, CEB1 then shuts down automatically, as is always the case when restart light is used.

Before STLAB10’s unplanned outage at 11:19, both LPARs were running at a CPU utilization rate of about 70%. CPU utilization for STLAB11 reached nearly 100% while restarting the CEB1 member was recovering its retained locks.

After CEB1 ended, CPU utilization remained high for STLAB11 because workload balancing (WLB) moved all of CEB1’s user threads to CEB2, and STLAB11 had to handle all the workload that is normally intended for two LPARs.

After LPAR STLAB10 comes up again and CEB1 is restarted normally on STLAB10, WLB rebalances the user threads among CEB1 and CEB2, and STLAB11’s intense workload is relieved in turn.

This CPU utilization rate pattern throughout this scenario is shown in Figure 4-6.

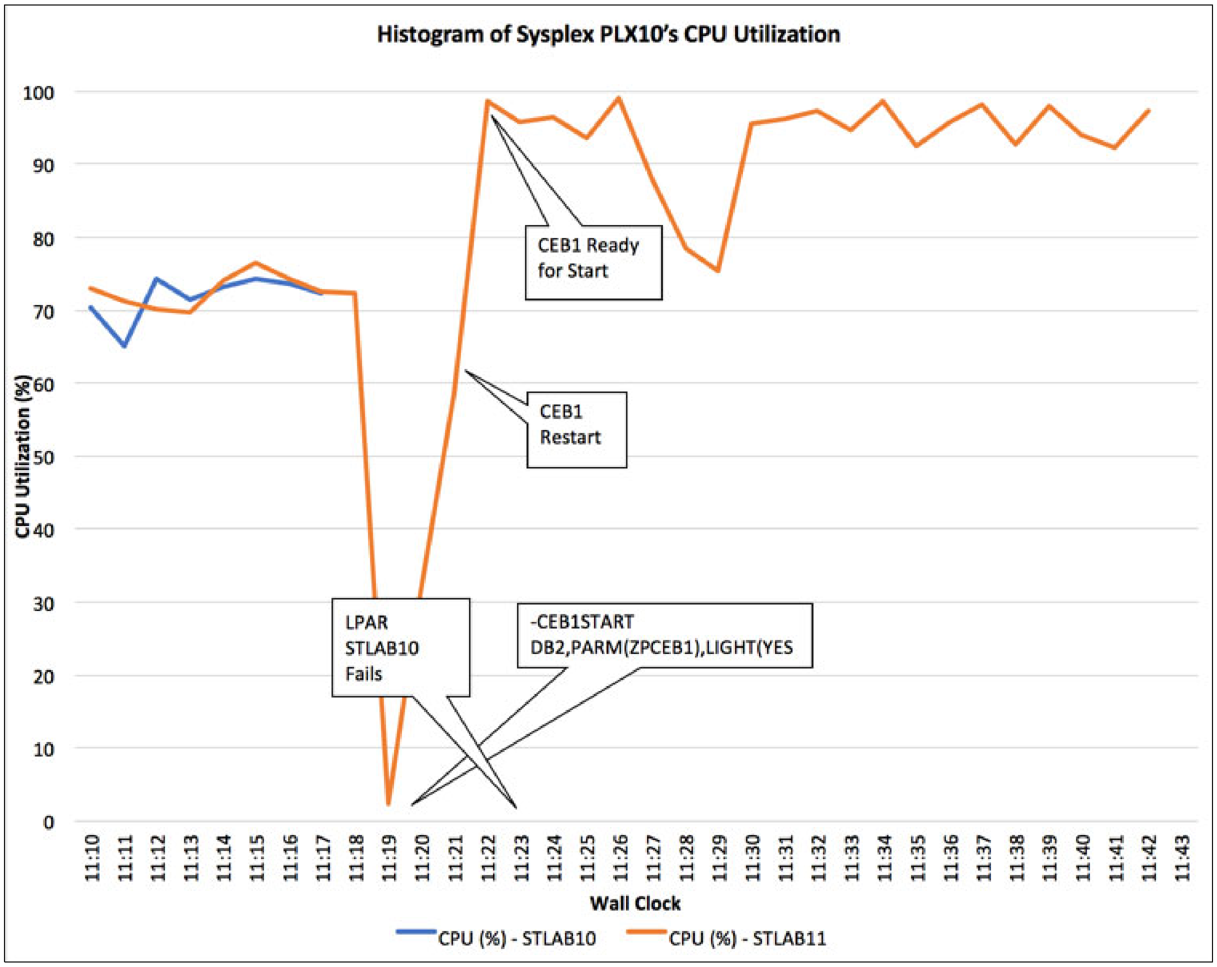


Figure 4-6 Histogram of sysplex PLX10's CPU Utilization

Before STLAB10's unplanned failure, CEB1 and CEB2 together were processing approximately 5,300 transactions per second. Transaction processing came almost to a halt, with only a few transactions running on CEB2 until CEB1 successfully resolved the retained locks.

After workload balancing shifted all user threads to CEB2, the ETR of CEB2 rose to mid-3000s. At this point, transactions that were running were experiencing unusually high contention and were taking longer to complete.

The evolution of the transaction rate that was observed during the test is shown in Figure 4-7.

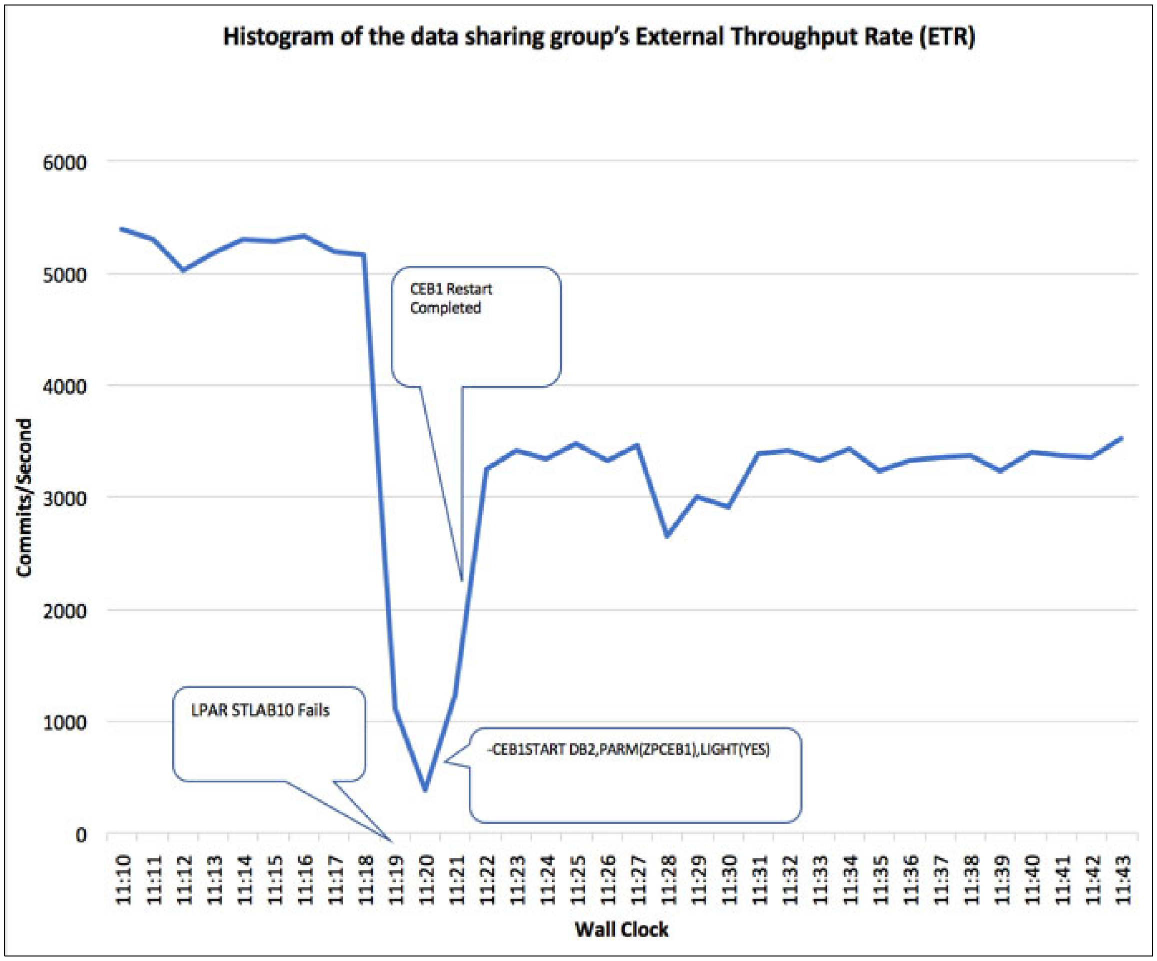


Figure 4-7   Histogram of the data sharing group's ETR

Figure 4-8 on page 154 shows STLAB11's storage usage. Approximately 85,000 MB was used by the LPAR when the normal workload was running. During the 2-minute interval in which CEB1's restart LIGHT(YES) is executing, STLAB11's storage footprint went up to 86,500 MB. The additional 1,500 MB was used by CEB1 to resolve incomplete Units of Recovery (URs) and recover retained locks.

After restart light completed and CEB1 ended, storage usage remained higher than normal because CEB2 had more thread storage usage after WLB moved all CEB1's user threads to CEB2. It is expected that STLAB11's real storage usage returns to a normal level after CEB1 reopens on STLAB10.

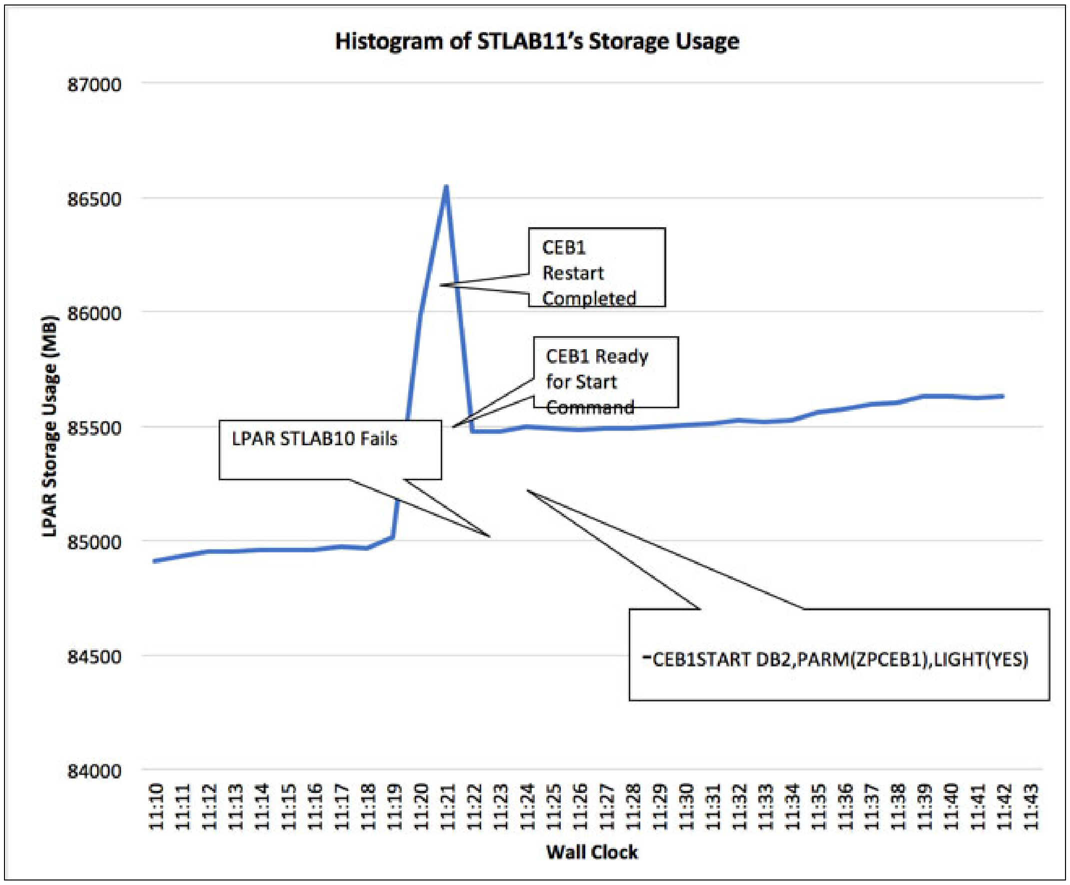


Figure 4-8 Histogram of STLAB11's storage usage

4.7.2 Usage considerations

Db2 12 peer recovery does not change the way restart LIGHT(YES) works. The same recovery process occurs. The advantage of using the new peer recovery feature is that the activity of recovering retained locks and in-flight URs can be controlled by Db2 without involving any manual operator interventions or the need to set up external tools (such as ARM) to trigger the process.

However, if you are using automated system restart or ARM, there is no other benefit of using the Db2 peer recovery feature.

When using other types of automation, you have more control over where Db2 is restarted and which restart LIGHT option to use. Db2 peer restart always uses LIGHT(YES).

Peer recovery applies only to Db2 for z/OS. When a system fails, other components, such as IMS, CICS, and WebSphere, also most likely must be restarted, if only to resolve indoubt units of work. Db2 peer recovery does not help with restarting those components.

Customers must evaluate their own environments to decide whether to use Db2 peer recovery.

5

Workload level measurements

In this chapter, we describe the results of some key performance measurements by using different IBM internal workloads. They are used to measure overall performance enhancements by using a combination of features from the latest Db2 for z/OS release. For more information about the type of work that was performed by each of the workloads, see Appendix A, “IBM Db2 workloads” on page 375.

In addition to the usual benchmarks, a special set of tests were conducted to evaluate how hard we might drive the new fast unclustered insert algorithm. These measurements are also described in this chapter.

The following workloads were used to evaluate the overall and some specific Db2 12 performance enhancements:

- High volume INSERT benchmark
- Classic IRWW workload measurements
- Distributed IRWW workload performance measurements
- IBM Brokerage transaction workload
- RTW workload
- High insert batch workload measurements

This chapter includes the following topics:

5.1 High volume INSERT benchmark

Db2 12 introduces many performance-related enhancements, such as Insert Algorithm 2, the support for active log data sets that are larger than 4 GB, and a reduction in latch class 19 and 23 and page latch contention. With these enhancements, Db2's ability to process many inserts in a short time is better than ever. To verify this ability, we assembled a team to determine whether inserting 10 million rows per second can be accomplished in a Db2 for z/OS environment.

At the end of the project, a maximum insert rate of 11.7 million inserts per second was achieved for a table with no index defined. A rate of 5.3 million inserts per second was achieved for a table with a single index defined. These results are described in more detail in the following sections.

5.1.1 Workload description

The high volume of insert operations that are performed during this benchmark was driven from an SQLJ (static Java) application that is running on Linux for System z. HiperSockets communication was used between the Linux for System z and z/OS images to reduce network latency. Sysplex workload balancing was enabled so that the inserts were distributed evenly across all members of the Db2 data sharing group based on resource availability.

Table and index description

The table that was used for this benchmark consists of 10 columns, including BIGINT, INTEGER, CHAR, VARCHAR, TIMESTAMP, and DECIMAL columns. A table space with a total of 300 partitions (range-partitioned with absolute page numbering) was created by using one of the BIGINT columns as the partitioning column. The inserted data rows were approximately 200 bytes. COMPRESS YES was used on the table space to reduce the size of the data.

For the workload run where the table had a (non-unique) index that is defined, the specified columns for the index were BIGINT and INTEGER columns. A 32 K buffer pool was used for the index to reduce the amount of index page split activity.

SQLJ driver application program description

A Java application (SQLJ) that uses static SQL to access the target table was developed to generate the insert statements. To reduce the TCP/IP send and receive network communication overhead, the inserts were batched into groups of 200 inserts (which resulted in one multi-row insert with a rowset size of 200) per commit.

Sysplex workload balancing was enabled for the application to evenly distribute the inserts across all Db2 members in the data sharing group. A value of RTPIFACTOR=10 was specified in the z/OS IEAOPTxx parmlib member to reduce the influence the Performance Index (PI) value has on the weights that are assigned by WLM to each LPAR. With the default setting of 100, we observed abnormal situations where some Db2 members are dropped off the sysplex list and no longer receive any work.

The package that was used by the SQLJ application was bound with RELEASE(DEALLOCATE) to reduce CPU consumption.

5.1.2 Hardware configuration

A single z13 box was used to conduct this high-volume insert benchmark. Two internal coupling facilities (ICF) were used to house the Db2 lock structure, SCA structure, and group buffer pools (GBPs) that were needed for the 12-wa Db2 data sharing group, and the other structures necessary for the base 4-way sysplex configuration.

Db2 12 and z/OS 2.2 were used on each LPAR, while the CFs were using a CFCC level 21 code base. The CF links between the ICFs and z/OS LPARs consisted of four Internal Coupling Facility Peer links (ICPs) and four Coupling over InfiniBand links (CIBs).

Two Linux for System z LPARs in the same z13 box were used to drive the benchmark workload. Insert transactions that were generated by 1200 clients were distributed to the 12 members of the Db2 data sharing group by using sysplex workload balancing. HiperSockets (Internal Queued Direct or IQD) were used to connect the Linux for System z LPARs and z/OS LPARs.

During the benchmark, seven IBM DS8870 control units were used to store the Db2 logs and data (and index). Each of these control units supports four 16 GB channels or eight 8 GB channels. Approximately 70% of the drives are 15 K RPM drives and the rest are 10 K RPM drives.

Each Db2 active log data set was approximately 50 GB in size to minimize full active log data set switching. The active logs were dual copies on six IBM DS8870 control units and each log was striped across four volumes.

All Db2 members' LOGCOPY1 log data sets were defined on CEC0 I/O enclosures and adapters of the control units, and all LOGCOPY2 logs on CEC1. To evenly distribute the logs, each control unit supported logs from two data sharing members. The Db2 data (and index) and Db2 catalog were on a separate DS8870 controller. The hardware configuration that was used by this high volume insert benchmark is shown in Figure 5-1.

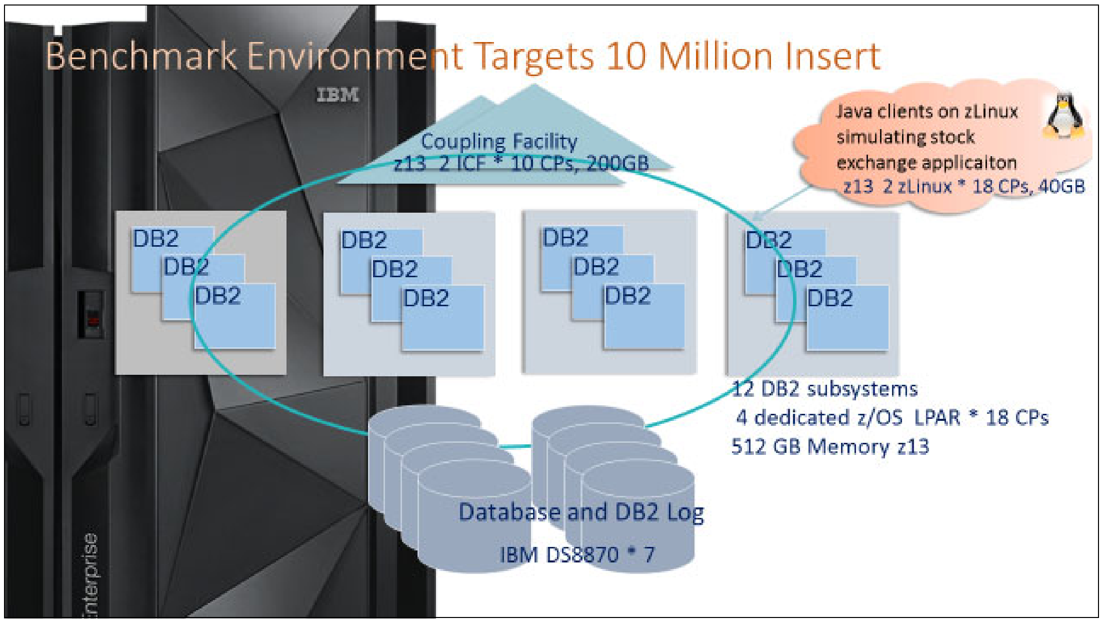

Benchmark Environment Targets 10 Million Insert
IBM
Coupling Facility
z13 2 ICF * 10 CPs, 200GB
Java clients on zLinux simulating stock exchange applicaiton
z13 2 zLinux * 18 CPs, 40GB
DB2
12 DB2 subsystems
4 dedicated z/OS LPAR * 18 CPs
512 GB Memory z13
Database and DB2 Log
IBM DS8870 * 7
Enterprise

Figure 5-1 Configuration for 10 million insert benchmark

5.1.3 Performance results

The workload was run by using 1200 threads for two different table configurations. During the first measurement, the insert target table had no index and in the second measurement, the target insert table had a defined partitioning index. Distributing the logs on six different IBM DS8870 control units and separating data from logs improves the log write rate and reduces log wait time in Db2.

An insert rate of 11.7 million per second was achieved during this measurement by using a table without an index. An insert rate of 5.3 million per second was reached when the table had a partitioning index. The benchmark results are listed in Table 5-1.

Table 5-1 Measurement results summary

	Table without index	**Table with partitioning index**
# Inserts / second	11,797,700	5,364,300
Average log write wait time (ms)	3.953	5.13
Total logging rate (MBps)	2,017	1,663
Average log write I/O size (bytes/request)	106,964	147,617

The breakdown of the number rows that were inserted per second by each Db2 member for the measurement that is run by using a table without an index that is defined is shown in Figure 5-2.

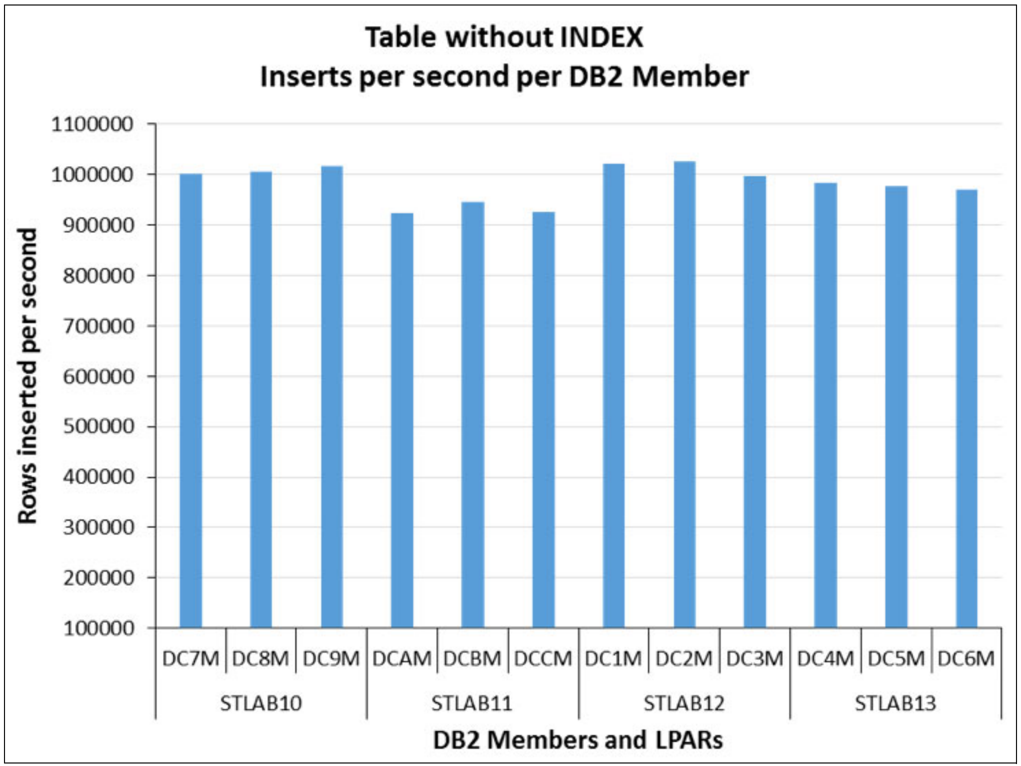


Figure 5-2 Insert rate per member that uses a table without an index

The breakdown of the number of rows that are inserted per second by each Db2 member for the measurement that is run by using a table with a partitioning index defined is shown in Figure 5-3.

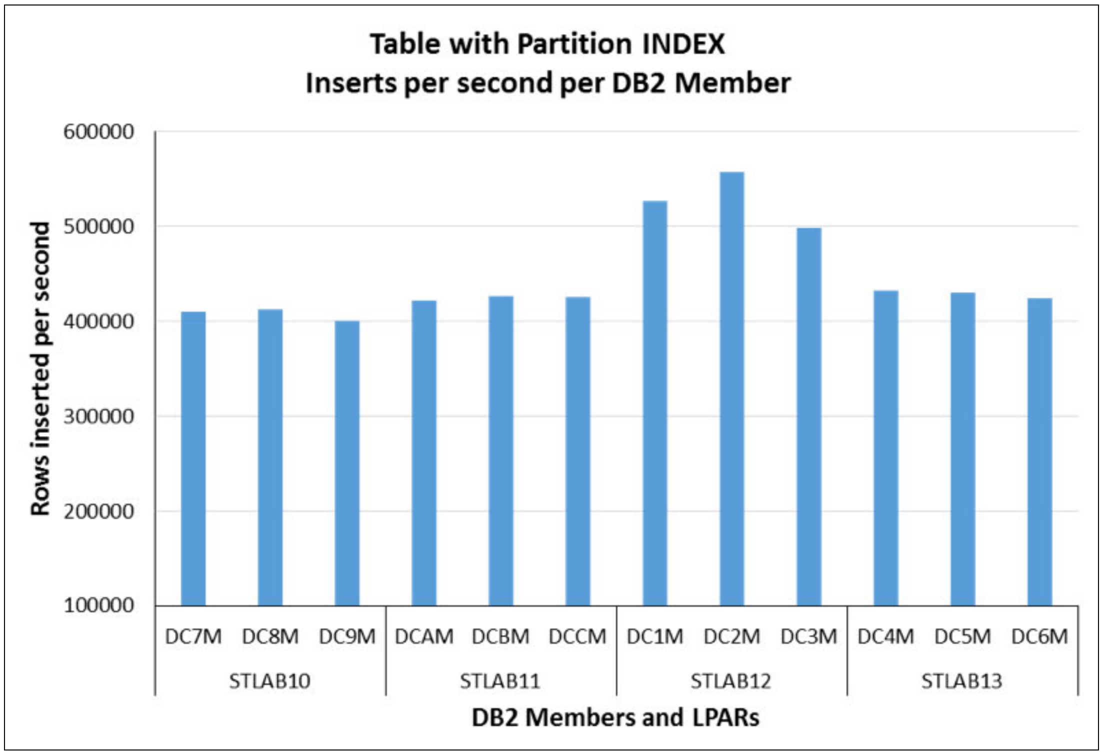


Figure 5-3 Insert rate per member that uses a table with one index

The Db2 members that are running on the STLAB12 system perform more inserts per second than the other members. This result is largely because their Db2 active logs were all on faster DASD, which reduced the elapsed time of the transaction. By removing the insert bottleneck and using insert algorithm 2, the insert rate continues to increase until you encounter the next bottleneck, such as logging I/Os, or CPU capacity. To use the full power of insert algorithm 2, it is important to ensure that other limiting factors, such as log write I/O, I/O in general, or CPU capacity are considered.

5.2 Classic IRWW workload measurements

In this section, performance evaluation results that use the Db2 Classic IRWW workload are documented for several measurements that use Db2 11 and Db2 12 in one-way and two-way data sharing environments.

The workload's transactions are generated from IMS subsystems, which are locally connected to the Db2 subsystems by using the IMS attachment facility. The measurement scenarios were conducted in a sequence that customers are most likely to follow when they migrate their Db2 subsystems.

For more information about the IRWW workload, see A.3, "IRWW distributed workload" on page 379.

5.2.1 Db2 11 to Db2 12 migration scenarios (one-way data sharing)

These Db2 measurements were conducted on a z/13 LPAR that used four dedicated general CPs that was running z/OS V2.2. IMS 11 was used as the transaction manager by using Message Processing Programs (MPPs) to drive the workload. All measurements used a workload with 66% thread reuse. Packages are bound by using the RELEASE(COMMIT) BIND option. The results of these measurements, which includes the internal throughput rate (ITR), are listed in Table 5-2.

Table 5-2 ITR and CPU time for one-way classic IRWW that uses RELEASE(COMMIT)

Classic IRWW measurement	ITR (commits/sec)	Total CPU/commit (microseconds)	Db2 12 vs. Db2 11 CPU/commit (delta %)
Baseline: Db2 11 NFM	4977	376	N/A
Db2 V12R1M100 (FTB Auto) No rebind	5047	361	-4.0
Db2 V12R1M100 (FTB Auto) After rebind	5048	354	-5.9
Db2 V12R1M500 (FTB Disabled) No additional rebind	4975	369	-1.9
Db2 V12R1M500 (FTB Auto) No additional rebind	5059	356	-5.3

Consider the following points regarding the information in Table 5-2:

- FTB refers to the Fast Traversal Blocks feature, which is also known as fast index traversal.
- The rebind was done by using the default APREUSE(NONE) option.
- As elsewhere in this document, the Total CPU time/commit is calculated by adding the Db2 class 2 CPU time per commit and the total address space CPU time per commit.

A new performance feature that uses in-memory index optimization techniques (called fast index traversal) is introduced in Db2 12 (for more information about this feature, see Chapter 2, "Scalability enhancements and subsystem performance" on page 23). Significant performance benefits can be achieved by using this new feature for transactions that use random index access.

As the data in Table 5-2 shows, the Classic IRWW workload benefits greatly from this new Db2 12 feature. A significant reduction in Total CPU/commit time and a notable increase in transaction throughput, which is indicated by the ITR numbers, can be achieved by specifying INDEX_MEMORY_CONTROL=AUTO in DSNZPARM (to enable fast index traversal). ITR is the number of COMMITs/sec normalized to 100% CPU busy.

The significant reduction in getpage requests in Db2 12 with fast index traversal enabled is the main contributing factor for the ITR and Total CPU/Commit time improvement. The getpage activity details are listed in Table 5-3 on page 161. By using INDEX_MEMORY_CONTROL=AUTO, the Db2 12 measurements that use function level V12R1M500 yielded a 46.6% reduction in the number of getpages/commit compared to the Db2 11 NFM measurement.

Table 5-3 One-way classic IRWW that uses RELEASE(COMMIT) getpages

Classic IRWW measurements	Getpages /commit	Db2 12 vs. Db2 11 Getpages/commit (delta %)
Baseline: Db2 11 NFM	81.91	N/A
Db2 V12R1M500 (FTB Disabled)	81.90	0.0
Db2 V12R1M500 (FTB Auto)	43.64	-46.7

One other finding from the measurement runs is that rebinding application packages appeared to provide a small benefit in Db2 12, even for applications that use simple SQL, such as the classic IRWW workload.

Real storage usage

The use of the new fast index traversal feature in Db2 V12 requires extra real memory to store the FTBs. A new index manager storage pool is created for each Db2 subsystem or member to store the FTBs.

In the one-way data sharing scenario that is described in this section INDEX_MEMORY_CONTROL=AUTO or DISABLED were used. With the AUTO setting, Db2 uses a minimum of 10 MB, or up to 20% of the allocated buffer pool storage (whichever is larger). The increase in Db2 V12 real storage is shown by the increase in non-buffer pool storage, as listed in Table 5-4.

Table 5-4 One-way classic IRWW that uses RELEASE(COMMIT) DBM1 non-buffer pool storage

Classic IRWW measurement	DBM1 Non-Buffer Pool Storage (MB)	Db2 12 vs. Db2 11 Non-Buffer Pool Storage (delta %)
Baseline: Db2 11 NFM	1391	N/A
Db2 V12R1M100 (FTB Auto) No rebind	1745	25.4
Db2 V12R1M100 (FTB Auto) After rebind	1741	25.2
Db2 V12R1M500 (FTB Disabled) No additional rebind	1641	18
Db2 V12R1M500 (FTB Auto) No additional rebind	1746	25.6

When INDEX_MEMORY_CONTROL=AUTO is used, an increase in non-buffer pool storage usage is clearly seen, as listed in Table 5-4.

5.2.2 Db2 11 to Db2 12 migration scenarios (two-way data sharing)

The two-way data sharing environment is configured similarly to the one-way environment. Both Db2 members are running on one z13 LPAR, each with four dedicated general CPs available. The transactions are generated from two IMS subsystems, one on each of the LPARs with connections to the Db2 member local to them.

The measurements that were taken during typical Db2 11 to Db2 12 migration stages by using a two-way data sharing configuration are listed in Table 5-5 on page 162.

Table 5-5 Two-way classic IRWW that uses RELEASE(COMMIT) ITR and CPU time

Classic IRWW measurement	Average ITR (commits/sec) per member	Total CPU/commit (microseconds) per member	Db2 12 vs. Db2 11 CPU/commit (delta %)
Baseline: Db2 11 NFM	3584	653	N/A
Db2 V12R1M100 (FTB Auto) No rebind	3635	629	-3.7
Db2 V12R1M100 (FTB Auto) After rebind	3548	621	-4.8
Db2 V12R1M500 (FTB Disabled) No additional rebind	3672	622	-4.7
Db2 V12R1M500 (FTB Auto) No additional rebind	3725	607	-7.0

When running the classic IRWW workload in a two-way data sharing environment, the effects of the data sharing overhead that is triggered by introducing inter-Db2 read/write interest can be observed. When comparing the measurement results of the same migration step between the one-way (see Table 5-2 on page 160) and two-way (see Table 5-5) data sharing tests, the data sharing overhead is evident and reflected in an increase in Total CPU/Commit time and reduction of the ITR per Db2 member.

However, the benefits of the fast index traversal feature are not diminished when comparing the two-way to the one-way data sharing group measurements.

5.3 Distributed IRWW workload performance measurements

The distributed IRWW workload is the main workload that is used to evaluate Db2 12's performance capabilities in distributed environments. The workload includes the following transactions:

- SQCL: CLI (dynamic) that uses Db2 Connect
- JDBC: JDBC (dynamic) that uses JCC T4 driver
- SQLJ: Static Java that uses JCC T4 driver
- SPCB: Stored procedures in COBOL (static) that uses Db2 Connect
- SPSJ: Stored procedures in SQLJ (static) that uses JCC T4 driver
- SPNS: Stored procedures in native SQL (static) that uses JCC T4 driver
- SP500: Total of 500 stored procedures in native SQL (static) that uses JCC T4 driver

Each of these transactions issues SQL statements to simulate the same set of business functions, as described in A.3, "IRWW distributed workload" on page 379.

The test environment that was used to run these workloads used the following configuration:

- A z13 z/OS LPAR with 3 dedicated general CPs and 32 GB of memory that used z/OS 2.2.
- A Linux on System z on a z13 LPAR with three general CPs with Db2 Connect and Db2 for Linux, UNIX, and Windows 10.5 (s130428).
- JCC driver 3.66.33, JDK 1.7.0, IBM J9 VM (build 2.6, JRE 1.7.0 Linux s390x-64 20121024_126071).

- IBM HiperSockets communication is used between the z/OS and Linux for System z LPARs.
- All packages are bound by using the RELEASE(DEALLOCATE) BIND option. The `MODIFY DDF PKGREL` command is used to control the use of the high-performance DBAT feature.

Multiple measurement runs that use this workload were analyzed to determine how well Db2 12 can process distributed work. The evaluation process focuses on identifying performance differences between Db2 11 and Db2 12. The Db2 12 system was running at function level V12R1M500. All packages were rebound at that function level before the measurements were conducted. The total transaction CPU time per commit and Internal Throughput Rate (ITR) are the two main key indicators that were used during the evaluation.

5.3.1 Distributed IRWW workload CPU usage

The measurement data was collected on the Db2 server side. From the collected data, Db2 accounting, statistics, and RMF reports were generated for analysis. The measurements were done on an LPAR with no zIIP processors available, so the data does not account for zIIP usage.

The Total CPU time per commit serves as the main metric in determining the CPU cost during these performance tests. It is calculated by adding up the total address space CPU time per commit from all the Db2 system address space (MSTR, DBM1, DIST, and IRLM) in the Db2 statistics report. The resulting time accounts for the CPU cost of the network communication, and the Db2 agent CPU cost.

Non-stored procedure workloads

Table 5-6 lists the delta as the percentage difference between the measurement runs that use Db2 11 and Db2 12, by using the DDF address space total CPU time per commit and Total CPU time per commit metrics for the three non-stored procedure transactions of the distributed IRWW workload. For example, -3.61 means Db2 12’s total CPU time per commit improved by 3.61% over Db2 11. All of the workloads showed a CPU cost reduction of up to 3.75%, which demonstrates that Db2 12 performed slightly better compared with Db2 11 when processing non-stored procedure type distributed workloads.

Table 5-6 Distributed IRWW non-stored procedure workloads CPU time delta percentage

Delta %	**PKGREL(COMMIT)**		**PKGREL(BNDOPT)**	
Workload	**DDF/commit**	**Total CPU time /commit**	**DDF/commit**	**Total CPU time /commit**
SQCL	-3.32	-3.61	-2.08	-2.60
JDBC	-0.73	-0.28	-2.68	-2.87
SQLJ	-3.25	-3.75	-1.31	-2.12

Stored procedure workloads

Table 5-7 lists the delta (in percentage) between Db2 12 and Db2 11 measurement runs for different types of stored procedure workloads. Db2 accounting class 1 CPU time and Total CPU time per commit from statistics are used to compare the performance difference between the two Db2 versions.

Table 5-7 Distributed IRWW stored procedure workloads CPU time delta percentage

Delta %	**PKGREL(COMMIT)**		**PKGREL(BNDOPT)**	
Workload	**CL1 CPU time**	**Total CPU time /commit**	**CL1 CPU time**	**Total CPU time /commit**
SPCB	-2.05	-3.07	-3.00	-4.31
SPSJ	-2.98	-3.50	-3.39	-3.95
SPNS	-1.85	-2.61	-2.41	-3.49
SP500	-4.12	-4.06	-0.96	-1.20

The data shows that Db2 12 can offer a CPU cost reduction of up to 4.3% compared to Db2 11. The CPU time reduction can be attributed to several factors, such as Db2 storage management improvements, DDF improvements, and the use of the fast index traversal feature in Db2 12.

5.3.2 Distributed IRWW workload throughput

From the data that is gathered during the different measurement runs, the normalized throughput (ITR) is calculated to use a basis for comparison between the work processing capabilities of Db2 11 and Db2 12. ITR is used because it accounts for the LPAR CPU usage rate.

The data that is listed in Table 5-8 lists ITR improvements ranging from a few percent up to 9% for Db2 12 compared to Db2 11 for the different types of workload. The improvements are the result of the collective improvements from the new fast index traversal feature, the claim/declaim skipping technique when RELEASE(DEALLOCATE) is used, EDM pool management optimization, and other areas of improvements in Db2 12.

Table 5-8 Distributed IRWW workloads ITR delta percentage

Delta (%)	**Normalized throughput (ITR)**	
Workload	**PGKREL(COMMIT)**	**PKGREL(BNDOPT)**
SQCL	4.06	3.68
SPCB	2.49	3.89
JDBC	1.82	2.25
SQLJ	2.67	2.29
SPNS	2.56	4.02
SPSJ	3.34	4.05
SP500	4.85	8.94

5.3.3 Real storage usage

Db2 12 for z/OS implemented several changes in the area of virtual and real storage management to improve performance. Before Db2 12, several Db2 components used their own storage management processes, which can result in unintended performance effects. In Db2 12, these Db2 components were changed to use the services that are provided by the Db2 Storage Manager component to manage storage for them in a centralized manner.

This change enables Db2 Storage Manager to gain more insight into how the Db2 storage is used within the entire Db2 address space so it can communicate better with other Db2 resource managers. This communication helps to determine the amount of real storage available for conditional use and thus improves the stability and availability of the Db2 subsystem.

A small percentage of real storage increase is expected as a result of these changes; however, this increase also was observed in previous Db2 versions.

A series of performance measurements were conducted by using the distributed IRWW workload to evaluate the changes in Db2's storage usage in Db2 12. Db2 storage consumption is closely related to the number of concurrent active threads in the Db2 subsystem. Therefore, different numbers of concurrent active distributed threads were used as the main variable during these tests. Respectively, 500, 1000 and 1500 active threads were used for the Db2 12 and Db2 11 tests by using a non-data sharing environment.

All measurements were done by using the same z/OS 2.2 system that was running on a z13 processor.

Table 5-9 lists the Db2 storage usage below and above 2 GB for all performance tests for Db2 11 and 12 by using a varying number of concurrent threads.

Table 5-9 IRWW Distributed workload Db2 storage usage with varying number of threads

In MB	**Db2 11 (NFM)**			**Db2 12 (V12R1M500)**		
Number of concurrent threads	**500**	**1000**	**1500**	**500**	**1000**	**1500**
DBM1 PRIVATE STORAGE BELOW 2 GB						
FIXED POOL BELOW	3.37	4.97	6.51	3.83	5.79	7.81
VARIABLE POOL BELOW	0.94	1.09	1.26	0.95	1.12	1.29
GETMAINED BELOW	0.1	0.1	0.1	0.1	0.1	0.1
DBM1 PRIVATE STORAGE ABOVE 2 GB						
FIXED POOL ABOVE	11.22	16.03	23.29	9.12	15.83	21.30
VARIABLE POOL ABOVE	1.00	1.98	2.96	1.23	2.22	3.14
GETMAINED ABOVE	0.22	0.22	0.22	0.2	0.2	0.2
STORAGE MGR CONTROL BLOCKS ABOVE 2 GB	1.34	1.34	1.34	1.34	1.34	1.34
REAL STORAGE IN USE						
REAL FOR DBM1	20201.53	20317.62	20416.81	20437.98	20522.03	20585.32
SUBSYSTEM SHARED STORAGE ABOVE 2 GB	1652.6	3100.18	3248.73	1788.55	3257.56	3443.85
TOTAL REAL STORAGE IN USE-SUMMARY	21959.38	23591.04	23891.29	22312.7	23926.17	24225.65

The measurements results indicate that the following changes occurred in Db2 storage usage in Db2 12 compared to Db2 11:

- Variable storage below 2 GB was reduced approximately 2.3%.
- Fixed pool above 2 GB was reduced about 8%.
- Real storage for DBM1 (including buffer pool storage) that is in use increased by approximately 6%.
- Total Real Storage that is in use increased by approximately 1.6%.

5.3.4 Conclusion

By using many of the Db2 12 improvements in storage management, EDM pool management, and the introduction of the fast index traversal feature, Db2 12 can offer over 4% reduction in CPU cost, with a 4% ITR increase for the distributed IRWW workloads when compared to Db2 11. However, a (modest) real storage usage increase of 1.6% was observed for the workload when it was run with Db2 12.

5.4 IBM Brokerage transaction workload

The IBM Brokerage transaction workload is a complex OLTP workload that runs various SQL stored procedures to simulate transactions in a brokerage firm. For more information about the workload, see Appendix A, “IBM Db2 workloads” on page 375.

5.4.1 One-way data sharing performance measurements

This complex OLTP workload that uses a one-way data sharing configuration was migrated from Db2 V11 NFM to Db2 12 at function level V12R1M100. Then, function level V12R1M500 was activated. Performance measurements were conducted at each step to track how the workload performs at each stage of the migration process.

The baseline test was run by using Db2 11 NFM. Then, a second test was performed after migrating to Db2 12 at function level V12R1M100 without rebinding any packages. The third test was done with Db2 12 still at the V12R1M100 function level, but after rebinding all application packages. The fourth and final test was done after Db2 12 function level V12R1M500 was activated, and all packages being rebound a second time. All scenarios that were used during these measurements featured plans and packages that are bound with the RELEASE(COMMIT) bind option.

The measurements were performed by using the following environment:

- A z13 LPAR with 12 general CPs
- z/OS 2.2

Also, the Db2 11 NFM one-way data sharing measurement was used as the baseline measurement.

The Db2 class 2 CPU time improvement percentages of the different Db2 12 performance measurements of the IBM Brokerage online transaction workload (by using the Db2 11 measurement as the basis for comparison) is shown in Figure 5-4 on page 167.

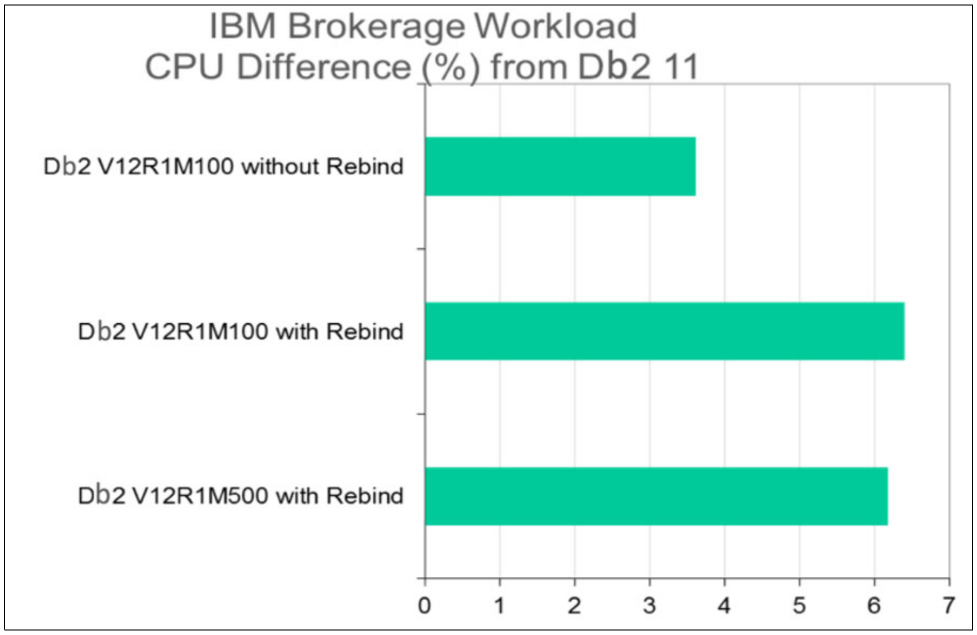


Figure 5-4 Class 2 CPU time improvements comparing against the Db2 11 baseline

The workload runs 30 stored procedures and achieves a transaction rate of 5000 - 6000 transactions per second. The following CPU time reduction percentages were achieved in the three Db2 12 migration stages:

- After migration to Db2 12 at function level V12R1M100, without running a REBIND of the stored procedure packages, we see a 3.6% CPU time reduction compared to the Db2 11 baseline test.
- A REBIND is run for all packages specifying APREUSE(ERROR) to ensure that the same access paths are used for all the packages. In this scenario, Db2 class 2 CPU time is reduced further to 6.4%.
- The final measurement that is performed after activating Db2 12 function level V12R1M500, and a REBIND using APREUSE(ERROR) for all the packages, resulted in a similar Db2 class 2 CPU time saving of approximately 6% compared to Db2 11 baseline.

5.4.2 Two-way data sharing performance measurements

For this set of measurements, migration of the same complex OLTP workload was performed the same way as described in 5.4.1, “One-way data sharing performance measurements” on page 166 for the one-way data sharing IBM Brokerage workload benchmark tests. However, the measurements in this test were conducted by using a two-way data sharing environment.

As before, the system is migrated from Db2 11 NFM to Db2 V12R1M100, and then to function level V12R1M500. Performance measurements were captured as the migration moved forward through the different stages to demonstrate how the workload performs at each of these migration phases by using a two-way data sharing configuration.

The baseline test was run by using Db2 11 NFM. Then, the second test was done after migrating to Db2 12 at function level V12R1M100, and after REBINDing all application packages that use APREUSE(ERROR) to preserve the access paths from the Db2 11 NFM baseline test.

The third test was done after activating Db2 function level V12R1M500, and a REBIND of all packages with APREUSE(ERROR), which allowed the workload to run at a higher function level and keep the access path of the Db2 11 NFM baseline test.

The fourth and final test was done with Db2 at function level V12R1M500 and a REBIND of all packages, this time with APREUSE(NONE), which allows new access paths to be selected for applications when applicable. As before, all scenarios use plans and packages that are bound with RELEASE(COMMIT).

The measurements were performed by using the following environment:

- Two z13 LPARs with six general CPs each.
- Two internal coupling facilities (ICF) with three dedicated CPs each.
- Four ICPs and four CIBs are shared between two LPARs and two ICFs.
- z/OS 2.2 and CFCC Level 21.

Many internal optimizations and enhancements apply to Db2 12 that are not applicable to older Db2 releases, which contributes to the performance improvement of Db2 12 at function level V12R1M100 (and above). The following internal optimizations and new features are included:

- The use of a new PL/X compiler for Db2 12 performance sensitive modules.
- Log latch relief (LC19).
- The use of the fast index traversal feature is available at function level V12R1M100, but only for non-group buffer pool-dependent index objects.

At function level V12R1M500, Db2 12's fast index traversal feature can also be used for index objects that are group buffer pool dependent when they meet the fast index traversal criteria. This feature can increase the performance benefits the fast index traversal functionality provides. REBIND with APREUSE(NONE) allows new access paths to be selected (when applicable), which introduces the possibility that better access paths are picked to achieve better application performance.

Figure 5-5 on page 169 shows the getpage reduction per transaction for the different migration scenarios. The getpage reduction (when going from Db2 11 NFM to Db2 12 at function level V12R1M100) is the result of the fast index traversal feature, although the usage is limited to non-GBP dependent index objects.

The reduction in the number of getpages when going from function level V12R1M100 to V12R1M500 is because at that function level, GBP-dependent indexes can also use the fast index traversal feature.

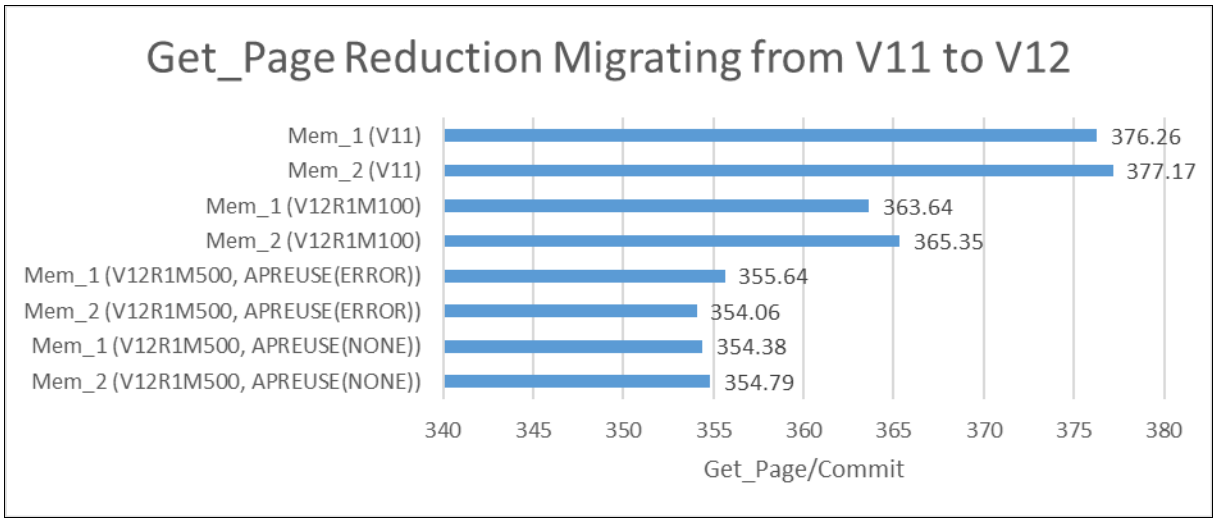


Figure 5-5   Getpage reduction for different migration stages

At function level V12R1M100, 300 indexes were using the fast index traversal feature, with all allocated FTBs using 120 MB of storage. At function level V12R1M500, 1020 indexes were using this feature, with 415 MB of storage used for FTBs.

The access path changes that were introduced by REBINDing with APREUSE(NONE) were minor. The getpage count is nearly the same for the measurement that was taken at function level V12R1M500 after rebinding the packages with APREUSE(ERROR) and the measurement that was taken at function level V12R1M500 after rebinding with APREUSE(NONE).

The total Db2 CPU time per transaction, which consists of the Db2 class 2 CPU time plus MSTR, DBM1, and IRLM CPU time per transaction for each migration scenario is shown in Figure 5-6.

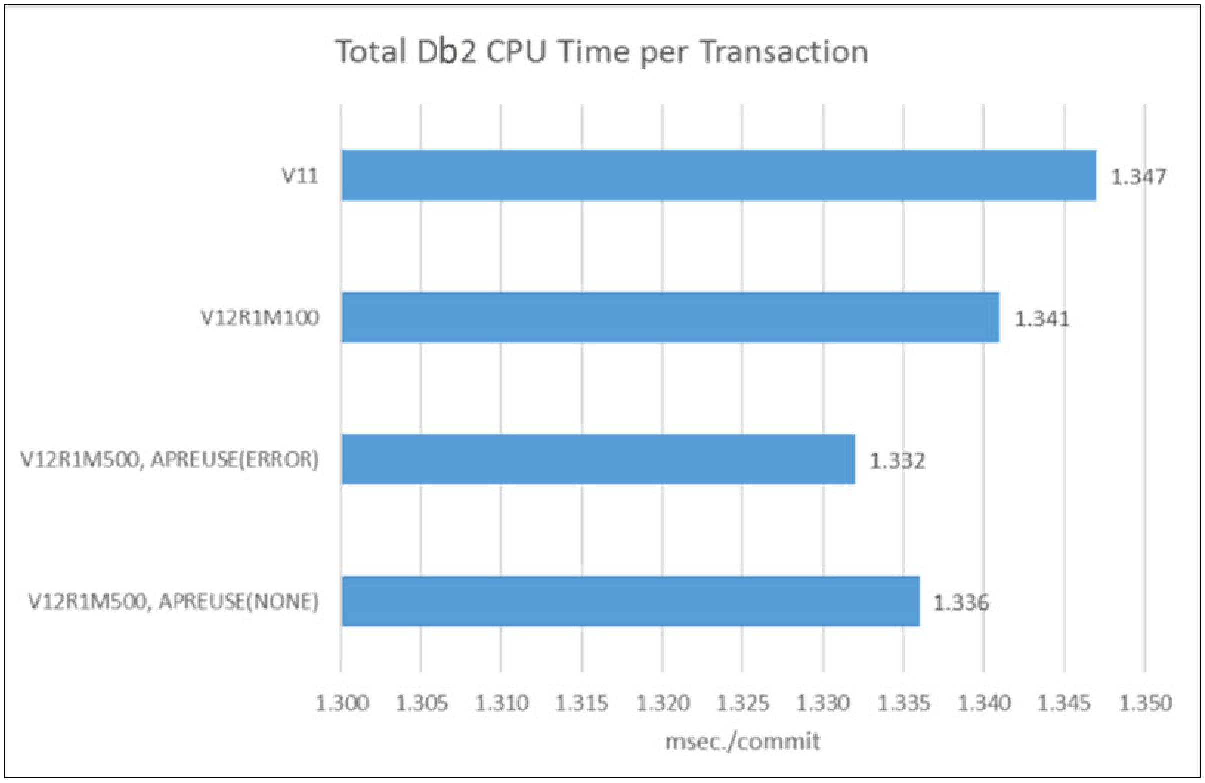


Figure 5-6 Total Db2 CPU time per transaction

Db2 12 at function level V12R1M100 benefits from many Db2 12 internal optimizations that are not available in older Db2 releases. At function level V12R1M500, Db2 shows more performance benefits because GBP-dependent index objects also become eligible for fast index traversal.

After rebinding with APREUSE(NONE), we observed minor access path changes in several packages of the workload. However, Db2 accounting reports indicated similar getpage counts per occurrence for the packages with changed access paths. Also, class 7 CPU time was nearly identical for these packages.

The data sharing group's ITR for the different migration scenarios is shown in Figure 5-7. The ITR increases during the migration from V11 to V12. It is evident that Db2 12 at function level V12R1M500 performs better than Db2 11 NFM for this IBM brokerage workload in a two-way data sharing environment.

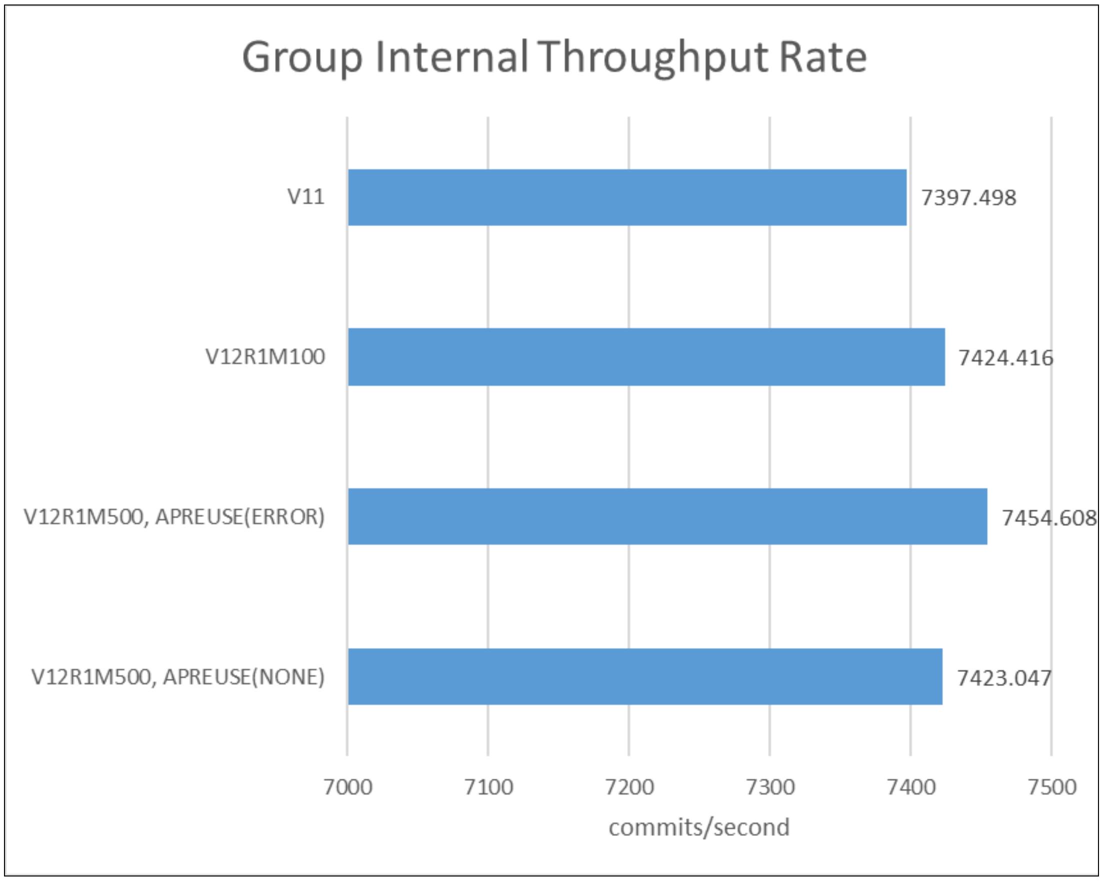


Figure 5-7 Group ITR

As for the memory consumption, the Db2 DBM1 address space's real storage usage is higher for Db2 12, as shown in Figure 5-8.

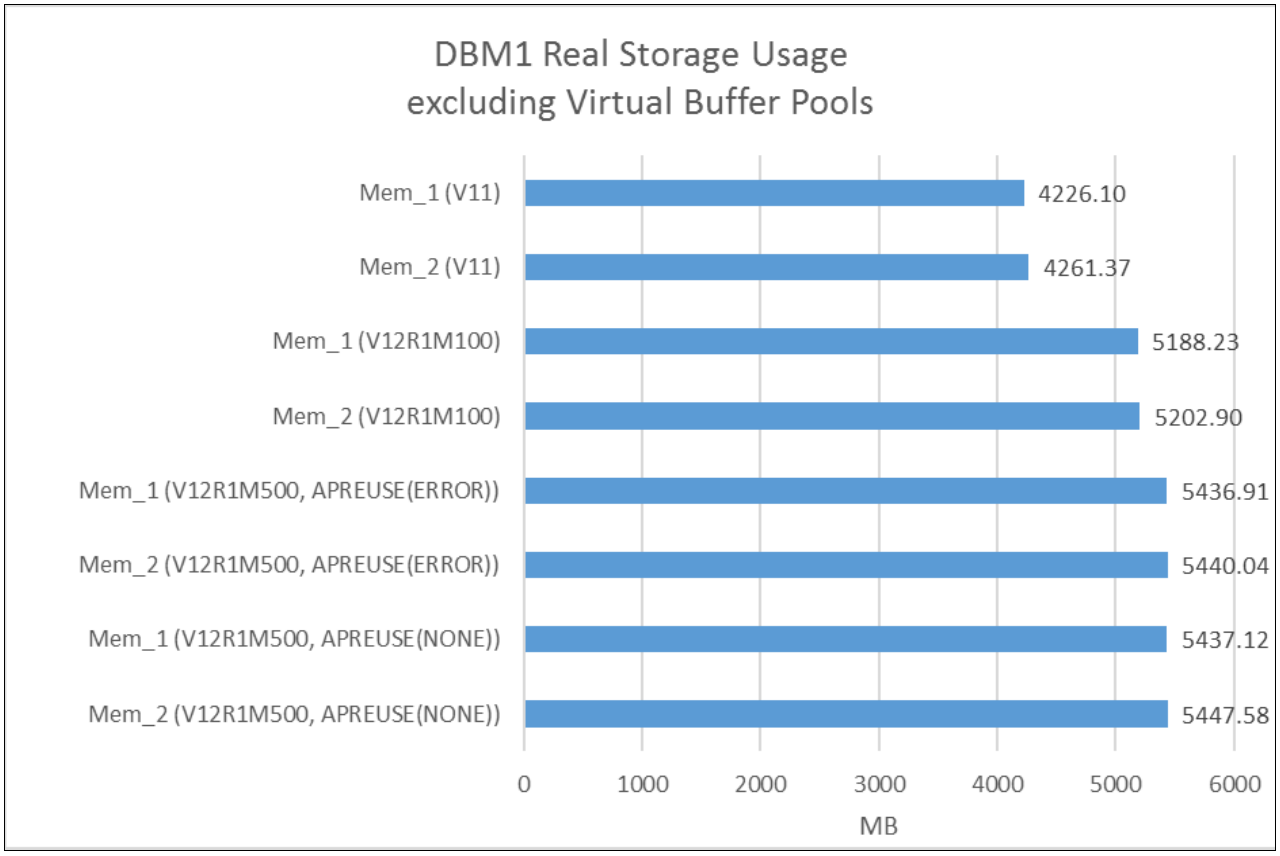


Figure 5-8 DBM1 real storage usage excluding virtual buffer pools

At function level V12R1M100, the Db2 DBM1 address space uses approximately 1 GB more real storage than Db2 11 NFM. Out of that storage, approximately 120 MB is allocated by the fast index traversal feature for FTBs.

At function level V12R1M500, another 250 MB of real storage is used by the Db2 DBM1 address space, and FTB storage grew from 120 MB to approximately 415 MB. This growth indicates that most of the extra DBM1 storage usage at the V12R1M500 function level compared to V12R1M100 is for more FTB storage.

In summary, migration of the IBM Brokerage Transaction Workload in a two-way data sharing environment, from Db2 11 NFM to Db2 12 at function level V12R1M500, experiences performance improvement during the migration process, as indicated by the decreasing Db2 CPU time per transaction and increasing ITR when Db2 is migrated from V11 to V12. DBM1 real storage usage with Db2 12 increased compared to Db2 11 NFM, as a result of internal storage management changes and the new fast index traversal feature.

5.5 RTW workload

The Relational Transaction Workload (RTW) is a standard workload that is used by the CICS performance team in Hursley to assess changes in performance characteristics for new CICS releases for applications accessing Db2. The Db2 team employed the RTW workload as one of its performance test benchmark workloads to achieve better performance analysis coverage for CICS-Db2 type of workloads. The workload was modified slightly compared to the version the CICS team uses to better reflect the Db2 behavior of the workload.

The RTW workload includes the following characteristics:

- All programs that make up the workload are written in COBOL.
- The workload uses 20 (data) tables in segmented table spaces, 11 non-unique indexes, and 16 unique indexes.
- It issues on average 200 Db2 calls per transaction.
- The transaction mix includes 26% select, 3% insert, 7% update, 3% delete, 6% open cursor, 36% fetch cursor, 6% close cursor operations.

The following environment configuration is used for the non-data sharing RTW workload:

- One z13 LPAR with eight general CPs
- Four CICS regions that are running CICS TS V5.1
- A non-data sharing Db2 subsystem
- z/OS 2.1
- DS8870 DASD controller
- TPNS on another z13 LPAR with four general CPs to drive the workload

The following environment was used by the two-way data sharing RTW workload:

- Two z13 LPARs with eight general CPs each
- Eight CICS regions running CICS TS 5.1 with four regions that are running on each of the LPARs
- Two z13 ICF LPARs running CFLEVEL 21 CFCC
- Two Db2 subsystems, each running on one of the two LPARs
- DS8870 DASD controller
- TPNS on another z13 LPAR with four general CPs to drive the workload

5.5.1 Performance measurements for non-data sharing RTW workload

Five performance measurements that use the RTW workload were conducted during different migration stages, from Db2 11 NFM to Db2 12 at function level V12R1M500. A typical customer might run their Db2 system in any of the following migration phases:

- Baseline measurement with Db2 11 NFM
- Db2 12 at function level V12R1M100 measurement without rebinding application packages
- Db2 12 at function level V12R1M100 measurement after rebinding application packages with APREUSE(ERROR)
- Db2 12 at function level V12R1M500 measurement with application packages previously rebound at V12R1M100 with APREUSE(ERROR)
- Db2 12 at function level V12R1M500 measurement after rebinding application packages with APREUSE(NONE)

The analysis of the results mainly focuses on the following key performance indicators:

- Db2 total CPU time per commit (calculated by adding up the Db2 class 2 CPU time per commit and total Db2 address space CPU time per commit)
- Internal throughput rate (ITR), which is calculated by dividing the workload's commit rate by the LPAR CPU utilization percentage

The total CPU time per commit results for the five measurement scenarios are shown in Figure 5-9.

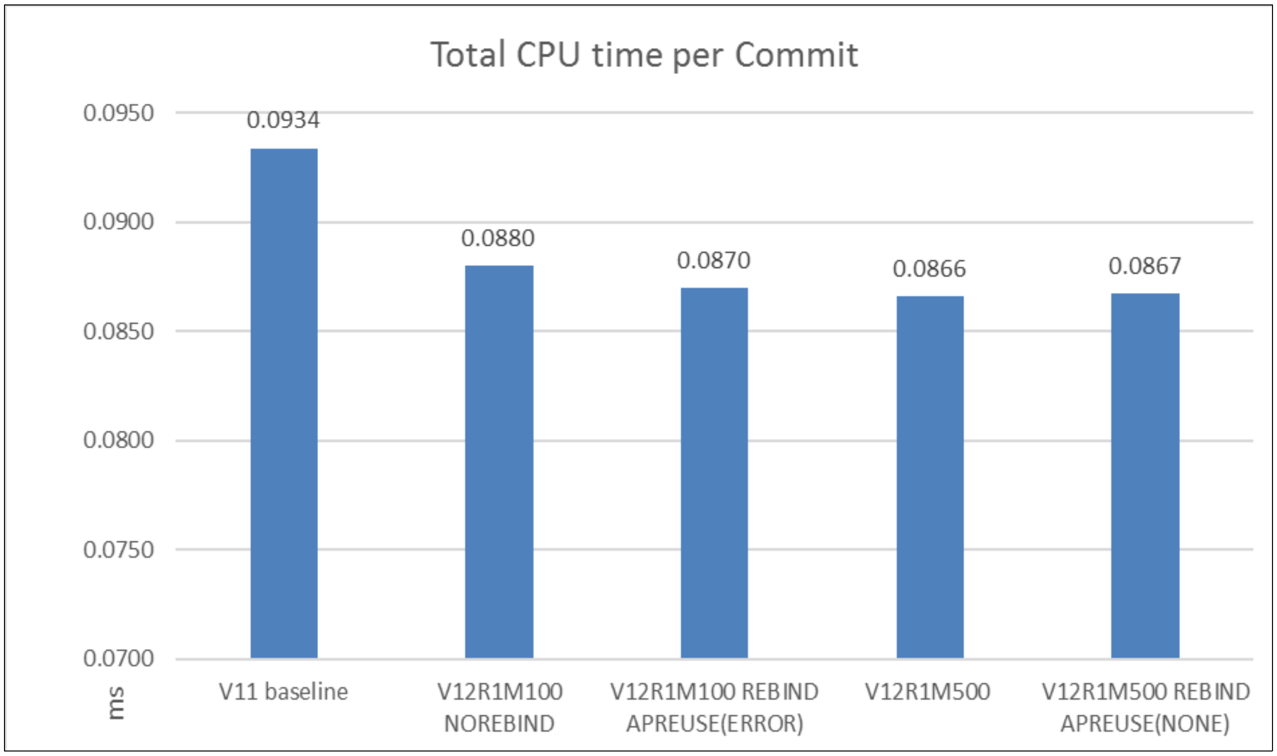


Figure 5-9 Total CPU time per commit

The ITR results for the same five measurement scenarios are shown in Figure 5-10.

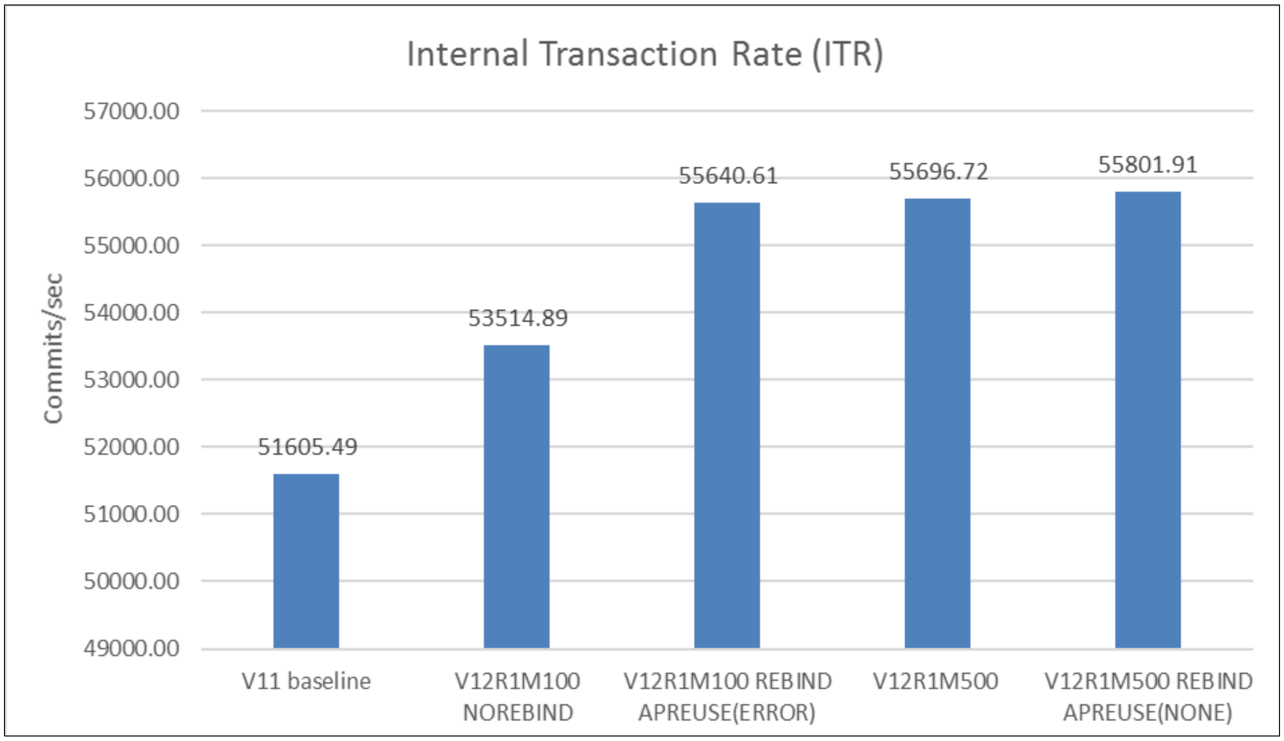


Figure 5-10 ITR

Based on the test results, the following observations can be made:

- All the tests that were run with Db2 12 performed better than the Db2 11 NFM baseline test.
- After migrating to Db2 12 function level V12R1M100 without rebinding the application packages, some package puffing[1] overhead occurred when the workload was run. As most of the Db2 12 internal optimizations are in effect at function level V12R1M100, a 5.76% saving in total CPU time per commit, and a 3.7% ITR increase was observed compared to the baseline test.
- After rebinding all the application packages with APREUSE(ERROR), all the original access paths were honored. With the package puffing overhead removed, comparing to the baseline Db2 11 NFM measurement, the total CPU time per commit improved further to 6.86% while the ITR increase grew to 7.82%.
- The Db2 12 measurement at function level V12R1M500 without rebinding the application packages again (thus running with packages rebound at function level V12R1M100) showed similar total CPU time per commit and ITR results than the previous test. The result produced an 7.26% CPU improvement and 7.93% of ITR boost compared to the Db2 11 baseline.
- Rebinding all application packages with APREUSE(NONE) after activating Db2 12 function level V12R1M500 resulted in zero access path changes; therefore, this final test performed similar as the previous measurement, showing a 7.15% reduction in total CPU time per commit value and an 8.13% increase in ITR compared to the baseline Db2 11 NFM test.

[1] Db2 typically expects control structures to be in the current release format. If that is not the case, for example because a package was not rebound on the current release, Db2 adjusts the control structure to be in line with what the current release is expecting. This internal process is sometimes called *puffing*.

The DBM1 address space's real storage usage during all five measurements is shown in Figure 5-11.

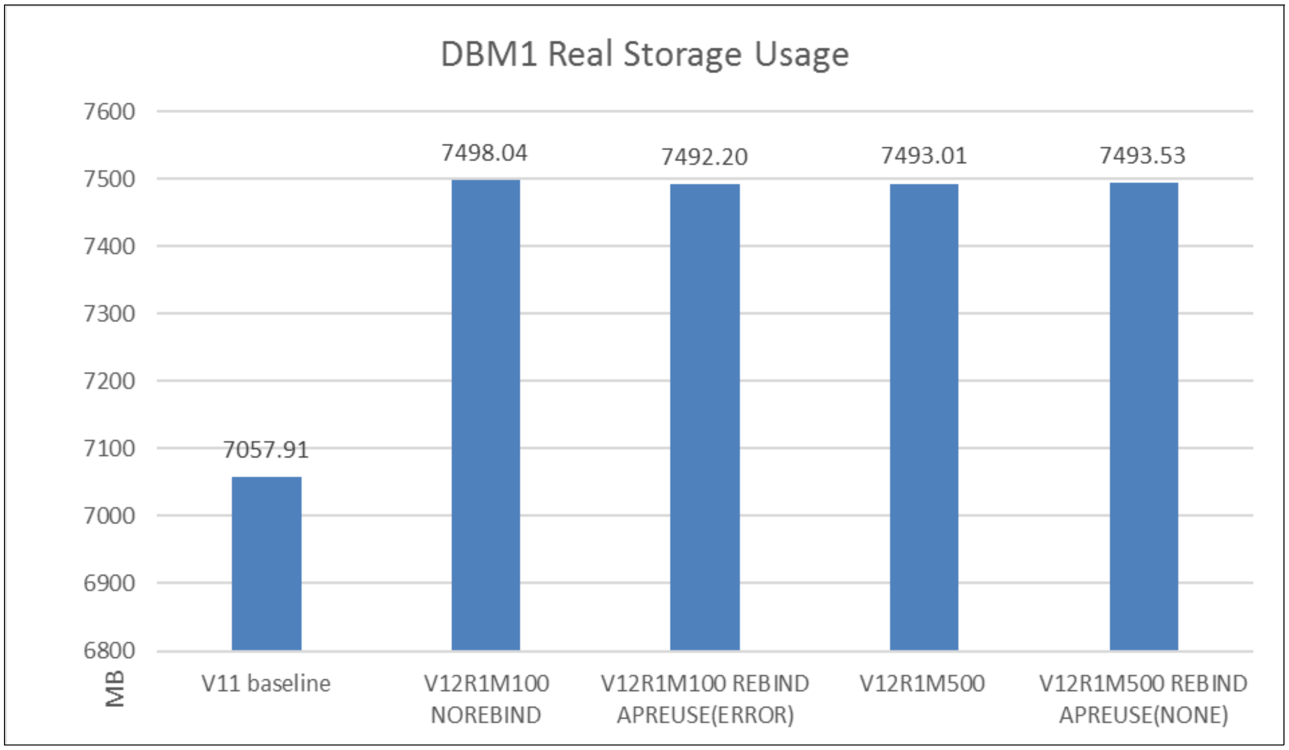


Figure 5-11   DBM1 real storage usage

After Db2 is migrated from Db2 11 to Db2 12 (as shown in Figure 5-11), approximately 450 MB more real storage is needed for the DBM1 address space because of the storage management-related changes that were introduced in Db2 12. In this case, the main contributor to the storage growth is an increase in the size of buffer manager-related control blocks. As the RTW workload uses large buffer pools (more than 42 GB in total), the storage increase is more visible than in other workloads. The fast index traversal feature caused eight of the RTW workload's indexes non-leaf pages to be in-memory, although in total, all the FTBs took less than 1 MB of storage because the indexes are small.

Migrating the RTW workload that is running in a Db2 non-data sharing environment from Db2 11 to Db2 12 resulted in obvious performance improvements. Because no access path changes were made for any of the RTW application packages after migrating to Db2 12, the workload had similar performance characteristics during the last three measurements after the application packages were rebound at the V12R1M100 level. Real storage consumption after migrating to Db2 12 grew by 450 MB and less than 1 MB out of the 450 MB was used by the fast index traversal feature.

5.5.2 Performance measurements: Two-way data sharing RTW workload

The same procedure that was used to conduct the RTW workload measurements in a non-data sharing environment was used to take performance measurements by using the RTW workload in a two-way data sharing environment. The same key performance indicators, total CPU time per commit and ITR were used to analyze the performance results.

The performance results of the tests that are shown in Figure 5-12 and Figure 5-13 on page 178. The results of each Db2 data sharing member and the group-level average CPU time and total ITR also are shown in the charts.

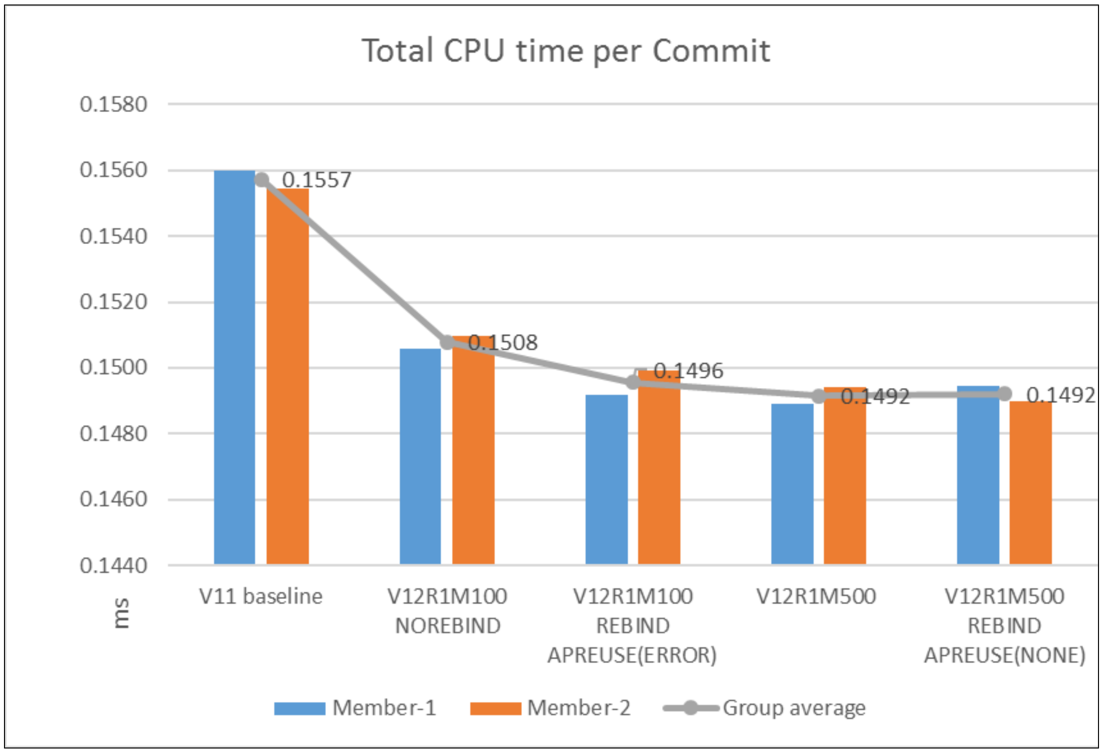


Figure 5-12 Total CPU time per commit

Figure 5-13 shows the ITR for the different RTW workload runs that use two-way data sharing environment.

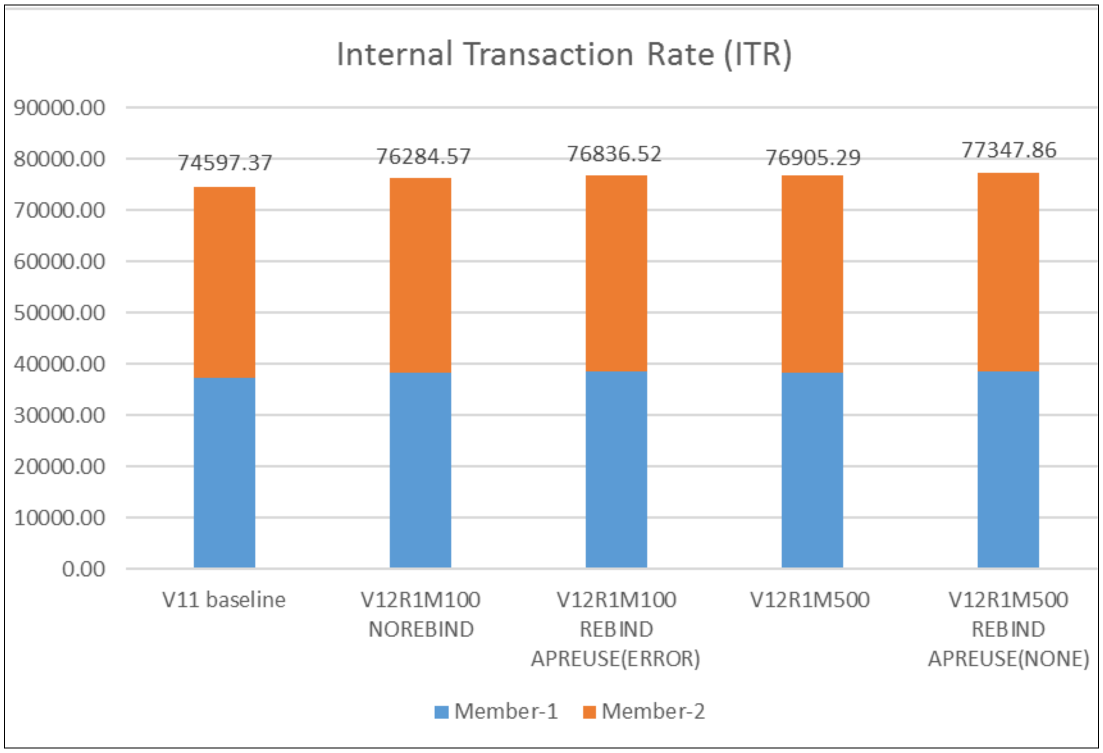


Figure 5-13 Data sharing ITR

Based on the test results, the following observations were made:

- All Db2 12 tests performed better than the Db2 11 NFM baseline test.
- As with the non-data sharing measurements, the package puffing overhead is still present without rebinding the Db2 application packages after migrating to Db2 12 at function level V12R1M100. As most of the Db2 12 internal optimizations are enabled even without rebinding, Db2 still showed a 3.17% group level improvement in total CPU time per commit and a 2.26% group level ITR increase compared to the Db2 11 baseline test.
- Six indexes included FTBs that were created with a total size of 270 KB for each Db2 member.
- Rebinding all application packages with APREUSE(ERROR) ensures that all of the original access paths are reused. Without the package puffing overhead, comparing to the baseline Db2 11 measurement, the Db2 12 V12R1M100 measurement's total CPU time per commit improvement compared to the Db2 11 baseline test reached 3.96% and ITR increased by 3%.
- The Db2 12 function level V12R1M500 measurements without rebinding the application packages (thus running with packages rebound at function level V12R1M100) performed a bit better than the previous test. The total CPU time per commit value improved by 4.22% and the Group ITR grew by 3.09% comparing to the Db2 11 baseline test.
- Eight indexes included FTBs that were created by using 851 KB of storage on each Db2 member.

- Rebinding the application packages with APREUSE(NONE) after activating function level V12R1M500 resulted in zero access path changes. Therefore, the final measurement performed about the same as the previous V12R1M500 measurement. Comparing to the baseline test, the total CPU time per commit value showed a 4.16% improvement, and the data sharing group's ITR increased by 3.69%.

The DBM1 real storage usage for each of the Db2 members during each of the five measurements is shown in Figure 5-14.

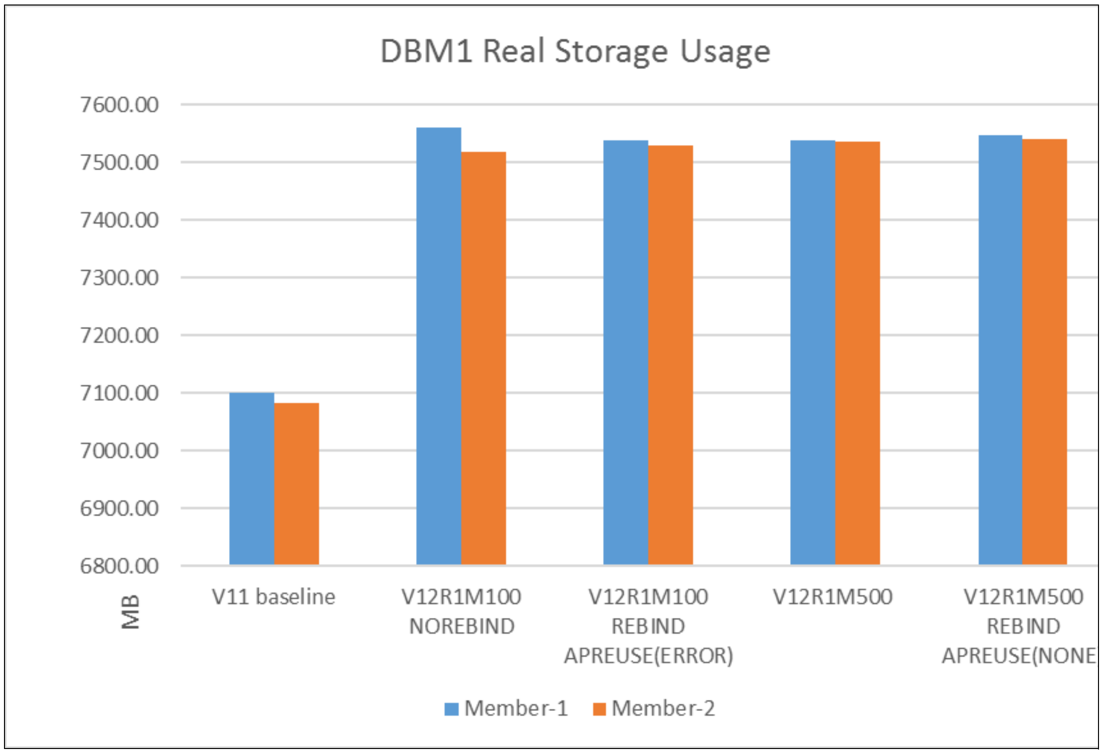


Figure 5-14 DBM1 real storage

After Db2 migrates from V11 to V12 (as shown in Figure 5-14), approximately 450 MB more real storage is needed for the DBM1 address space of each of the Db2 data sharing members because of the storage management-related changes that were introduced in Db2 12.

The fast index traversal feature caused six indexes to include FTBs that were allocated when at function level V12R1M100 taking up 270 KB of storage per Db2 member. Two more indexes included FTBs that were allocated after activating function level V12R1M500 and all eight FTBs combined took up 851 KB of storage per Db2 member.

When in a two-way data sharing environment is run, the RTW workload showed notable performance improvements after migrating to Db2 12 from Db2 11. Db2 12 at function level V12R1M500 causes two other indexes to use the fast index traversal feature compared to the workload that is running at function level V12R1M100. This situation occurs because at function level V12R1M500, GBP-dependent indexes are also eligible to use the fast index traversal feature. This use resulted in slightly higher CPU savings and better ITR results. Real storage usage after migrating to Db2 12 grew by about 450 MB for each of the Db2 members when the RTW workload is run.

5.6 High insert batch workload measurements

In this section, we describe the results of several performance measurements that are related to workloads with high insert rates for the following batch workload scenarios:

- Archive sequential insert
- Journal table insert
- Message queue insert and delete
- Random insert and delete workload

The measurements were conducted with Db2 11 and Db2 12 by using a two-way data sharing environment, with operating System z/OS 2.2 and z13 hardware. The analysis focuses on workload performance differences between Db2 11 and Db2 12.

5.6.1 Archive sequential insert

In this scenario, sequential insert performance was evaluated running a workload that used the following configuration:

- Three types of tables spaces: Classic partitioned table spaces (PTS), partition-by-growth (PBG) table spaces, and partition-by-range (PBR) table spaces
- Member cluster (MC) and row level locking (RLL)
- Three tables and six non-unique indexes
- A total of 10 partitions for PTS and PBR table spaces
- Multi-row (100) insert and committing after each rowset insert
- A two-way data sharing group with 100 concurrent threads

The programs first insert data to the empty tables. Then, it delete two-thirds of the rows to create “holes” in the pages. Finally, more rows are inserted into the populated tables.

The Db2 class 2 CPU time results (as shown in Figure 5-15 on page 181), demonstrate that the use of Db2 12 can improve class 2 CPU time by up to 22%. Consider the following points:

- For PBG table spaces, the CPU time is improved by 11% when inserting into empty tables, and by 8% when inserting into the populated tables.
- For PBR table spaces, the CPU time improved by 22% when inserting to the empty tables, and by 18% when inserting into populated tables.
- For PTS tables spaces, the CPU time improved by 14% when inserting to the empty tables, and by 15% when inserting into the populated tables.

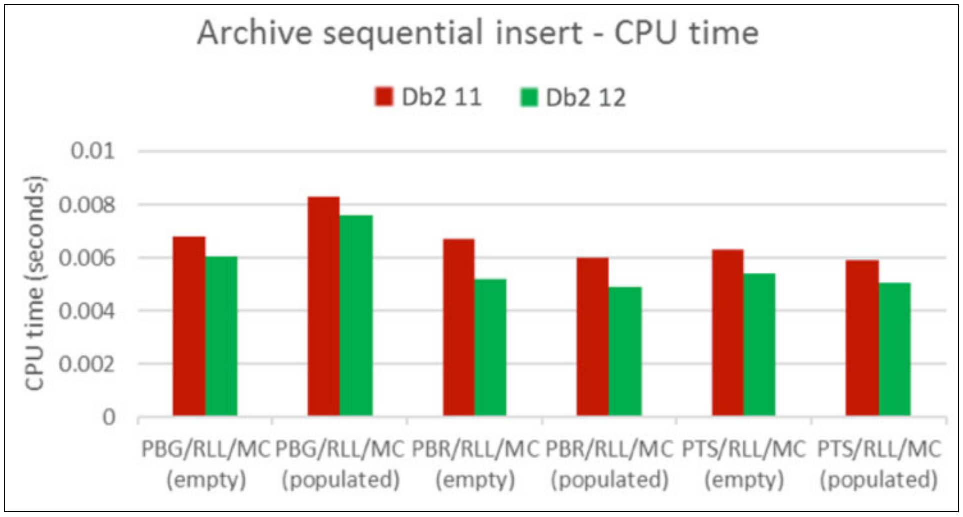


Figure 5-15 Archive sequential insert Class 2 CPU time per commit

The major factors that contribute to the CPU time improvement include the up to 94% getpage reduction for data pages (only for UTS PBR and PBG) because of the use of insert algorithm 2, the log write latch contention (LC19) reduction, and the buffer manager timer latch (LC23) removal.

The throughput results that are shown in Figure 5-16 show a significant external throughput rate (ETR) increase for PBG table spaces in Db2 12. However, for PBR table spaces, the insert throughput increase is observed only when inserting into populated tables.

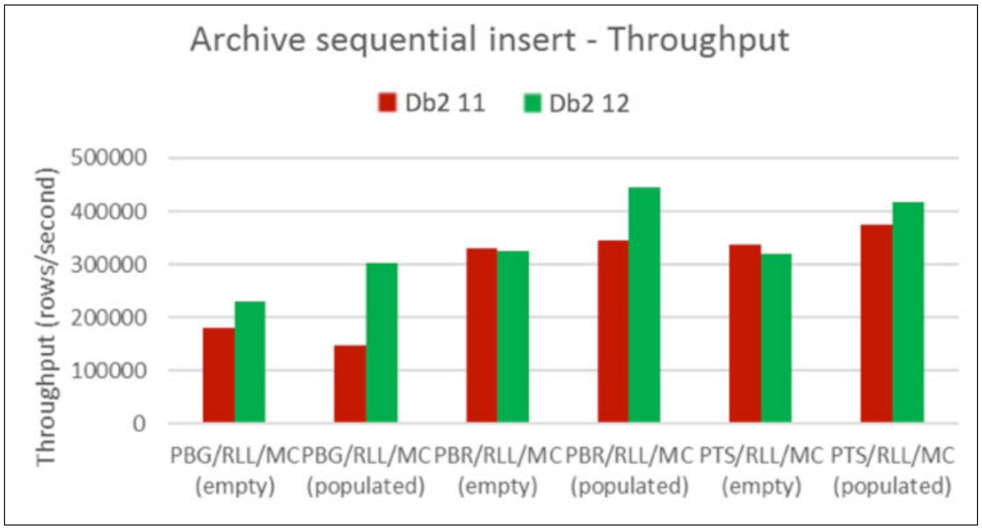


Figure 5-16 Archive sequential insert External throughput rate

For PTS table spaces, no obvious change in the throughput was observed when inserting into empty tables or populated tables because the throughput gain from the reduction in LC19 and LC23 contention is offset by a significant increase in the average suspension time for other Db2 latches.

5.6.2 Journal table insert

In this scenario, a sequential insert workload with the following configuration was used:

- Two types of table space: Classic PTS and PBR.
- Using the MEMBER CLUSTER with APPEND NO attribute, and MEMBER CLUSTER with APPEND YES.
- Row level locking (RLL).
- Three tables, five unique indexes, and one non-unique index. Each table has 20 partitions (note the difference in the indexes between the journal inserts and archive sequential inserts, as described in 5.6.1, "Archive sequential insert" on page 180).
- Multi-row (100) insert and commit after each rowset insert.
- A two-way data sharing environment with 100 concurrent threads.

The Db2 class 2 CPU time results (as shown in Figure 5-17) show CPU time improvements of up to 24% for the journal table insert scenario in Db2 12, compared to Db2 11. The contributing factors to this improvement are similar to factors as described in the previous insert scenario.

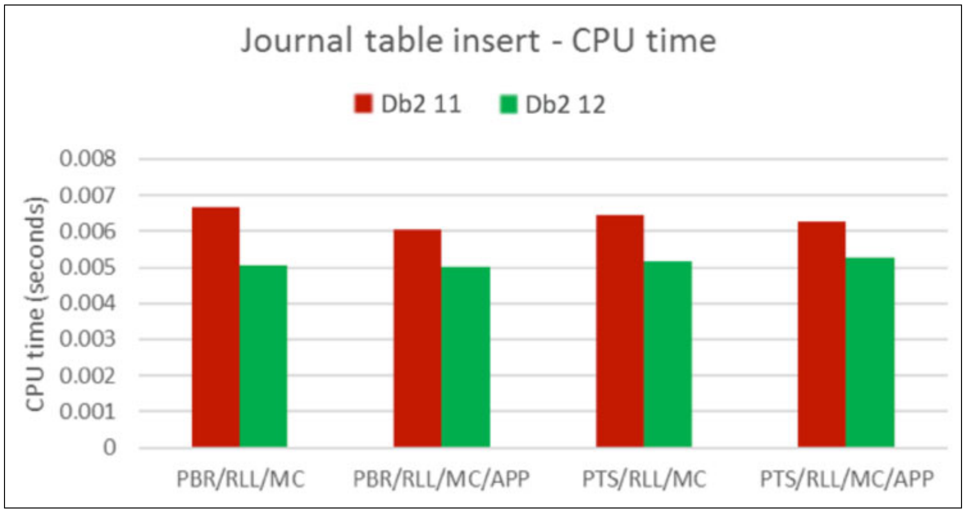


Figure 5-17 Journal table insert Db2 class 2 CPU time

In addition, the fast index traversal feature reduced the total number of index getpages by 38%. Insert performance was not affected by the APPEND option, as shown in Figure 5-17. Almost no class 2 CPU time difference was observed between tests that used MEMBER CLUSTER with and without APPEND.

The throughput results that are shown in Figure 5-18 on page 183 show that no significant change was observed in insert external throughput rate between Db2 12 and Db2 11. The wait time reduction from page latch removal and log write latch removal has no positive effect on the throughput rate. This result is due to the other bottlenecks in the application, network, and contention that is caused by index split wait.

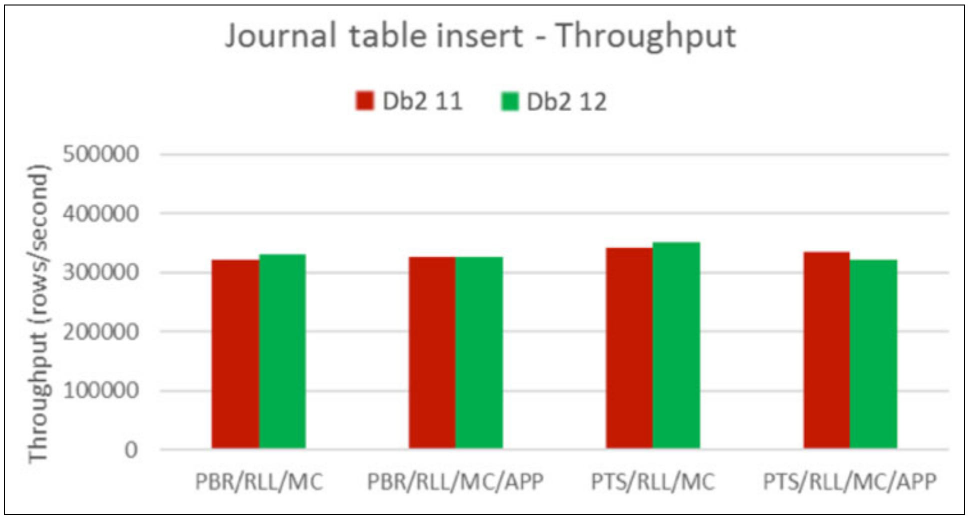


Figure 5-18 Journal table insert external throughput rate

5.6.3 Message queue insert and delete

This test scenario simulates a message queue management application. The Db2 table is treated as a queue. A task inserts some rows sequentially into this queue table, while it also starts to process and delete rows at the front of the queue table. This scenario features the following configuration:

- A PBG table space
- Row level locking
- Multi-row insert with 10 rows per commit
- A one-way and two-way Db2 data sharing group

The CPU time results that are shown in Figure 5-19 show a Db2 class 2 CPU time improvement of 6% for the one-way and two-way benchmarks in Db2 12 compared to Db2 11. Log write latch contention (LC19) removal and buffer manager timer latch (LC23) removal are the biggest contributors to the class 2 CPU time reduction.

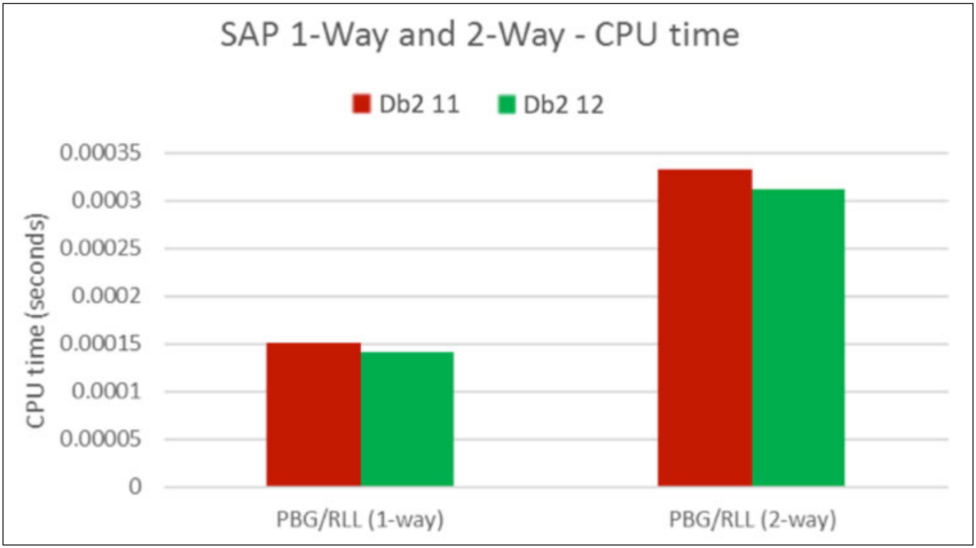


Figure 5-19 Message queue insert and delete SAP 1-way and 2-way Class 2 CPU time

The external throughput results that are shown in Figure 5-20 show that no significant difference was observed in insert throughput between Db2 11 and Db2 12. As the CPU time is not the biggest component of the transaction elapsed time, the 6% CPU time improvement did not result in a significant increase in throughput rate.

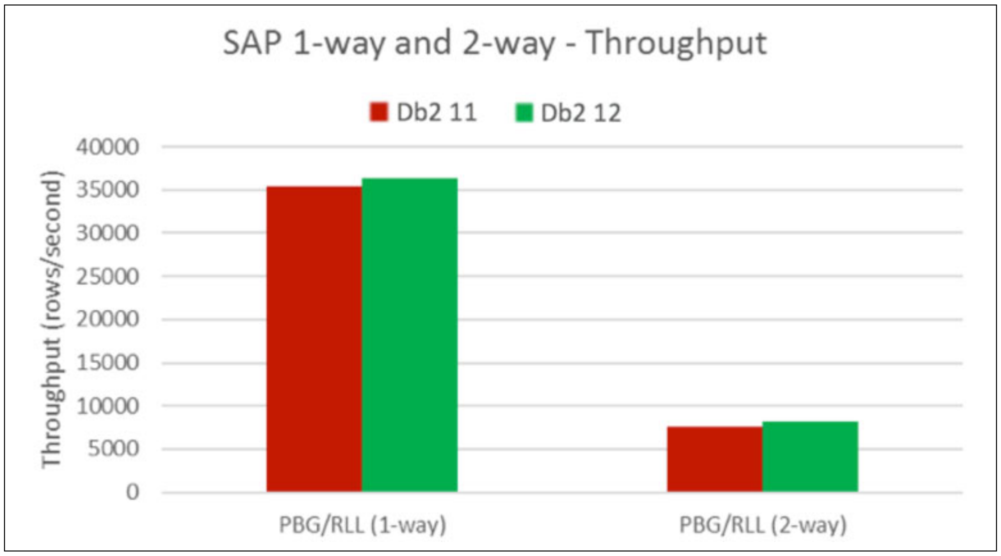


Figure 5-20 Message queue insert and delete SAP 1-way and 2-way insert throughput

5.6.4 Random insert and delete workload

This workload scenario uses an application with a random insert and delete pattern. The benchmark setup uses the following configuration:

- Three types of tables spaces:
  - Classic PTS
  - PBG
  - PBR
- Page level locking (PLL)
- One table and one unique clustering index
- 100 partitions for the PTS and PBR case
- A two-way data sharing group that is concurrently running 180 threads, which are performing inserts and 120 threads executing deletes

During the test runs, the table is first seeded with 40 million rows. Then, 4 million rows are randomly inserted. At the same time, 2.7 million rows are randomly deleted. The inserts are done concurrently by 180 threads and the deletes are done concurrently by 120 threads that re spread across the two Db2 data sharing members.

The Db2 class 2 CPU time results, as shown in Figure 5-21, show the following results:

- For random insert, a 4% improvement was achieved for PBG and PBR table spaces in Db2 12 compared to Db2 11. For PTS table spaces, no notable difference was observed between the Db2 12 and Db2 11 test runs.
- For random delete, significant (up to 18%) CPU improvements were observed during the Db2 12 tests for all three types of table spaces compared to their corresponding Db2 11 tests. The significant CPU improvement comes from the use of the fast index traversal feature, which allows a total getpage reduction of up to 40%.

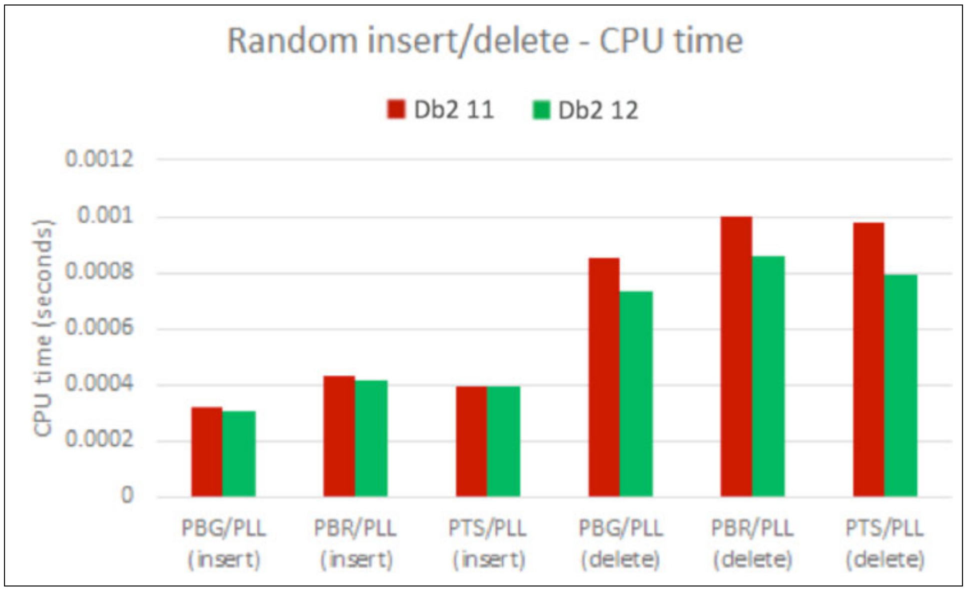


Figure 5-21 Random insert/delete Class 2 CPU time

The throughput results that are shown in Figure 5-22 show that the Db2 11 and Db2 12 runs resulted in similar insert and delete throughput rates without any significant differences between the two Db2 versions for any type of table space. As in the example that is described in 5.6.3, “Message queue insert and delete” on page 183, these transactions are not CPU bound; therefore, no significant increase in throughput rate is expected.

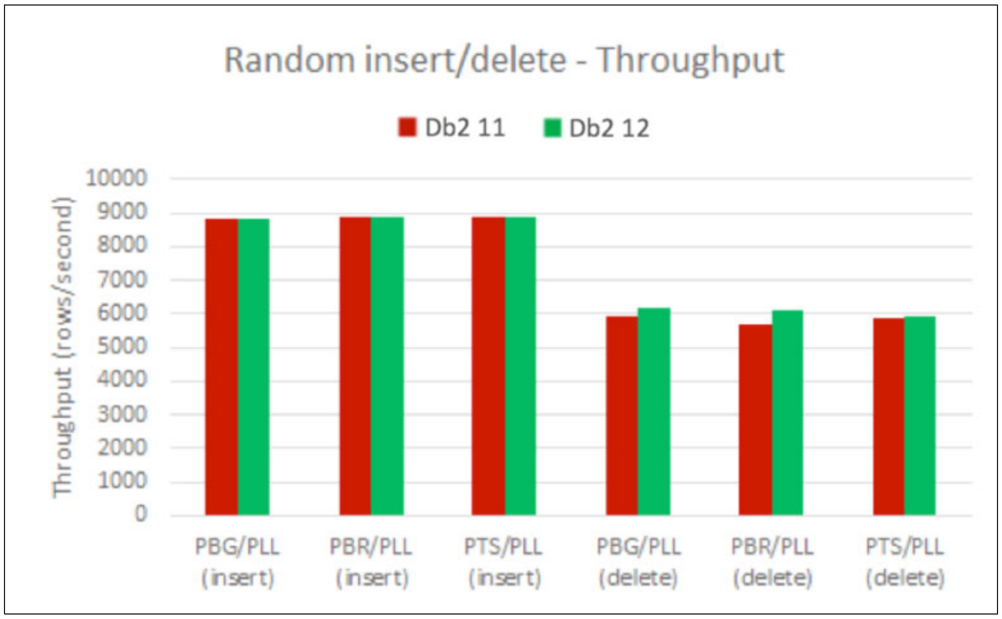


Figure 5-22 Random insert/delete throughput rate

6

Distributed system

Db2 12 provides various enhancements that improve its interactions with distributed systems. These enhancements include functionality, such as DRDA Fast Load and support for the z/OS Connect Feature Pack.

The chapter includes the following topics:

Tip: For more information about the topics that are described in this chapter, see the “Connectivity and administration routines” chapter of *IBM DB2 12 for z/OS Technical Overview*, SG24-8383, which also features the following topics:

- Maintaining session data on the target server
- Preserving prepared dynamic statements after a ROLLBACK
- Profile monitoring for remote threads and connections
- Stored procedures that are supplied by Db2

6.1 DRDA Fast Load

Often, users must load data that is in distributed clients into Db2 for z/OS. This process often requires transporting the data to the z/OS server before the data is loaded. The Db2 provided stored procedure DSNUTILU enables users to run Db2 utilities from a Db2 application program, including remote clients.

The typical process involves transferring the input file to a z/OS LPAR, which calls DSNUTILU from a client to start the LOAD utility and directs it to the transferred file. Finally, the data is loaded into Db2 for z/OS tables. This process complex and time-consuming.

Db2 12 for z/OS introduces DRDA Fast Load, which enables quick and easy loading of data from files that are on distributed clients. This new feature eliminates the need to transfer the data to z/OS before loading.

The Db2 Call Level Interface (CLI) APIs and command line processor (CLP) were enhanced to support remote loading of data to Db2 for z/OS. This new feature is available in all Db2 v11.1 FP1 and later client packages and Java applications.

DRDA Fast Load supports the following file formats:

- DELIMITED (DEL)

  A sequential file that features row and column delimiters. EBCDIC, ASCII, and UNICODE are supported.
- INTERNAL (INT)

  A file that is created by running Db2 for z/OS UNLOAD by using the INTERNAL FORMAT option. LOAD does not validate the data to ensure Db2 internal format. It can be created by a client application (such as SAP), but the internal format must be respected and validated.
- SPANNED (SPN)

  A sequential file for allowing the loading of tables with LOB/XML columns. The user must build their spanned format data stream or file, where each record is preceded with an 8-byte integer that includes the length of stream for a row.

The Db2 for z/OS stored procedure DSNUTILU now supports a new INDDN keyword, SYSCLIEN, for the LOAD utility control statement.

The Db2 CLP now supports the zload keyword, as shown in Example 6-1.

Example 6-1 CLP zload syntax example

```
ZLOAD FROM <filename> OF {INT | SPANNED | DEL [ALF | ENL | CRLF]}[WITH
<utilityid>] [MESSAGES <msg.filename>] USING <loadstmt>
```

Performance tests show that DRDA Fast Load performs better than LOAD alone, with a significant reduction in elapsed time.

The task that extracts data blocks and passes them to the LOAD utility is zIIP eligible and 100% of the CPU can be offloaded to zIIP specialty engines. This task is run by using the WLM goal of the Db2 distributed address space service class.

Note: Db2 Fast Load requires Db2 for Linux, UNIX, and Windows clients Version 11.1 Fix Pack 1 or above. Db2 clients are available at the Download Fix Packs by version for IBM Data Server Client Packages website.

6.1.1 Db2 Fast Load measurements

This section describes the results of the performance observations when comparing DRDA Fast Load with the local Db2 LOAD utility.

The test environment for the workload used the following configuration:

- z13 z/OS LPAR with 17 dedicated general CPs, 32 GB of memory, and z/OS 2.02.
- Linux for System z on z13 LPAR with 10 general CPs with Db2 Connect and Db2 for Linux, UNIX, and Windows 11.1 FP1.
- IBM HiperSockets communication between the z/OS and Linux for System z LPARs.

The scenarios compare the performance of the traditional FTP plus local LOAD process versus loading the data directly from the client by using DRDA Fast Load.

The test cases consist of loading data from 160 data files into a Db2 table that was created in a table space that is defined with 160 partitions. On the client side, 160 parallel scripts load the data into each partition.

Example 6-2 show the bash scripts that were used for driving the parallel **`zload`** commands. In this script, the `zload12a` file is started to run the **`zload`** command.

Example 6-2 Bash script to drive 160 parallel DRDA Fast Load script

```
#!/bin/bash
let j=1
while [ [ $j -lt 160 ] ]
do
  ( zload12a $j $j.time) >& $j.log &
   let j=j+1
done
```

The contents of the `zload12a` file is shown in Example 6-3.

Example 6-3 zload command to load a single partition out of 160 partitions

```
db2 zload from DB2DRDA.ZLOADB.ITRADE.P000$1  with "zload$1"  messages
zloadINT.p$1.txt using "TEMPLATE SORTIN DSN DISTRGR.ZLOADI$1.T&TIME.  UNIT 3390
SPACE(100,40) CYL VOLUMES(H210BE) DISP(NEW,DELETE,DELETE) BUFNO(20) TEMPLATE
SORTOUT DSN DISTRGR.DC1X.ZLOADO$1.T&TIME.  UNIT 3390 SPACE(100,40) CYL
VOLUMES(H210BD)  DISP(NEW,DELETE,DELETE)  LOAD DATA INDDN SYSCLIEN
WORKDDN(SORTIN,SORTOUT) LOG(NO) NOCOPYPEND FORMAT INTERNAL  INDEXDEFER NPI INTO
TABLE USRT001.TRADE PART $1 REPLACE PREFORMAT REUSE NUMRECS 11138614
"
```

Each file consists of data in internal format to be loaded into each partition.

Scalability scenario

In the scalability scenario, 15, 30, and 60 concurrent LOADs were run to measure the scalability of the DRDA Fast Load. The loads were run from a client machine against different partitions of the same table in Db2 for z/OS. The scenario consisted of loading 15 internal format files into 15 partitions, 30 files into 30 partitions, and 60 files into 60 partitions. The table was empty before the load.

Figure 6-1 shows the LOAD rate in MBps versus the number of concurrent zload tasks. The results indicate that LOAD performance increases with the number of parallel zload runs. If enough hardware resources are available, running parallel zload tasks increases LOAD throughput.

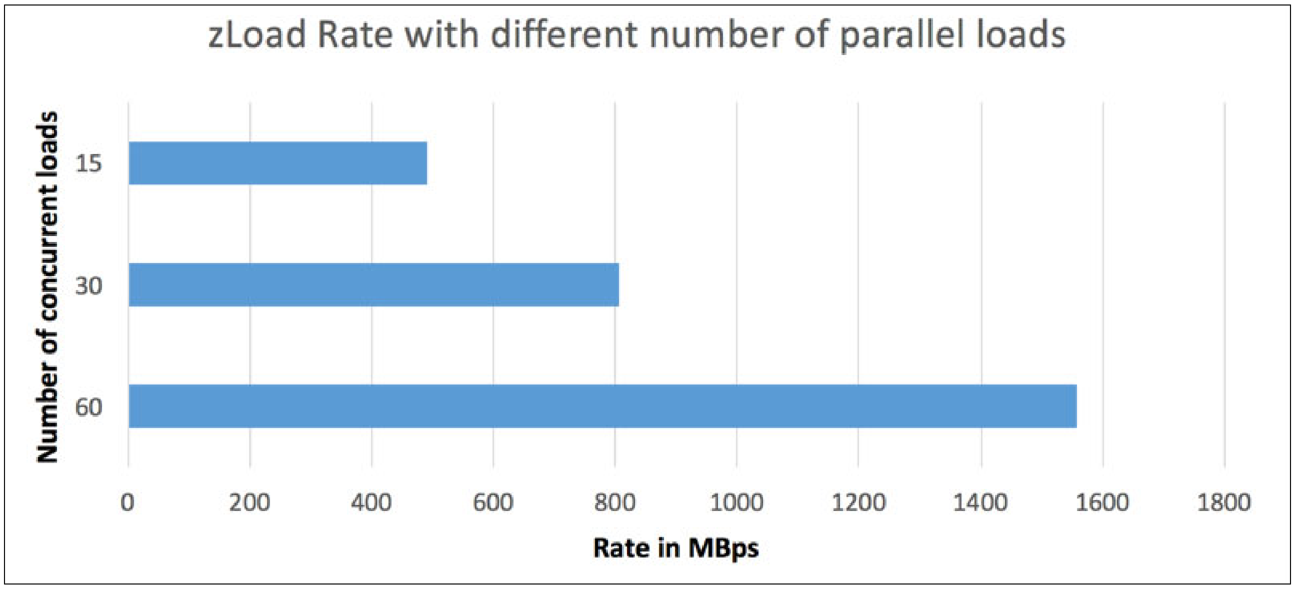


Figure 6-1 zload scalability increased LOAD throughput with number of parallel zload tasks

Load methods comparison scenario

This set of measurements are taken to demonstrate the performance comparison of the following techniques when loading data in internal format into a single table partition in Db2 for z/OS:

- DRDA Fast Load
- File transfer to z/OS (by using FTP) followed by local LOAD
- Db2 for Linux, UNIX, and Windows Client import utility

The Db2 for Linux, UNIX, and Windows command import utility inserts data from an external file with a supported file format into a table, hierarchy, view, or nickname. The import command also is available as part of the Db2 for Linux, UNIX, and Windows clients.

Example 6-4 shows the import syntax that is used during the test scenario.

Example 6-4 Import command that is used in test scenarios

```
“import from part1.ixf of IXF "modified by compound=100 commitcount 10"  insert
into USER001.TRADE"
```

The following options in the “modified by” section are necessary for performance:

- `compound=x`: x is a number 1 - 100, inclusive. It uses non-atomic compound SQL to insert the data, and x statements are attempted each time.
- commitcount n: Performs a COMMIT after every *n* records that are imported.

Figure 6-2 shows the results of the measurements for this scenario.

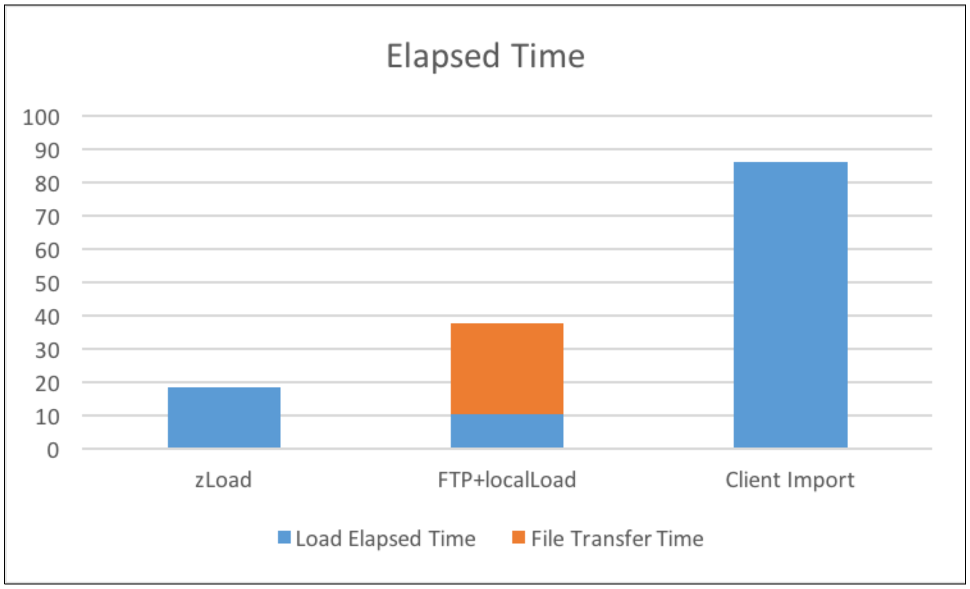


Figure 6-2 Load method comparing zload, FTP+load, and client import

Db2 Fast Load takes approximately 18 seconds to complete loading the data, while the import utility takes 86 seconds. In the FTP combined with local load scenario, the local load takes 10 seconds to load after the data is on the host. FTP takes 27.27 seconds, which results in a total elapsed time of 37.6 seconds.

The measurements show that Db2 Fast load is approximately two times faster than FTP and load, and roughly 4.7 times faster than the import utility.

The number of rows and the size of the data loaded per table that were used for the comparison of zload, FTP+Local load, and import is listed in Table 6-1.

Table 6-1 Data size and number of rows

Table	**CUSTOMER**	**ORDERLINE**
Number of rows	12000000	120004652
Size (Bytes)	6803777316	8520330292

The Db2 accounting data, which is extracted from Omegamon PE reports, shows the performance benefits of DRDA Fast Load compared to Db2 LOAD. The data indicates that Db2 Fast Load reduces total elapsed time by 25% to 33%, while Class 1 CPU time is reduced by approximately 2%. The results are listed in Table 6-2.

Table 6-2 Local LOAD versus DRDA Fast Load, in seconds

	Local LOAD		**DRDA Fast Load**		**Delta (%)**	
	CUSTOMER	ORDERLINE	CUSTOMER	ORDERLINE	CUSTOMER	ORDERLINE
DDF						
CL1 ET			81.482367	119.525887		
CL2 ET			0.304971	0.126698		
CL1 CP			0.005357	0.005341		
CL2 CP			0.003641	0.00362		
UTILITY						
CL1 ET	56.359761	85.828302	80.951364	119.153029	43.63	38.83
CL2 ET	33.219679	77.26465	33.2303	60.840731		
CL1 CP	29.479038	50.894291	28.818106	49.689923	-2.24	-2.37
CL2 CP	27.410148	46.861883	27.514255	46.733212	0.38	-0.27
FTP TIME	65.394	73.195				
Total elapsed time	121.75	159.02	81.48	119.53	-33.08	-24.84
Rate (MBps)	55.88	53.58	83.50	71.28	49.42	33.05

The Db2 Fast Load utility uses DDF and runs in an enclave SRB. Also, this CPU processing is eligible for zIIP offload.

The LPAR that is used to run the tests did not have zIIP processors available. Even so, the Db2 accounting traces can be used to report the accumulated CPU time that is used on a standard CP for work that is eligible on an IBM specialty engine.

A portion of an Omegamon PE report for running the DRDA Fast Load scenario is shown in Example 6-5.

Example 6-5 Import command used in test scenarios

```
MEASURED/ELIG TIMES     APPL (CL1)
 -------------------    ----------------
 ELAPSED TIME           0.980277
  ELIGIBLE FOR ACCEL    N/A

 CP CPU TIME            0.128805
  ELIGIBLE FOR SECP     0.113854
```

For this scenario, the CPU time that is eligible for specialty engine is 0.113854 seconds. The total CPU time is 0.128805 seconds, which results in an 88% CPU offload eligible ratio.

6.2 Integrated Db2 support of REST APIs

Db2 for z/OS V12 provides a set of Db2 Representational State Transfer (REST) services with which you can create, manage, discover, and run user-defined REST services in Db2 without requiring the Db2 Adapter for z/OS Connect.

You can start Db2 REST services with any RESTful API client, including the IBM z/OS Connect Enterprise Edition (EE) V2.0 or IBM's Cloud-based API Management solution. These new services extend the value of applications that access Db2 data by allowing a more efficient and scalable API for mobile and cloud applications without the burden of installing more products or configuring a new network interface.

The new services use the existing distributed data facility (DDF) capability. It provides the same class of the following fully integrated Db2 facilities services you expect when DDF is used to access Db2 data:

- Authentication
- Authorization
- Client information
- Enclave classification
- System profiling
- Sysplex awareness
- Accounting
- Statistics
- Displays
- Monitoring

Db2 Adapter for z/OS Connect, which was delivered in 2015, is a service provider that connects REST web, mobile, and cloud applications to Db2 z/OS. For more information about Db2 Adapter for z/OS Connect, see 6.3, "z/OS Connect Feature Pack support".

6.3 z/OS Connect Feature Pack support

WebSphere Liberty for z/OS Connect is software that enables a z/OS system, such as CICS, IMS, or Db2 for z/OS, to participate in today's mobile computing environment. It runs inside WebSphere Liberty Profile for z/OS and provides an interface between mobile and cloud devices and backend systems. It provides RESTful APIs, accepts JSON data payloads, and communicates with backend systems for data and transactions.

To support WebSphere Liberty z/OS connect, Db2 for z/OS provides a Db2 Adapter for z/OS Connect through the z/OS Connect Feature Pack.

This comprehensive solution allows users to discover, model, enable, and publish RESTful services on trusted Db2 z/OS systems. These web services can be used by mobile, web, or cloud applications that can access Db2 z/OS systems in secure, managed, and optimized ways.

Db2 Adapter is an integrated solution into the IBM Liberty Profile for z/OS Connect feature. Db2 Adapter for z/OS is an abstract resource that is represented by a simple SQL statement. The statement can be a query statement, such as a select or update statement, or a call statement to a stored procedure.

The Db2 Adapter for z/OS Connect is being deprecated and the functionality integrated into Db2 DDF and managed as a native service. Db2 as a service provider will continue to allow the invocation of a service by using the Db2 Adapter invocation API format after the service is created in Db2 to minimize application disruptions and to ease migration.

An overview of the WebSphere Liberty z/OS Connect architecture is shown in Figure 6-3.

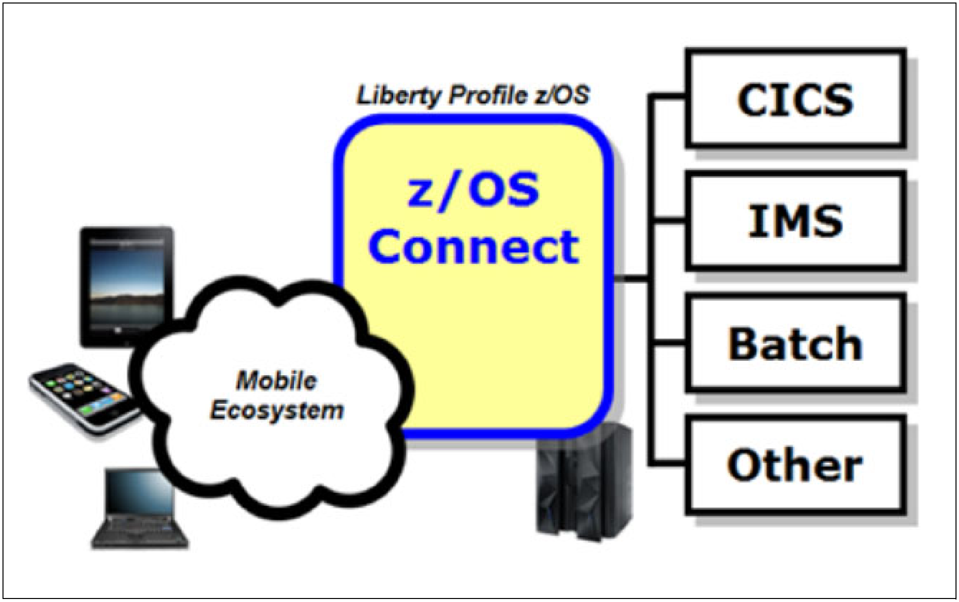


Figure 6-3 WebSphere Liberty z/OS Connect architecture

Note: For more information about IBM WebSphere Liberty z/OS Connect, see the IBM WebSphere Liberty z/OS Connect web page.

6.3.1 Db2 Adapter for z/OS Connect workload measurements

This section describes the performance measurements of Db2 Adapter for z/OS Connect workloads. The purpose of these measurements is to document the performance expectations for a Db2 z/OS with a Db2 Adapter for z/OS Connect workload.

Workload description

The RESTful service contains a single select statement on a table that is in Db2 z/OS, and the select statement always returns one row. The workload consisted of 8 groups of clients with 30 concurrent users (each for a total of 240 users maximum), each sending an HTTPS request to the Db2 Adapter for z/OS Connect server. For each measurement scenario, the number of clients in the group is increased to understand the scalability of the Db2 Adapter for z/OS Connect services.

Measurement environment

All measurements were done by using a one-tier environment, where WebSphere Liberty profile server and Db2 z/OS are running on the same physical LPAR. Four LPARs on the same physical machine were used with two WebSphere Liberty Servers on each LPAR. The following hardware and software configuration was used in the measurements:

- Each z13 (CPU model NE1) LPAR with 20 CPUs and 32 GB of real memory
- z/OS level 2.01.00

- WebSphere Liberty version 8.5.5.5/wlp-1.0.8.cl50520150305-2202
- Java 7 64 bit -pmz6470sr8-20141026_01 (SR8)
- db2zAdaptor GA build- bundle version 1.0.0.201511181332
- db2zAdaptor data source that uses type 2 JDBC driver
- Eight-way data sharing group with two Db2 members on each LPAR

Measurement results

The workload was run first with 60, 120, and 240 concurrent clients. No changes were made to the server configuration between measurements.

External Throughput Rate (ETR) is the throughput rate of the workload in transactions per second. The ETR results for each run with the number of clients doubling for each measurement are shown in Figure 6-4.

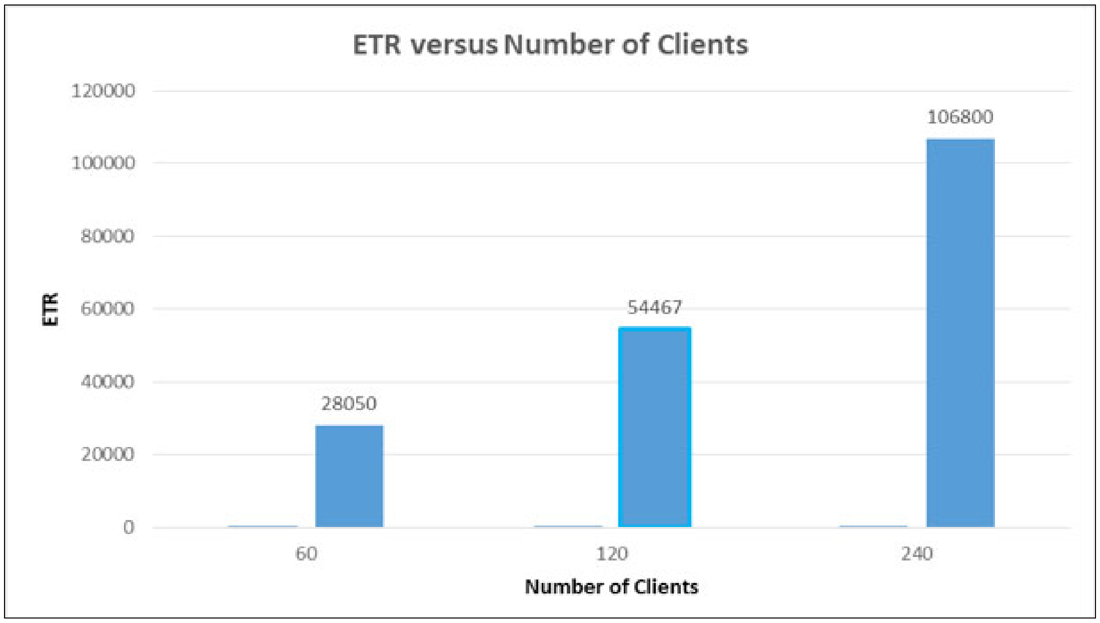


Figure 6-4 Observed ETR versus number of clients

The data shows that the total throughput increased linearly with the number of concurrent users. Also, the Db2 Adapter for z/OS achieves a rate of 106,800 transactions per second when running with 240 concurrent users, each sending REST requests.

CPU time eligible for Special Engine CP results

The CPU time that is eligible for specialty engine of our scenarios is listed in Table 6-3. Data shows that the percentage that is eligible for SECP is approximately 50%.

Table 6-3 Percent of eligible SECP of the scenarios

Number of clients	**60**	**160**	**240**
CP CPU time	0.000256	0.000268	0.000270
Eligible for SECP	0.000147	0.000127	0.000124
Percentage eligible for SECP	57.42	47.39	45.93

7 SQL and application enablement

Db2 12 offers various enhancements for SQL and application enablement. These enhancements include SQL support enhancements, such as advanced triggers and prepare avoidance after a rollback. Enhancements also are featured in other areas, such as JSON support, referential integrity, and XML.

This chapter includes the following topics:

7.1 SQL support

Db2 12 offers the following enhancements that pertain to SQL support:

- Advanced trigger support
- Pagination
- MERGE statement
- Non-deterministic expression columns for auditing
- KEEPDYNAMIC (YES) after ROLLBACK
- Dependency Manager Hash (DMH) table
- Unicode column in an EBCDIC table
- Piece-wise data modification

Tip: For more information about the SQL Support enhancements and new features in Db2 12 for z/OS, see *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

7.1.1 Advanced trigger support

Db2 12 for z/OS enhances the support for triggers. Application logic in the form of SQL PL can be specified in the trigger body starting in Db2 12 Function Level 500.

These new triggers are called *advanced triggers*, as opposed to the *basic triggers* that are supported by older releases. The SQL PL support in triggers makes it easier to port triggers from the Db2 family or competitive database products. The extensions also allow a user to debug and maintain multiple versions of a trigger.

Triggers that use the same pre Db2 12 syntax that is defined in Db2 12 for z/OS results in the creation of a basic trigger with unchanged performance characteristics if SQL PL is not specified in the trigger body. However, if you want SQL PL support in the trigger body, you must create advanced triggers by omitting the MODE DB2SQL option.

Note: Advanced triggers are created only when SQL PL support is needed because, with the new debugging and versioning features that are associated with the advanced triggers, performance degradation in CPU can result, even if a basic trigger is accidentally defined as an advanced trigger by removing the MODE DB2SQL option.

When you create an advanced trigger in Db2 12 for z/OS, its procedural statements are converted to a representation that is stored in the database directory, as is done with other SQL statements. The options are stored in the database catalog tables as in older releases. When one of these advanced triggers is activated, the internal representation is loaded from the directory and the Db2 engine runs the trigger body.

Execution performance of basic triggers versus advanced triggers

To support more flexible and robust features in advanced triggers in Db2 12, such as debugging and versioning, you see certain execution overhead when running advanced trigger compared to existing basic triggers with the same logic. To understand the performance affect, the measurements are executed by running the same simple triggers without contain SQL PL logic in basic and advanced triggers.

The CPU increase that you might experience (depending on the type of trigger even if it does not contain SQL PL logic) is listed in Table 7-1.

Table 7-1 Basic and advanced trigger performance comparison

Trigger body	**Db2 12 for z/OS Class 2 Trigger CPU (basic trigger)**	**Db2 12 for z/OS Class 2 Trigger CPU (advanced trigger)**	**Change**
SIGNAL SQLSTATE '75001' ('Test SQLPL trigger');	0.034678	0.038890	12.14%
INSERT INTO TB002 (C1) VALUES (NEW.PID+1);	0.059801	0.073879	23.54%
UPDATE TB002 SET C1 = NEW.PID + C1 WHERE C1 > 0;	0.111511	0.128181	14.95%
DELETE FROM TB002 WHERE C1 = NEW.PID;	0.075580	0.092044	21.78%
SET new.COL1 = (CURRENT TIMESTAMP - CURRENT TIMEZONE), new.PID = 1 * 4; set two transitional variables with WHEN clause	0.066693	0.069040	3.52%
CALL SQLPL001(); call a stored procedure	0.039684	0.059965	51.11%
VALUES (COUNTER_PLUS(new.PID)); values (inline function)	0.048273	0.064197	32.99%
SET new.COL1 = (SELECT IBMREQD FROM SYSIBM.SYSDUMMY1); set one transitional variable	0.085933	0.101520	18.14%
SELECT IBMREQD FROM SYSIBM.SYSDUMMY1; full select	0.080004	0.154533	93.16%

To avoid the performance penalty for the simple trigger statements, MODE DB2SQL should be kept from the existing trigger as Db2 uses the MODE DB2SQL option to determine the types of trigger it generates. Db2 creates basis triggers by using the MODE DB2SQL option. Without MODE DB2SQL, Db2 creates advanced triggers.

If you migrate from Db2 11 for z/OS to 12 without dropping and re-creating the triggers, they remain as basic triggers and maintain the same performance characteristics. The APPLCOMPAT column in SYSIBM.SYSTRIGGERS remains V11R1 for those migrated triggers. If the trigger body contains SQL PL logic, an advanced trigger is created, even if you specify MODE DB2SQL in Db2 12 for z/OS.

If the default option MODE DB2SQL is specified, the basic trigger uses the old code path. Otherwise, the trigger is created as an advanced trigger that uses the new code path. Basic triggers that are defined in Db2 12 include a value of V12R1 in the APPLCOMPAT column in the SYSIBM.SYSTRIGGERS catalog table and a value of “T”, which represents an old-style trigger in the TYPE column of table SYSIBM.SYSPACKAGE (a value of “1” identifies a trigger as an advanced trigger).

Another way to identify a basic trigger is by using the SQLPL column in the SYSIBM.SYSTRIGGERS catalog table. A value of “N” means that a basic trigger was created. A value of “Y” means that an advanced trigger was created (even if there might not be any SQL PL logic in the trigger body).

Tip: If your trigger does not need SQL PL logic, define them by using the MODE DB2SQL option in Db2 for z/OS 12 to avoid any performance degradation.

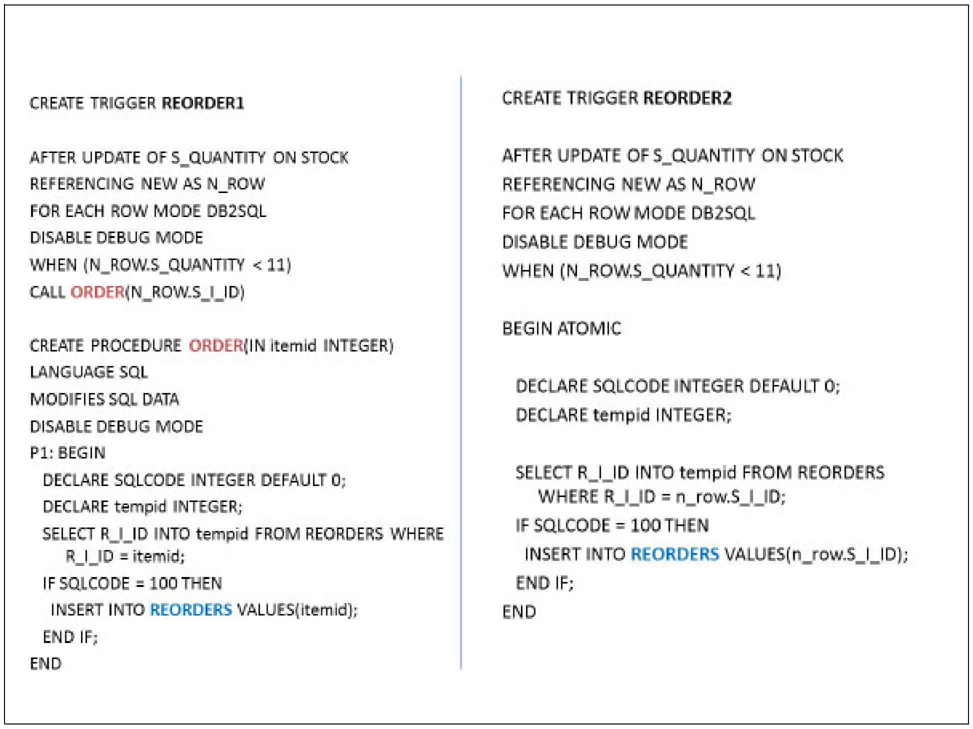
Performance of using an advanced trigger versus a basic trigger calling a stored procedure

Before the advanced trigger support that is provided in Db2 12, when complex logic of some kind was needed to accomplish a certain task that couldn't be done with simple triggers, often times using stored procedures was the solution to achieve the goal.

With the new advanced trigger support, the logic that was within the stored procedure can now be embedded inside the trigger.

Consolidating a basic trigger that is calling a stored procedure into a new advanced trigger

Sample code for a trigger that is calling a stored procedure (on the left), and a “consolidated” advanced trigger that contains the logic that was in the stored procedure inside the trigger body (on the right) is shown in Figure 7-1.



```
CREATE TRIGGER REORDER1

AFTER UPDATE OF S_QUANTITY ON STOCK
REFERENCING NEW AS N_ROW
FOR EACH ROW MODE DB2SQL
DISABLE DEBUG MODE
WHEN (N_ROW.S_QUANTITY < 11)
CALL ORDER(N_ROW.S_I_ID)

CREATE PROCEDURE ORDER(IN itemid INTEGER)
LANGUAGE SQL
MODIFIES SQL DATA
DISABLE DEBUG MODE
P1: BEGIN
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE tempid INTEGER;
  SELECT R_I_ID INTO tempid FROM REORDERS WHERE
      R_I_ID = itemid;
  IF SQLCODE = 100 THEN
    INSERT INTO REORDERS VALUES(itemid);
  END IF;
END
```

```
CREATE TRIGGER REORDER2

AFTER UPDATE OF S_QUANTITY ON STOCK
REFERENCING NEW AS N_ROW
FOR EACH ROW MODE DB2SQL
DISABLE DEBUG MODE
WHEN (N_ROW.S_QUANTITY < 11)

BEGIN ATOMIC

  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE tempid INTEGER;

  SELECT R_I_ID INTO tempid FROM REORDERS
     WHERE R_I_ID = n_row.S_I_ID;
  IF SQLCODE = 100 THEN
    INSERT INTO REORDERS VALUES(n_row.S_I_ID);
  END IF;
END
```

Figure 7-1 Consolidating a stored procedure calling trigger into an advanced trigger

Performance measurement results

Both triggers that are shown in Figure 7-1 were started by repeatedly updating rows of a table. Db2 accounting and statistics reports were collected to analyze the Db2 CPU, elapsed time, and CPU costs in both of the above-mentioned scenarios.

The following measurement results were obtained:

- No noticeable Db2 CPU reduction from the use of a consolidated trigger (<1%).
- Approximately 1 - 2% improvement in throughput.

Advanced triggers use more CPU than the (pre-V12) basic triggers, which might be responsible for not seeing a resultant CPU benefit from this scenario.

Note: The main benefit of the use of an advanced trigger for this purpose is the reduced complexity and maintenance of the Db2 environment. This use results in only a single trigger that must be maintained instead of maintaining a trigger and a stored procedure.

7.1.2 Pagination

With the increasing demands for mobile and web applications, application developers often want to display a subset of rows back to the user. Displaying a subset of rows at a time allows the user to "page" through the data. Db2 12 for z/OS introduces the following types of pagination support:

- Data-dependent
- Numeric-based

Data-dependent pagination

Data-dependent pagination is a method whereby an application developer can display pages of data at a time. The data that is returned in the last row of the page is the data that is used to fetch the next page of data. Application developers code this style of pagination in their applications and return pages of data until no other rows qualify.

The following key features ensure data-dependent pagination returns the correct data:

- All of the columns that are specified in the ORDER BY clause must be in ascending or descending order.
- All of the columns that are combined (specified in the ORDER BY clause) must generate a unique value.

Before Db2 12, the disjunctive form was required when we wanted to obtain a result set that was based on a logical key value, as shown in Example 7-1.

Example 7-1 Disjunctive form example

```
SELECT L_SHIPDATE,L_COMMITDATE, L_RECEIPTDATE
   FROM  LINEITEM
   WHERE (L_SHIPDATE='1998-09-01' AND L_COMMITDATE='1998-09-01' AND L_RECEIPTDATE
='1998-09-01')
   OR (L_SHIPDATE='1998-09-01' AND L_COMMITDATE='1998-09-01' AND
   L_RECEIPTDATE >'1998-09-01')
   OR (L_SHIPDATE='1998-09-01' AND L_COMMITDATE > '1998-09-01')
   OR (L_SHIPDATE > '1998-09-01');
```

With data-dependent pagination, we can use a simpler syntax, as shown in Example 7-2.

Example 7-2 Data-dependent pagination example

```
SELECT L_SHIPDATE, L_COMMITDATE, L_RECEIPTDATE
   FROM LINEITEM
   WHERE (L_SHIPDATE,L_COMMITDATE,L_RECEIPTDATE)>=('1998-09-01', '1998-09-01',
'1998-09-01');
```

The performance of data-dependent pagination is similar to the performance of the same query that is written in disjunctive form. The advantage of data-dependent pagination is in the simplicity of its syntax.

Numeric-based pagination

Another form of pagination developers might employ in their applications is numeric-based pagination, in which data is returned to the user from a particular starting row and for a particular number of rows. The new OFFSET clause is used to return data in this manner (such as returning data from a particular starting row for a particular number of rows).

The offset of the beginning row is 0 (not 1) and the value of the offset must be 0 or positive. Before Db2 12, application developers might use other methods, such as a scrollable cursor, a rowset cursor, an OLAP function, a stored procedure, or application logic that skips the necessary number of rows before displaying the rows to the user. An example of the numeric-based pagination that uses the new OFFSET clause is shown in Example 7-3.

Example 7-3 Numeric-based pagination example

```
SELECT *
   FROM PART
WHERE P_PARTKEY > 50000
ORDER BY P_PARTKEY
OFFSET 101 ROW FETCH FIRST 100 ROWS ONLY;
```

If a user used a query to return the first 100 rows, the user can use the query that is shown in Example 7-3 to skip forward to row 101 by using the OFFSET clause to return the next 100 rows.

We compared the following scenarios:

- A developer issues a FETCH FIRST m+n ROWS query. The developer must then skip the first m-1 rows by using application logic.
- A developer issues an OFFSET m ROW FETCH FIRST n ROWS query. The developer does not need to do any further processing to skip the first m-1 rows because these rows are skipped already.

When comparing the above two scenarios we see that for the second case where the customer uses a query with OFFSET clause:

- The customer can see up to 85% CPU time reduction in cases where there is an index, and the OFFSET clause can be pushed down to Data Manager.

  Note: The OFFSET clause can only be pushed down if it is on a single base table (no join), without residual predicate or OLAP.

- The customer can see up to 98.5% reduction in Class 1 CPU and up to 94% reduction in Class 2 CPU time due to fewer rows being returned to the application.

Usage considerations

When trying to determine which pagination feature to use and how to use it, the following guidelines are suggested:

- When customer knows the data value of the row to start from, they should use data-dependent pagination because it generally performs better than offset pagination. However, if they do not have a particular value to page to, they can use offset pagination.
- If available, the customer should use a WHERE clause to get to the starting point of the data they are paging through and avoid large OFFSET values.
- Large OFFSET values are expensive and should be avoided because all skipped rows must still be scanned.
- Indexes help performance, which is especially true for pagination.

7.1.3 MERGE statement

Db2 9 for z/OS provided the initial support for the MERGE statement. The MERGE statement before Db2 12 allowed application developers to update rows in a table or insert new rows into a table, depending on the source values that were specified. If the source values match the target values, the rows are updated; otherwise, the new values are inserted into the target table. The source values can be an expression, a host-variable-array, or NULL.

Starting in Db2 12 Function Level 500, the MERGE statement is enhanced to allow the following functionality:

- Extra source value support. The source values can also be data from tables, data from views, or results from a full select.
- Extra data modification support, which introduced the delete operation if rows match the matching condition.
- More matching condition options.
- Support for more predicates on the matching conditions.
- Atomicity.

Performance improvement

In addition to enhancements, Db2 12 improves the performance of the MERGE statement through the optimization changes that are described in this section.

Sort avoidance

Because a record in the MERGE target might be matched with only one row with the MERGE source, the sort operation is needed to detect whether a row in the MERGE target was modified only once.

However, the sort operation might be avoided in some cases, such as when a unique index is defined on MERGE source and all index keys are referenced in a MERGE ON search condition clause.

Enable list prefetch

In Db2 12, MERGE uses the RID list access method to UPDATE or DELETE a row in the MERGE target. Therefore, if the RID of the next record that is being modified is in the buffer pool, the synchronous I/O time is reduced.

Performance evaluation and results

Starting in Db2 12 Function Level 500, the MERGE statement can use table reference as the MERGE source, but MERGE in Db2 11 or older releases cannot. These performance improvements are implemented in new Db2 12 MERGE usage.

The performance comparison is between an application and a single Db2 12 MERGE statement, which both implement the same function. However, the difference is that the application must define a cursor and corresponding host variables, fetch every row from the source table, and implement the programming logic.

In the scenarios that are described in this section, the MERGE statement uses table references as the source table. The source table and target table include the same column definition and the same unique index on the MERGE ON search condition column. The source table contains 450 million rows and the target table is empty at the beginning of the process.

Insert scenario

Because the target table is empty at first, the MERGE statement that is shown in Example 7-4 inserts all rows from the source table CUSTSOUR into the target table CUSTTARG. The UPDATE SET part is not operated.

Example 7-4 MERGE statement

```
EXEC SQL
MERGE INTO CUSTTARG AS T1
    USING CUSTSOUR AS T2
    ON T1.C_CUSTKEY = T2.C_CUSTKEY
  WHEN MATCHED
    THEN
      UPDATE SET T1.C_NAME = T2.C_NAME
  WHEN NOT MATCHED
    THEN
INSERT(C_CUSTKEY,C_NAME,C_ADDRESS,C_NATIONKEY,C_PHONE)

VALUES(C_CUSTKEY,C_NAME,C_ADDRESS,C_NATIONKEY,C_PHONE);
EXEC SQL COMMIT;
```

Part of the application that implements the same function as the MERGE statement is shown in Example 7-5.

Example 7-5 Application for insert scenario

```
EXEC SQL DECLARE C1 CURSOR WITH HOLD FOR
   SELECT C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE
   FROM
   CUSTSOUR;
EXEC SQL OPEN C1;
while(1)
   {
     EXEC SQL FETCH C1
     INTO :custkey,:name,:address,:nationkey,:phone;
     EXEC SQL
          SELECT 1 AS DUMMY INTO :dummy
          FROM CUSTTARG WHERE C_CUSTKEY = :custkey;
     if( sqlca.sqlcode == 100) /* if not exist, then insert*/
     {
         EXEC SQL
         INSERT INTO CUSTTARG
         (C_CUSTKEY,C_NAME,C_ADDRESS, C_NATIONKEY, C_PHONE)
         VALUES(
         :custkey,:name,:address,:nationkey,:phone);
     }
   }
   EXEC SQL COMMIT;
```

Compared to the application solution, the Db2 12 MERGE statement uses sort avoidance and enables list prefetch, which shows a 43% CPU reduction and a 36% elapsed time reduction for the insert scenario.

Update scenario

In this scenario, the target table features the same rows as the source table. In this situation, the same MERGE statement as shown in Example 7-5 updates the C_NAME column of the target table CUSTTARG to the C_NAME value of the source table CUSTSOUR.

Part of the application that implements the same function as the MERGE statement is shown in Example 7-6.

Example 7-6 Application for update scenario

```
EXEC SQL DECLARE C1 CURSOR WITH HOLD FOR
   SELECT C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE
   FROM
   CUSTSOUR;
EXEC SQL OPEN C1;
while(1)
   {
     EXEC SQL FETCH C1
     INTO :custkey,:name,:address,:nationkey,:phone;
     EXEC SQL
          SELECT 1 AS DUMMY INTO :dummy
          FROM CUSTTARG WHERE C_CUSTKEY = :custkey;
     if( sqlca.sqlcode == 0) /* if exist, then update*/
     {
         EXEC SQL
         UPDATE CUSTTARG SET C_NAME = :name
         WHERE  C_CUSTKEY =   :custkey;
     }
   }
   EXEC SQL COMMIT;
```

Compared to the application, the use of the MERGE statement shows a 53% reduction in CPU time and a 47% elapsed time reduction for the update scenario.

Update and insert scenario

In preparation for this scenario, all of the even rows in the target table are deleted. The same MERGE statement that is shown in Example 7-6 inserts all even rows into the target table, and updates the C_NAME column of odd rows.

A sample of the application that implements the same function as the MERGE statement (but without the use of MERGE) is shown in Example 7-7.

Example 7-7 Application for update and insert scenario

```
EXEC SQL DECLARE C1 CURSOR WITH HOLD FOR
   SELECT C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE
   FROM
   CUSTSOUR;
EXEC SQL OPEN C1;
while(1)
   {
     EXEC SQL FETCH C1
     INTO :custkey,:name,:address,:nationkey,:phone;
     EXEC SQL
          SELECT 1 AS DUMMY INTO :dummy
          FROM CUSTTARG WHERE C_CUSTKEY = :custkey;
     if( sqlca.sqlcode == 0) /* if exist, then update*/
     {
         EXEC SQL
         UPDATE CUSTTARG SET C_NAME = :name
         WHERE  C_CUSTKEY =   :custkey;
     }
     if( sqlca.sqlcode == 100) /* does not exist, then insert*/
     {
         EXEC SQL
         INSERT INTO CUSTTARG
         (C_CUSTKEY,C_NAME,C_ADDRESS, C_NATIONKEY, C_PHONE)
         VALUES(
         :custkey,:name,:address,:nationkey,:phone);
     }
   }
   EXEC SQL COMMIT;
```

Compared to the application, the MERGE statement shows a 46% CPU reduction and a 44% elapsed time reduction for the combined update and insert scenario.

Delete scenario

In this scenario, the target table is the same as the source table. The MERGE statement that is shown in Example 7-8 deletes all rows that are in the target table.

Example 7-8 MERGE statement

```
EXEC SQL
MERGE INTO CUSTTARG AS T1
    USING CUSTSOUR AS T2
    ON T1.C_CUSTKEY = T2.C_CUSTKEY
  WHEN MATCHED
    THEN
      DELETE
  WHEN NOT MATCHED
    THEN
INSERT(C_CUSTKEY,C_NAME,C_ADDRESS,C_NATIONKEY,C_PHONE)

VALUES(C_CUSTKEY,C_NAME,C_ADDRESS,C_NATIONKEY,C_PHONE);
EXEC SQL COMMIT;
```

Part of the application that implements the same function as the MERGE statement is shown in Example 7-9.

Example 7-9 Application for delete scenario

```
EXEC SQL DECLARE C1 CURSOR WITH HOLD FOR
   SELECT C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE
   FROM
   CUSTSOUR;
EXEC SQL OPEN C1;
while(1)
   {
     EXEC SQL FETCH C1
     INTO :custkey,:name,:address,:nationkey,:phone;
     EXEC SQL
          SELECT 1 AS DUMMY INTO :dummy
          FROM CUSTTARG WHERE C_CUSTKEY = :custkey;
     if( sqlca.sqlcode == 0) /* if exist, then delete*/
     {
         EXEC SQL
         DELETE CUSTTARG WHERE C_CUSTKEY=:custkey;
     }
   }
   EXEC SQL COMMIT;
```

Compared to the application, the MERGE statement shows a 52% CPU reduction and a 70% elapsed time reduction for the delete scenario.

7.1.4 Supporting KEEPDYNAMIC (YES) for ROLLBACK

The KEEPDYNAMIC BIND/REBIND option determines whether Db2 keeps dynamic SQL statements after the point of commit or rollback.

Before Db2 12 for z/OS, dynamic SQL statements are kept only in the local dynamic statement cache after COMMIT points if the KEEPDYNAMIC(YES) parameter was used.

However, if a transaction failed and resulted in a ROLLBACK rather than a COMMIT, the local copy of the dynamic statement is discarded, regardless of the KEEPDYNAMIC setting. Subsequent rerunnings of the same statement must be reprepared, or a SQLCODE of -518 is thrown.

Note: SQL code -518 indicates that `THE EXECUTE STATEMENT DOES NOT IDENTIFY A VALID PREPARED STATEMENT.`

This behavior forces application developers to program more complex logic. A programmer must code the logic that follows a ROLLBACK differently than the logic that follows a COMMIT.

Db2 12 for z/OS adds prepare avoidance support for ROLLBACK when KEEPDYNAMIC(YES) is specified. With the support, all the dynamic SQL statements can be prepared once and run many times, regardless whether the transaction succeeds.

When the global dynamic statement cache is enabled and the local dynamic statement cache is also turned on with KEEPDYNAMIC(YES), complete full and short prepare avoidance can be achieved even after a ROLLBACK With Db2 12 for z/OS.

This benefit can result in CPU savings for distributed applications that complete frequent ROLLBACKs.

Performance comparison before and after supporting KEEPDYNAMIC(YES) for ROLLBACK

CPU savings can be seen by comparing the Db2 accounting reports from before and after the inclusion of prepare avoidance support.

The necessary logic before prepare avoidance support is shown in Example 7-10.

Example 7-10 Program logic before Db2 12 for z/OS prepares avoidance support

```
PREPARE stmt1;
for(i=0; i<1000; i++)
{
EXECUTE stmt1;
if (sqlcode < 0)
{
record_error();
ROLLBACK;
PREPARE stmt1;  /* re-PREPARE after rollback is required */
}
else
{
COMMIT;
}
}
```

The necessary logic with prepare avoidance support is shown in Example 7-11.

Example 7-11 Program logic with prepare avoidance support

```
PREPARE stmt1;
for(i=0; i<1000; i++)
{
EXECUTE stmt1;
if (sqlcode < 0)
{
record_error();
ROLLBACK;
/* no need to re-PREPARE here */
}
else
{
COMMIT;
}
}
```

A performance comparison between before and after KEEPDYNAMIC(YES) Rollback Support scenarios is listed in Table 7-2.

Table 7-2 Performance Comparison Before and After KEEPDYNAMIC(YES) rollback support

	Before	**After**	**Difference**
CACHEDYN zparm	YES	YES	
KEEPDYNAMIC bind option	YES	YES	
Accounting Report Fields			
#ROLLBACKS	1000	1000	
NOT FOUND IN CACHE	0	0	
FOUND IN CACHE	1000	1	-999
IMPLICIT PREPARES	0	0	
PREPARES AVOIDED	0	999	999
Db2 CL.2 CP CPU TIME	1.778309	1.434686	-19.43%

These performance observations result from the best-case scenario, assuming all transactions fail with a ROLLBACK. Although such a case is unrealistic, the experiment was done to demonstrate the best benefit from this enhancement.

Transactions that end with a COMMIT reaped the CPU savings of prepare avoidance before this new feature. Performance savings can vary, depending on the percentage of transactions that result in ROLLBACKs and the short prepare cost in relation to the total cost for the transaction.

7.1.5 Improved storage usage for the Dependency Manager Hash (DMH) table

When a dynamic SQL statement is prepared, a copy of the statement is inserted into the global dynamic statement cache (DSC) if zparm CACHEDYN is set to YES. At the same time, Db2 object and authorization dependencies that are associated with that SQL are stored as entries in the so-called Dependency Manager Hash (DMH) table.

The DSC and DMH grow or contract as new dynamic SQL statements are prepared and added to the pool, or old statements are removed because of invalidation. If the EDM pool size reaches the threshold that is set by EDMSTMTC, Db2 starts the least recently used (LRU) algorithm to remove statements from the cache.

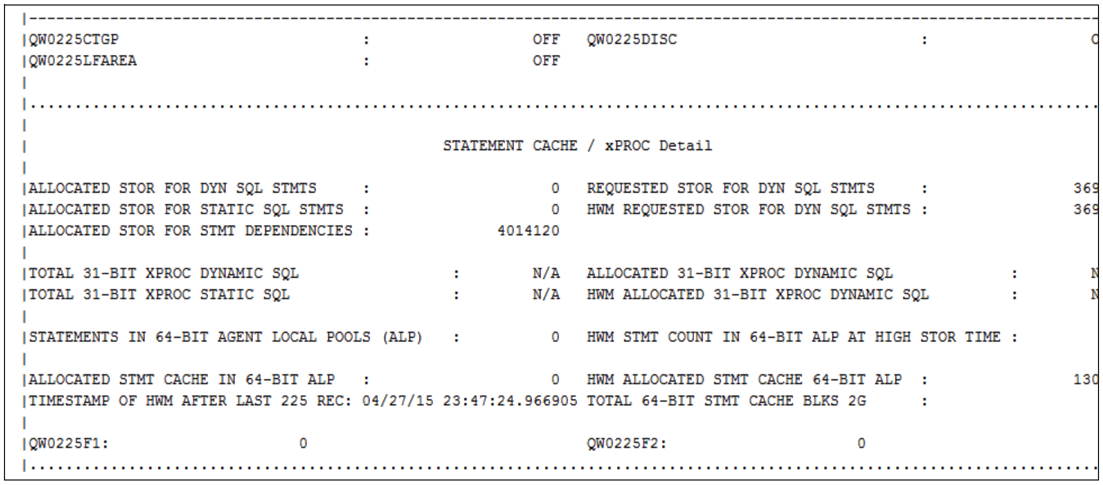
However, because of the inefficient storage usage of the DMH in the past, the DMH at times was full before the DSC. Then, the application received a resource unavailable SQLCODE of `-904 RC00E7910D`. Therefore, restarting Db2 to clear the DSC and the associated entries in the DMH was often the only solution.

Db2 12 introduces the following functional and performance enhancements for the DMH to significantly reduce the occurrences of DMH pool full conditions:

- The DMH table is now merged into the Statement Cache Blocks (SCB), which is another storage area within the DBM1 address space. A more efficient algorithm was adopted to reduce the overall storage consumption of the DMH, which makes it more in sync with the fluctuation of statements in the EDM dynamic statement cache.

- The zparm CACHE_DEP_TRACK_STOR_LIM that controls the upper limit of the DMH pool was deprecated. The zparm EDMSTMTC was expanded to not only set the soft limit for the EDM cache pool, but also for SCB pool because EDM and SCB are tied together through prepared dynamic statements and both include a size limit of 4 GB. The user can think of these two storage areas as one pool because now they are managed through one common knob.

  That section in IFCID 225, QW0225DMH ALLOCATED STOR FOR STMT DEPENDENCIES field, indicates the storage usage by DMH. Figure 7-2 shows a sample of an OMPE TRACE report, which shows the storage usage as reported by IFCID 225.



```
|------------------------------------------------------------------------------------------------------------------------
|QW0225CTGP                       :                  OFF   QW0225DISC                      :                C
|QW0225LFAREA                     :                  OFF
|
|........................................................................................................................
|
|                                              STATEMENT CACHE / xPROC Detail
|
|ALLOCATED STOR FOR DYN SQL STMTS     :                  0   REQUESTED STOR FOR DYN SQL STMTS     :              369
|ALLOCATED STOR FOR STATIC SQL STMTS  :                  0   HWM REQUESTED STOR FOR DYN SQL STMTS :              369
|ALLOCATED STOR FOR STMT DEPENDENCIES :            4014120
|
|TOTAL 31-BIT XPROC DYNAMIC SQL                :       N/A   ALLOCATED 31-BIT XPROC DYNAMIC SQL           :        N
|TOTAL 31-BIT XPROC STATIC SQL                 :       N/A   HWM ALLOCATED 31-BIT XPROC DYNAMIC SQL       :        N
|
|STATEMENTS IN 64-BIT AGENT LOCAL POOLS (ALP)  :         0   HWM STMT COUNT IN 64-BIT ALP AT HIGH STOR TIME :
|
|ALLOCATED STMT CACHE IN 64-BIT ALP   :                  0   HWM ALLOCATED STMT CACHE 64-BIT ALP  :              130
|TIMESTAMP OF HWM AFTER LAST 225 REC: 04/27/15 23:47:24.966905 TOTAL 64-BIT STMT CACHE BLKS 2G      :
|
|QW0225F1:                   0                             QW0225F2:                   0
|........................................................................................................................
```

Figure 7-2 Storage usage by DMH in IFCID 225

Performance observations

We ran our IBM Relational Warehouse Workload (IRWW) and periodically injected ALTER TABLESPACE for the table spaces the workload touches to trigger invalidation of cached dynamic SQL statements.

Through monitoring of the DBM1 storage consumption by using IFCID 225 records, we observed that the DMH storage did expand and contract as the number of cached statements in the DSC fluctuated, while it was relatively fixed and far less responsive with older versions of Db2 for z/OS.

Tip: By setting the zparm EDMSTMTC at a level below 80% of the 4 GB limit, Db2 intervenes much sooner and evicts statements from the DSC, which effectively minimizes the chances of the DMH becoming full.

7.1.6 Unicode column in EBCDIC table performance

Db2 11 for z/OS introduced the initial support for defining a Unicode column in an EBCDIC table.

However, the implementation was limited because the Unicode column was recorded as binary in the Db2 catalog, even though CHAR was specified as the column type in the CREATE TABLE statement. Along with the limitation came restrictions on the usability of such columns, such as indexability and string comparison.

Beginning with Db2 12 for z/OS with new function (Function Level 500) activated, a byte-based Unicode column can be defined in an EBCDIC table as any of the character or graphic data types, including CHARACTER, VARCHAR, CLOB, GRAPHIC, VARGRAPHIC, or DBCLOB to remove the restrictions of the binary string Unicode columns. These columns are recorded as characters and graphics in the Db2 catalog.

This section quantifies the performance results when populating and retrieving data out of Unicode columns in an EBCDIC table. It also compares it to performing the same operations against Unicode columns in Unicode tables, and EBCDIC columns in EBCDIC tables.

Environment setup

How the test environment was configured is shown in Figure 7-3.

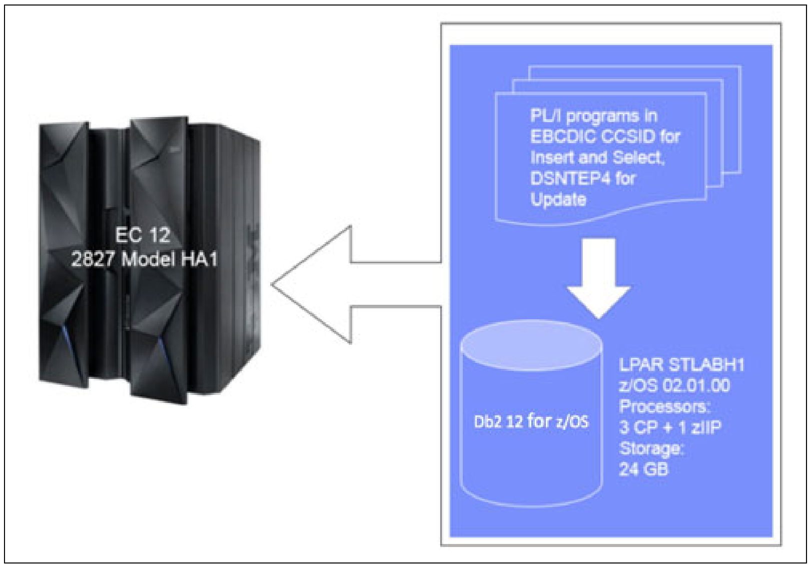


Figure 7-3 Hardware and software configuration

All Unicode columns in our test scenarios are defined as CHAR(20) and CCSID 1208, and only 10 bytes of data are inserted into the columns to trigger padding.

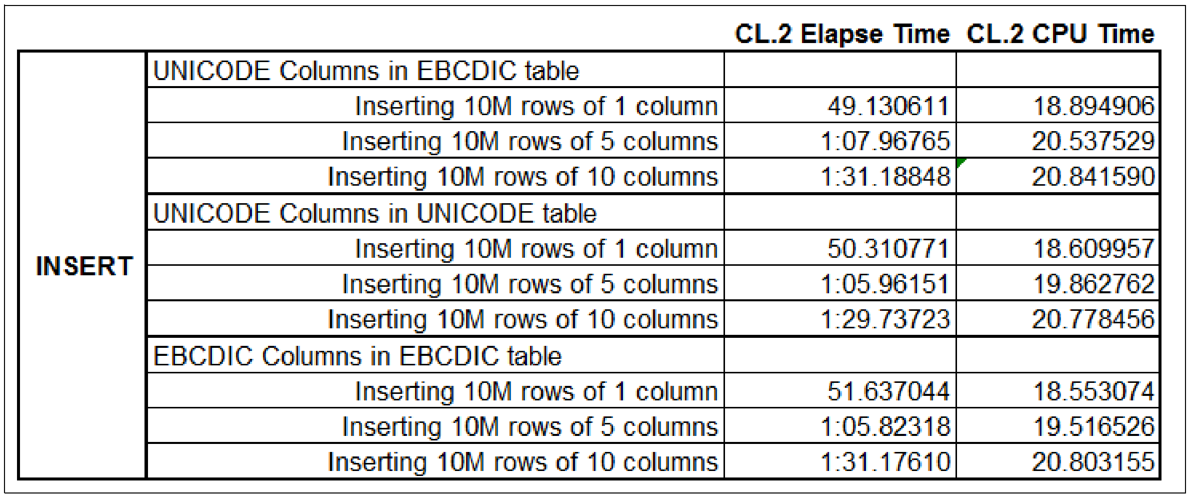
No indexes are defined on the table (indexes mixing EBCDIC and Unicode columns are not yet allowed by Db2). The table space that contains the test table is assigned to a buffer pool with 2.5 GB of storage, which is large enough to store the entire 10-column table.

The buffer pool is defined as PGFIX(YES) to use long-term page fixing.

Performance comparison

This section shows the measurements for inserting, selecting, and updating EBCDIC data in Unicode or EBCDIC columns in respective Unicode and EBCDIC tables.

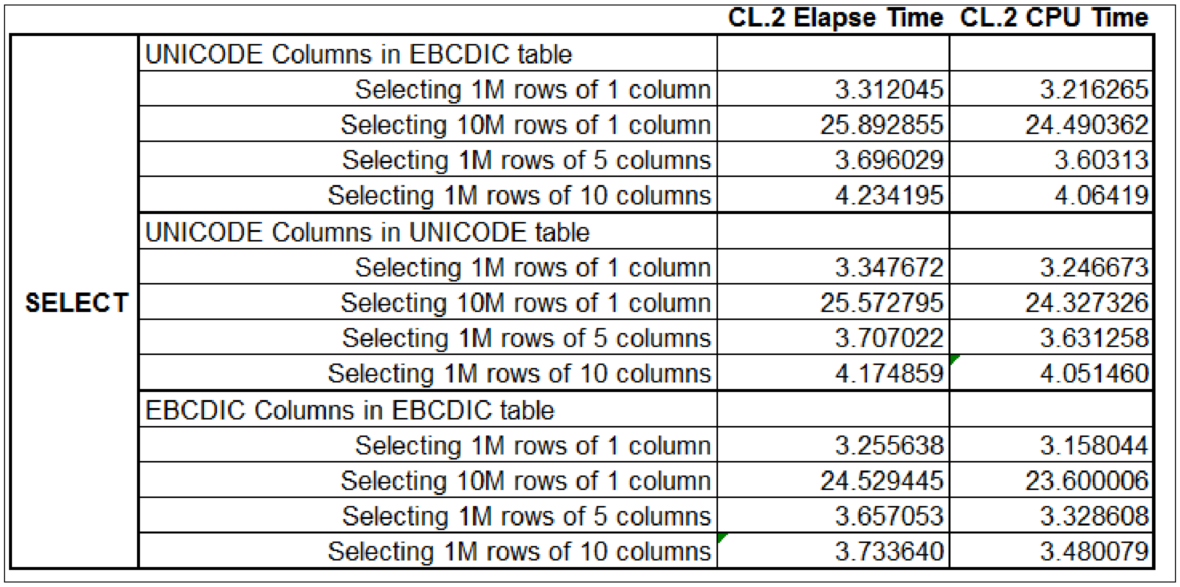
The INSERT EBCDIC performance results are shown in Figure 7-4.



		CL.2 Elapse Time	CL.2 CPU Time
INSERT	UNICODE Columns in EBCDIC table		
	Inserting 10M rows of 1 column	49.130611	18.894906
	Inserting 10M rows of 5 columns	1:07.96765	20.537529
	Inserting 10M rows of 10 columns	1:31.18848	20.841590
	UNICODE Columns in UNICODE table		
	Inserting 10M rows of 1 column	50.310771	18.609957
	Inserting 10M rows of 5 columns	1:05.96151	19.862762
	Inserting 10M rows of 10 columns	1:29.73723	20.778456
	EBCDIC Columns in EBCDIC table		
	Inserting 10M rows of 1 column	51.637044	18.553074
	Inserting 10M rows of 5 columns	1:05.82318	19.516526
	Inserting 10M rows of 10 columns	1:31.17610	20.803155

Figure 7-4 INSERT EBCDIC performance results

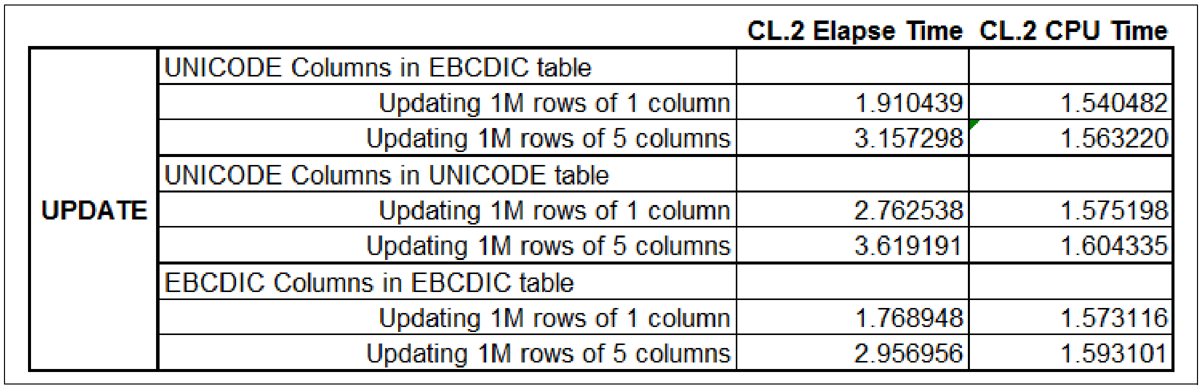
The SELECT EBCDIC data performance results are shown in Figure 7-5.



		CL.2 Elapse Time	CL.2 CPU Time
SELECT	UNICODE Columns in EBCDIC table		
	Selecting 1M rows of 1 column	3.312045	3.216265
	Selecting 10M rows of 1 column	25.892855	24.490362
	Selecting 1M rows of 5 columns	3.696029	3.60313
	Selecting 1M rows of 10 columns	4.234195	4.06419
	UNICODE Columns in UNICODE table		
	Selecting 1M rows of 1 column	3.347672	3.246673
	Selecting 10M rows of 1 column	25.572795	24.327326
	Selecting 1M rows of 5 columns	3.707022	3.631258
	Selecting 1M rows of 10 columns	4.174859	4.051460
	EBCDIC Columns in EBCDIC table		
	Selecting 1M rows of 1 column	3.255638	3.158044
	Selecting 10M rows of 1 column	24.529445	23.600006
	Selecting 1M rows of 5 columns	3.657053	3.328608
	Selecting 1M rows of 10 columns	3.733640	3.480079

Figure 7-5 SELECT EBCDIC performance results

The UPDATE EBCDIC data performance results are shown in Figure 7-6.



		CL.2 Elapse Time	CL.2 CPU Time
UPDATE	UNICODE Columns in EBCDIC table		
	Updating 1M rows of 1 column	1.910439	1.540482
	Updating 1M rows of 5 columns	3.157298	1.563220
	UNICODE Columns in UNICODE table		
	Updating 1M rows of 1 column	2.762538	1.575198
	Updating 1M rows of 5 columns	3.619191	1.604335
	EBCDIC Columns in EBCDIC table		
	Updating 1M rows of 1 column	1.768948	1.573116
	Updating 1M rows of 5 columns	2.956956	1.593101

Figure 7-6 UPDATE EBCDIC performance results

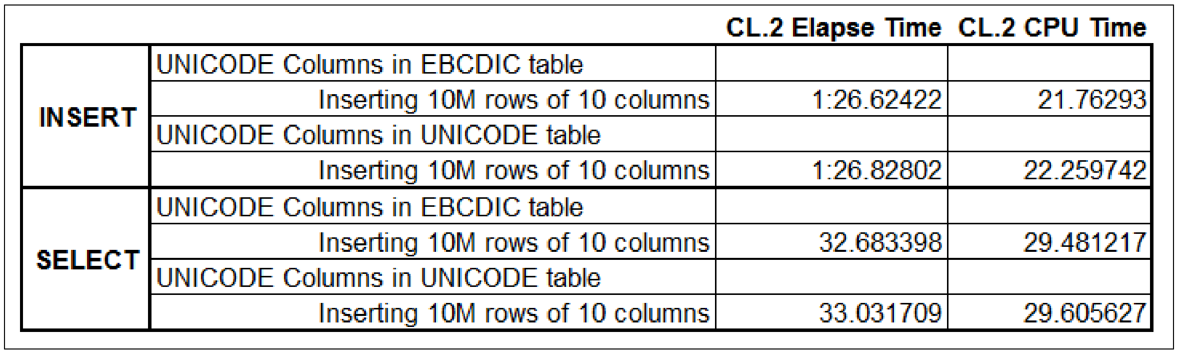
Inserting and selecting code page EBCDIC data from Unicode columns in EBCDIC tables perform similarly to performing the same insert and select against Unicode columns in Unicode tables and EBCDIC columns in EBCDIC tables. The only exception is that selecting from EBCDIC columns in EBCDIC tables outperforms the other two. This result occurs because when EBCDIC data is inserted into or selected from a Unicode table or column, the data is converted.

However, inserting EBCDIC data into a Unicode column is unrealistic (although it is a good baseline check) because no one intentionally adds a Unicode column to store EBCDIC data. Therefore, we completed another set of tests (as described next) to determine the cost of inserting and selecting Unicode data in a Unicode column of an EBCDIC table and a Unicode table.

In this scenario, all Unicode columns are defined as VARGRAPHIC (30), CCSID 1200 UTF-16 to be compatible with the PL/I program whose package and plan are bound with ENCODING(Unicode). This configuration ensures host variable and column data CCSID compatibility.

We insert only the 10-byte Unicode constant of CJK characters `X’4E004E014E024E034E044E054E064E074E084E09’`.

The INSERT, SELECT EBCDIC, UNICODE performance results are shown in Figure 7-7.



		CL.2 Elapse Time	CL.2 CPU Time
INSERT	UNICODE Columns in EBCDIC table		
	Inserting 10M rows of 10 columns	1:26.62422	21.76293
	UNICODE Columns in UNICODE table		
	Inserting 10M rows of 10 columns	1:26.82802	22.259742
SELECT	UNICODE Columns in EBCDIC table		
	Inserting 10M rows of 10 columns	32.683398	29.481217
	UNICODE Columns in UNICODE table		
	Inserting 10M rows of 10 columns	33.031709	29.605627

Figure 7-7 INSERT SELECT EBCDIC performance results

Performance results

The conclusion that we can draw from the data that is shown in Figure 7-7 is that inserting and selecting from Unicode columns in EBCDIC tables has similar performance characteristics as that of selecting Unicode columns in Unicode tables, which meets our performance expectation.

7.1.7 Piece-wise data modification

When the predicate in a delete statement qualifies for too many rows, the statement and recovery can be long-running because of the large unit of work. Beginning in Db2 12 for z/OS with new function (Function Level 500) activated, a FETCH FIRST n ROWS ONLY clause can be appended to the delete statement and break large delete operations into smaller and more manageable pieces. A COMMIT can then be issued in between these pieces to release locks and increase concurrency. In addition, smaller data modification allows for faster Db2 restart or faster data sharing failover and recovery.

Performance observation

A performance measurement was conducted comparing deleting 1 million rows versus deleting 200 rows while looping 5000 times (for a total of 1 million rows). For these runs, a test table was created by using the DDL that is shown in Example 7-12.

Example 7-12 DDL example for performance measurements

```
CREATE DATABASE DB18904;
CREATE TABLESPACE TS18904 IN DB18904
   USING STOGROUP SYSDEFLT
      PRIQTY 28800 SECQTY 57600
   FREEPAGE 0
   PCTFREE  0
   BUFFERPOOL BP8
   LOCKSIZE PAGE
   LOCKMAX 0
   CLOSE NO;

CREATE TABLE TB18904
      (
       PID      INT,
       COL01    CHAR(100))
IN DB18904.TS18904;
CREATE UNIQUE INDEX IX18904
   ON TB18904(PID)
   USING STOGROUP SYSDEFLT
      PRIQTY 14400 SECQTY 28800
   CLUSTER
   FREEPAGE 0
   PCTFREE 0
   BUFFERPOOL BP9
   CLOSE NO;
```

Then, the table was populated with 10 million rows with PID ranging 1 - 10000000 in ascending order. We then ran the two delete tests, as shown in Example 7-13. A LOAD was done between the tests to repopulate the table after the first delete.

Example 7-13 DELETE test statements

```
DELETE FROM USRT001.TB18904 WHERE PID < 1000001;
COMMIT;
for(i=1; i<5001; i++)  {
DELETE FROM USRT001.TB18904 WHERE PID < 1000001 FETCH FIRST 200 ROWS ONLY;
COMMIT;
}
```

With 5000 commits for test scenario 2, the in Db2 class 2 CPU and elapsed time for test 2 is approximately 2.5 times as high as test 1 with a comparable number of getpages, LOCK, and CHANGE requests. The significant increase in CPU time is expected and acceptable in exchange for better concurrency and recoverability that results from breaking up the mass delete into smaller chunks. It is a good application coding practice to control the number of rows that are processed by DELETE statements by specifying FETCH FIRST n ROWS ONLY. It also is good application coding practice to mitigate the effects of locking and logging when millions of rows are potentially or unintentionally affected by a simple DELETE statement.

7.2 Resource Limit Facility

Db2 12 for z/OS extends the Resource Limit Facility (RLF) reactive governing to support static SQL statements, such as SELECT (cursor or singleton), INSERT, UPDATE, MERGE, TRUNCATE, or DELETE. This new support is available from function level V12R1M100 and must be enabled by setting the following new system parameter:

```
RLFENABLE=ALL or STATIC
```

Where ALL means both dynamic and static, as shown in the following example:

```
STATIC means static SQL only
```

The DDL for the creation of the DSNRLMTxx table is shown in Example 7-14.

Example 7-14 DDL for creating the DSNRLMT01 table

```
CREATE TABLE DSNRLMT01
      (RLFFUNC       CHAR(1)      NOT NULL WITH DEFAULT,
       RLFEUAN       VARCHAR(255) NOT NULL WITH DEFAULT,
       RLFEUID       VARCHAR(128) NOT NULL WITH DEFAULT,
       RLFEUWN       VARCHAR(255) NOT NULL WITH DEFAULT,
       RLFIP         CHAR(254)    NOT NULL WITH DEFAULT,
       ASUTIME       INTEGER,
       RLFASUERR     INTEGER,
       RLFASUWARN    INTEGER,
       RLF_CATEGORY_B CHAR(1)     NOT NULL WITH DEFAULT)
```

For reactive governing using the DSNRLMT01 table (Example 7-14), RLFFUNC=B means that the row reactively governs static SQL statements by using client information (RLFEUID, RLFEUAN, RLFEUWN, and RLFIP).

The DDL for the creation of the DSNRLSTxx table is shown in Example 7-15.

Example 7-15 DDL for creating the DSNRLST01 table

```
CREATE TABLE DSNRLST01
      (AUTHID        VARCHAR(128) NOT NULL WITH DEFAULT,
       PLANNAME      CHAR( 8)     NOT NULL WITH DEFAULT,
       ASUTIME       INTEGER,
       LUNAME        CHAR( 8)     NOT NULL WITH DEFAULT,
       RLFFUNC       CHAR( 1)     NOT NULL WITH DEFAULT,
       RLFBIND       CHAR( 1)     NOT NULL WITH DEFAULT,
       RLFCOLLN      VARCHAR(128) NOT NULL WITH DEFAULT,
       RLFPKG        VARCHAR(128) NOT NULL WITH DEFAULT,
       RLFASUERR     INTEGER,
       RLFASUWARN    INTEGER,
       RLF_CATEGORY_B CHAR( 1)    NOT NULL WITH DEFAULT)
```

Consider the following points regarding Example 7-15:

- For reactive governing using the DSNRLST01 table (Example 7-14), RLFFUNC = A means that the row reactively governs static SQL statements by package or collection name.
- ASUTIME is used to limit the statement's execution time. Db2 issues SQLCODE -905 if the execution time exceeds the ASUTIME limit.

Tip: SQL code -905 means `UNSUCCESSFUL EXECUTION DUE TO RESOURCE LIMIT BEING EXCEEDED, RESOURCE NAME = resource-name LIMIT = limit-amount1 CPU SECONDS (limit-amount2 SERVICE UNITS) DERIVED FROM limit-source`.

7.2.1 Performance objectives and evaluation

RLF for static SQL includes the following the performance objectives:

- No regression when RLF is not active.
- Minimal performance affect for static SQL statements that are governed when RLF is active.

Several test scenarios were conducted to evaluate dynamic SQL statements, static SQL statements, and mixed SQL statements with RLF tables (including an RLF table with about 20 thousand records, with options of matching and governing). For more information about governing rules, see *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

The performance objectives were met. Up to 2% overhead was observed for static SQL statements that are not governed or governed with no limit (zparm's default).

With the RLF table data cached in memory, no getpages are available from the RLF table space. Therefore, we also observed significant CPU time improvements for the dynamic SQL statements in the case of local dynamic SQL with a large RLF table and matching UserID.

The following section describes the performance results when the IRWW SQLJ and JDBC workload was measured.

Distributed IRWW static workload

We observed an approximate 2% overhead when RLF was used.

The performance observations for the distributed IRWW static workload and governed with limit are listed in Table 7-3.

Table 7-3 Performance observations IRWW static workload

Delta (%) Total CPU/Commit	**No matching**	**Matching**
SQLJ (static)	1.03	1.11
Delta (%) CL2 CPU	**No matching**	**Matching**
SQLJ (static)	2.06	1.51

Distributed IRWW dynamic workload

With the RLF enhancement to cache RLF table data in memory to avoid getpages from the RLF table space, we also observed the following significant performance improvements for the distributed IRWW dynamic workload and governed with limit:

- Improved performance of approximately 7% of CL2 CP time
- Improved performance of approximately 3% Total CPU time

The measurements for this workload are listed in Table 7-4.

Table 7-4 Performance observations IRWW static workload

% Change	No matching	Matching
Total CPU/commit	2.65	3.89
CL1 CP	2.79	4.14
CL2 CP	6.89	7.21

7.3 JSON support

Db2 11 for z/OS added JSON support for requests from web, mobile, and cloud applications. Db2 12 for z/OS adds the ability to port JSON directly to Db2. Users can work with JSON data with the Type 4 Java Client driver or through SQL extensions. The JSON built-in functions and UDFs rely on JSON documents being stored in an internal binary format that is named BSON in Db2.

However, the JSON functionality is not included in the base Db2 product. You must install the Accessories Suite (a no-charge feature) to enable it.

The JSON_VAL built-in function is the main function that provides an SQL interface to extract and retrieve JSON data from BSON objects. In Db2 12, this function is enhanced so that the first argument is no longer required to be a BLOB column. In Db2 12, the first parameter can be the JSON objects from a view column, CASE expression, or table expression with UNION ALL.

In addition to the built-in function JSON_VAL, Db2 12 includes a set of UDFs for storing and retrieving BSON documents in the Db2 BLOBs.

The following UDFs are provided by Db2:

- SYSTOOLS.BSON2JSON
- SYSTOOLS.JSON2BSON
- SYSTOOLS.JSON_BINARY
- SYSTOOLS.JSON_BINARY2
- SYSTOOLS.JSON_GET_POS_ARR_INDEX
- SYSTOOLS.JSON_LEN
- SYSTOOLS.JSON_TABLE
- SYSTOOLS.JSON_TABLE_BINARY
- SYSTOOLS.JSON_TYPE
- SYSTOOLS.JSON_UPDATE
- SYSTOOLS.REGSP
- SYSTOOLS.REG_MATCHES
- SYSTOOLS.REG_MATCHES_NOFLAGS

7.3.1 Built-in function JSON_VAL

The Db2 for z/OS built-in function JSON_VAL provides an SQL interface to extract and retrieve JSON data from the database. This function accepts only the BSON type of JSON document, so its argument must be a column from the table that contains the JSON document in BSON format, or a SYSTOOLS.JSON2BON function that returns the BSON format of JSON document.

The JSON_VAL built-in function returns an element of a JSON document that is identified by the JSON field name that is specified in the search string. The value of the JSON element is returned in the data type and length that is specified in result-type.

For example, if the CUSTOMERS table is designed to store JSON data and we want to determine how many customers have a postal code of 95123, the query that is shown in Example 7-16 can be used.

Example 7-16 SELECT JSON example

```
SELECT count(*) from CUSTOMERS
WHERE JSON_VAL(DATA, '.zipcode', 's:5') = '95123';
```

As shown in Example 7-17, we can list customers' names whose account balance is less than 10000.

Example 7-17 SELECT JSON complex example

```
SELECT JSON_VAL(T1.DATA, ‘CustLastName’, ‘s:30’) || ‘, ‘ ||
JSON_VAL(T1.DATA,’CustFirstName’, s:30’) as “Customer Name”
FROM CUSTOMERS T1, ACCOUNTS T2
WHERE JSON_VAL(T1.DATA, ‘CUSTID’, ‘I' =JSON_VAL(T2.DATA, ‘CUSTID’, ‘I’)
AND JSON_VAL(T2.DATA,’Balance’, ‘f’) > 10000;
```

Another example is updating a JSON document in the table, as shown in Example 7-18.

Example 7-18 Update JSON document

```
UPDATE CUSTOMERS
SET DATA=SYSTOOLS.JSON2BSON(
       ‘{“CUSTOMERS”: {“CUSTID”:888,
                       “Lastname”:”John”,
                       “FirstName”:”Smith”,
                       …………
                      }
       }’)
WHERE JSON_VAL(DATA,’CUSTID’, ‘i:na’) = 888;
```

Similar to the UPDATE operation on JSON documents, we can delete a JSON document from the table. Example 7-19 shows an SQL statement that deletes a JSON document. By starting JSON_VAL in the search condition of the DELETE statement, you can locate the record to be deleted and the DELETE statement deletes the JSON data if it is found.

Example 7-19 DELETE JSON example

```
DELETE FROM CUSTOMERS
WHERE JSON_VAL(DATA, ‘CUSTID’, ‘i:na’) = 888;
```

Consider the following points:

- The `:na` is needed to ensure that the array type is not returned because Db2 does not support an array type.
- To achieve better performance for operations on JSON data, it is necessary to have a proper index on the elements in a JSON document.

7.3.2 UDFs for JSON support

Db2 for z/OS provides several user-defined functions to support operations on JSON data. To create them, the DDL that is provided in the Db2 Accessory Suite must be run. These UDFs are typically called by the JSON API to update or retrieve JSON data.

The following JSON UDFs are important:

- SYSTOOLS.JSON2BSON(INJSON CLOB(16M)) RETURNS BLOB(16M)

  This user-defined function accepts a JSON document as the argument, transforms it, and returns the BSON (Binary JSON) of that JSON document.

- SYSTOOLS.JSON_BINARY (INJSON BLOB (16M), INELEM VARCHAR (2048), MAXLENGTH INTEGER) RETURNS VARCHAR (2050) FOR BIT DATA

  This user-defined function returns a binary portion of JSON data and is typically used in select projections or where clause binary comparisons.

- SYSTOOLS.JSON_TABLE (INJSON BLOB (16M), INELEM VARCHAR (2048), RETTYPE VARCHAR (100)) RETURNS TABLE (TYPE INTEGER, VALUE VARCHAR (2048))

  This function returns a table with two columns. The first column is the BSON type; second column is the string value.

- SYSTOOLS.JSON_UPDATE (INJSON BLOB (16M), INELEM VARCHAR (32672)) RETURNS BLOB (16M)

  This function is used to update JSON data and is usually called from the JSON API.

- SYSTOOLS.JSON_LEN (INJSON BLOB (16M), INELEM VARCHAR (2048)) RETURNS INTEGER.

  This function returns the size of array of elements in JSON data and returns NULL if an element is not an array.

7.3.3 JSON performance

We evaluated the performance of INSERT, SELECT, UPDATE, and DELETE statements on JSON documents that are stored in Db2 z/OS tables. All measurements were performed by using Db2 for z/OS 11.

INSERT performance

To store a JSON document into a Db2 table, users can use the Java API that is included with the Db2 Accessory Suite or directly insert the BLOB data into the DATA column by using the JSON2BSON UDF, as shown in Example 7-20.

Example 7-20 INSERT JSON example

```
INSERT INTO CUSTOMERS VALUES (102,
   SYSTOOLS.JSON2BSON(‘{“CUSTOMERS”: {“CUSTID”:888,
                              “Lastname”:”John”,
                              “FirstName”:”Smith”,
                                    ...........
                                    }
                       }’));
```

The measurements’ results of inserting a JSON document into a Db2 table by using SQL and JSON API are listed in Table 7-5. The data shows that inserting a JSON document by using JSON Java API takes a bit longer than the use of SQL directly. This difference occurs because the client must construct a well-formed JSON document before inserting it into the table.

Table 7-5 Time Comparison (in microseconds) between SQL and JSON API

	CL1 elapsed time	**CL2 elapsed time**	**CP1 CP time**	**CL2 CP time**
SQL	607	557	134	101
API	636	557	146	101

SELECT performance

Users can retrieve the value of a specific field in a JSON document from the BLOB column DATA directly with the built-in function JSON_VAL.

The statement that was performed to assess the SELECT by using the built-in function JSON_VAL performance as an example of JSON_VAL usage is shown in Example 7-21.

Example 7-21 JSON Select with built-in functions

```
String str = " SELECT JSON_VAL(DATA, \'properties.BLOCK_NUM\', \'s:10\'), " +
"JSON_VAL(DATA, \'properties.LOT_NUM\', \'s:10\') " + " FROM TEST.CITYLOTS WHERE
JSON_VAL(DATA, \'properties.BLKLOT\', \'s:10:na\') ‘= ? " ;
```

The elapsed time and CPU time (in microseconds) of the SELECT statement and the JSON Java API are listed in Table 7-6. The API costs more than selecting directly from the table.

Table 7-6 Time Comparison (in microseconds) between SQL and JSON API

	CL1 elapsed time	**CL2 elapsed time**	**CP1 CP time**	**CL2 CP time**
SELECT	138	83	115	82
Find()	201	87	141	86

Therefore, the use of API, which in turn starts the built-in function JSON_VAL, has some overhead. However, the MongoDB-like API provides a rich set of operations to search and retrieve JSON documents.

UPDATE performance

To update data in a JSON document, you can use an SQL UPDATE statement or the built-in function JSON_VAL. However, with this method, the entire value of column DATA is updated and the input to the JSON data must be a well-formed JSON document. Alternatively, Db2 JSON API `DBCollection.FindAndModify()` can be used to find a document, modify some specific fields, return that document.

The elapsed time and CPU time (in microseconds) of the UPDATE statement versus the JSON Java API `FindAndModify()` are listed in Table 7-7. The use of the API costs more than updating a BLOB column in the table directly by using SQL. This difference occurs because the function `FindAndModify()` must start several UDFs to find the position of the column to be updated, then used the function update() to update data, and return the updated data.

Table 7-7 Time Comparison (in microseconds) between SQL and JSON API

	CL1 elapsed time	**CL2 elapsed time**	**CP1 CP time**	**CL2 CP time**
UPDATE	324	170	197	158
FindAndModify()	674	432	257	197

DELETE performance

To delete data in a JSON document, you can use the SQL DELETE statement. In addition, several Db2 JSON API methods are available to delete one or more JSON documents. `DBCollection.remove()` can be used to delete all documents or delete only specific documents in a collection. Another method, such as `FindAndRemove()`, can be used to delete a document in a collection (this method also returns the removed document).

The elapsed time and CPU time (in microseconds) of deleting a JSON document by using the SQL DELETE statement and the JSON Java API are listed in Table 7-8. The data shows that the use of the API costs more than directly deleting a BLOB column in the table. However, the use of the JSON API is easier to program by using JavaScript.

Table 7-8 Time Comparison (in microseconds) between SQL and JSON API

	CL1 elapsed time	**CL2 elapsed time**	**CP1 CP time**	**CL2 CP time**
DELETE	382	333	125	92
remove()	532	471	159	116
FindAndRemove()	496	379	205	145

7.4 Built-in LISTAGG aggregate function

Db2 12 for z/OS introduces a new built-in aggregate function called LISTAGG. It provides a simple and efficient manner to collapse rows of a column into a string.

Tip: This feature is available after the activation of Function Level 501 or higher. For more information about activating Function Level 501 (or higher) in Db2 12 for z/OS, see the IBM Knowledge Center.

The LISTAGG function is used to aggregate a set of string values within a group into one string. An optional separator argument, which is inserted between adjacent input strings, can be used to delimit the items in the result list. Specifying a comma as the separator produces a comma-separated list.

An optional ordering can also be specified for the items in the WITHIN GROUP clause.

7.4.1 Sample scenario and use cases for LISTAGG

This section describes an application example of the LISTAGG function.

Consider the table EMP and its data that is listed in Table 7-9.

Table 7-9 Table EMP

Dept_ID	E_Name	Emp_ID	Birth
10	Jack	0012	1983
10	Lily	0015	1990
20	Tom	0019	1983
20	Bob	0022	1976
20	Frank	0004	1983
20	Tom	0014	1985
30	Jerry	0028	1991
30	Chris	0021	1981
30	Jill	0002	1984
30	Jerry	0031	1984
30	Allan	0006	1995

Assume the requirement to output all employees' names under the same department into one row, in ascending order according to their birth date and employees' ID. This section shows three approaches to perform the task.

Method #1: Application + SQL

This method involves the following procedure:

1. Define a cursor for SQL to query the employee's name and department from the EMP table.
2. Fetch data from the cursor and use application logic to judge whether the employee's name is under same department and then perform concatenation.

This method requires users to implement the judgment logic and concatenation, which is error prone and uses time and resources.

Method #2: Use complex SQL/PL

The method of the use of complex SQL/PL to get the results that LISTAGG can provide easily is shown in Example 7-22.

Example 7-22 Complex SQL/PL to aggregate EMP data

```
CREATE TABLE AGGRESU(
    DEPT_ID INTEGER NOT NULL,
E_NAME VARCHAR(1000) NOT NULL);
CREATE PROCEDURE AggEmpName (…)
   BEGIN
     SET STMT= '';
     SET PRE_Dept_ID= '';
```

```
      FOR MYC CURSOR FOR
        SELECT Dept_ID, E_Name
        FROM EMP
        ORDER BY Dept_ID, Birth, Emp_ID
        DO
          IF PRE_Dept_ID = '' THEN
            SET PRE_Dept_ID = Dept_ID;
            SET STMT = STMT ||E_Name||';';
          ELSE
             IF PRE_Dept_ID = Dept_ID THEN
                SET STMT = STMT ||E_Name||';';
             ELSE
               INSERT INTO AGGRESU VALUES (PRE_Dept_ID,STMT);
               SET PRE_Dept_ID = Dept_ID;
               SET STMT= '';
                SET STMT = STMT ||E_Name||';';
             END IF;
          END IF;
      END FOR;
   END;
CALL AggEmpName (…) ;
```

Method #3: Use LISTAGG function

How the problem might be solved by using the LISTAGG function is shown in Example 7-23.

Example 7-23 Use of the LISTAGG function

```
SELECT Dept_ID, LISTAGG(ALL E_Name, ’; ’)
WITHIN GROUP (ORDER BY Birth, Emp_ID) AS Name_List
FROM EMP
GROUP BY Dept_ID;
```

The result set that is generated by this query is listed in Table 7-10.

Table 7-10 LISTAGG results

Dept_ID	Name_List
10	Jack; Lily
20	Bob; Frank; Tom; Tom
30	Chris; Jill; Jerry; Jerry; Allan

It is clear that the LISTAGG function is simpler and easier to use than the first and second methods.

7.4.2 Performance results for LISTAGG function

In the sample scenario that is described in 7.4.1, “Sample scenario and use cases for LISTAGG” on page 222, controlled test measurements compare the use of Db2 SQL/PL (Method #2) with the use of the LISTAGG function (Method #3).

The table that is used for performance testing features one unique index on the column group of the grouping column, ordering column, and the aggregated column. In total, 1400 rows in this table are included and the max aggregated string set has 320 strings.

Performance results show a 97% CPU reduction and a 94% elapsed time reduction when the LISTAGG function is used.

Note: For more information about LISTAGG, see the IBM Knowledge Center.

7.5 XML Optimizations

Db2 12 for z/OS enhances XML performance in several ways, with changes to XMLMODIFY, XMLTABLE, and access path optimizations.

This section describes XML performance enhancements and their performance measurements.

Note: For more information about XML-related enhancements in Db2 12 for z/OS, see *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

7.5.1 XML subdocument update

The XMLMODIFY built-in scalar function was introduced in Db2 10 for z/OS to allow XQuery updating expressions to be used in an SQL statement when an XML document is updated in an XML column.

The following basic updating expressions are supported:

- INSERT
- DELETE
- REPLACE

Before Db2 12, only a single basic updating expression is allowed in XMLMODIFY. In Db2 12 Functional Level 500, multiple updating expressions are allowed with which multiple updates in a single XMLMODIFY can be done by using one of the following methods:

- An FLWOR expression that contains an updating expression in its return clause.
- A conditional expression that contains an updating expression in its then or else clause.
- Two or more updating expressions that are separated by commas where all operands are either updating expressions or an empty sequence.

Note: Performing multiple updates in one update statement can greatly improve performance by avoiding calling XMLMODIFY multiple times.

XMLModify examples

In this section, examples of how you can use the XMLModify statement to perform multiple updates are presented. Consider a table that is called `purchaseOrders` that contains a column that is named PO that contains the XML document, as shown in Example 7-24.

Example 7-24 Sample XML document for XMLModify examples

```
<ipo:purchaseOrder
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:ipo="http://www.example.com/IPO"
  xmlns:pyd=http://www.examplepayment.com  orderDate="1999-12-01"
pyd:paidDate="2000-01-07">
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Helen Zoe</name>
    <street>47 Eden Street</street>
    <city>Cambridge</city>
    <postcode>CB1 1JR</postcode>
  </shipTo>
<items>
  <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>149.99</USPrice>
      <shipDate>2011-05-20</shipDate>
 </item>
  <item partNum="945-ZG">
       <productName>Sapphire Bracelet</productName>
       <quantity>2</quantity>
       <USPrice>178.99</USPrice>
       <comment>Not shipped</comment>
    </item>
   </items>
</ipo:purchaseOrder>
```

Multiple updates by using a sequence expression

Multiple updates by using a sequence expression are shown in Example 7-25. This statement can be used to replace the quantity of the item with partNum=”872-AA’ with 2, and delete the item with partNum=”945-ZG” at the same time.

In Db2 11, we must use two XMLMODIFY statements to achieve this result.

Example 7-25 XML multiple updates by using a sequence expression

```
UPDATE purchaseOrders
SET PO = XMLMODIFY(
 'declare namespace ipo="http://www.example.com/IPO";
 replace value of node
 /ipo:purchaseOrder/items/item[@partNum="872-AA"]/quantity
   with xs:integer(2),
 delete node
   /ipo:purchaseOrder/items/item[@partNum="945-ZG"]’);
```

Multiple updates by using an FLWOR expression

How to perform updates by using an FLWOR expression is shown in Example 7-26. This statement increases the `USPrice` of each item by 10%.

Example 7-26 XML multiple updates by using an FLWOR expression

```
UPDATE purchaseOrders
SET PO = XMLMODIFY(
 'declare namespace ipo="http://www.example.com/IPO";
  for $i in /ipo:purchaseOrder/items/item
  let $p := $i/USPrice
  where xs:decimal($p)>0
  return
    replace value of node $p with $p *1.1');
```

Multiple updates by using conditional expressions

How multiple updates are run by using a conditional expression is shown in Example 7-27. Here, all items with a quantity greater than one are updated with a discount of 10%. Also, if the quantity is one or less, the price is increased by 5%.

Example 7-27 XML multiple updates by using a conditional expression

```
UPDATE purchaseOrders
SET PO = XMLMODIFY(
  'declare namespace ipo="http://www.example.com/IPO";
   for $i in /ipo:purchaseOrder/items/item
   let $p := $i/USPrice
   let $q := $i/quantity
   where xs:decimal($p)>0 and xs:integer($q)>0
   return
   (
             if (xs:integer($q) > 1) then
      replace value of node $p with $p *0.9
    else
      replace value of node $p with $p *1.05
    )'
);
```

Performance results

The purpose of these tests is to ensure that the new method of grouping XML Update statements into a single XMLMODIFY statement uses equal or less CPU time than running multiple separate XMLMODIFY statements.

Single update versus multiple updates performance

The following XML Docs were used for testing this enhancement:

- IBM Healthcare 10MB doc: 50 docs inserted and 10 deletes were issued against each document
- IBM Catalog 1MB doc: 3000 docs inserted, 2 different scenarios tested:
  - 1 replace/delete issued against each document
  - 5 replace/deletes issued against each document

- IBM Catalog 10MB doc: 50 docs inserted, 2 scenarios tested:
  - 10 replaces issued against each document
  - 5 inserts issued into each document.

XML Update was run on these documents by using a single XMLMODIFY in Db2 12. It was compared to multiple XMLMODIFY statements with Db2 11. The results are described next.

The performance results of the IBM Catalog 1MB scenario are shown in Figure 7-8.

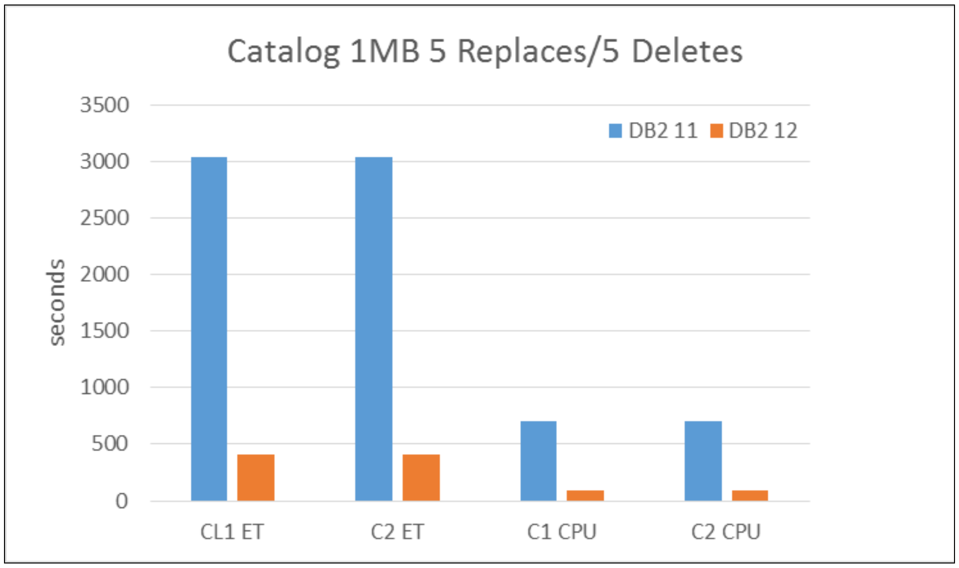


Figure 7-8 Catalog 1 MB with five replaces and five deletes times

The performance results of the Medline document scenario when comparing Db2 11 to Db2 12 for z/OS are shown in Figure 7-9.

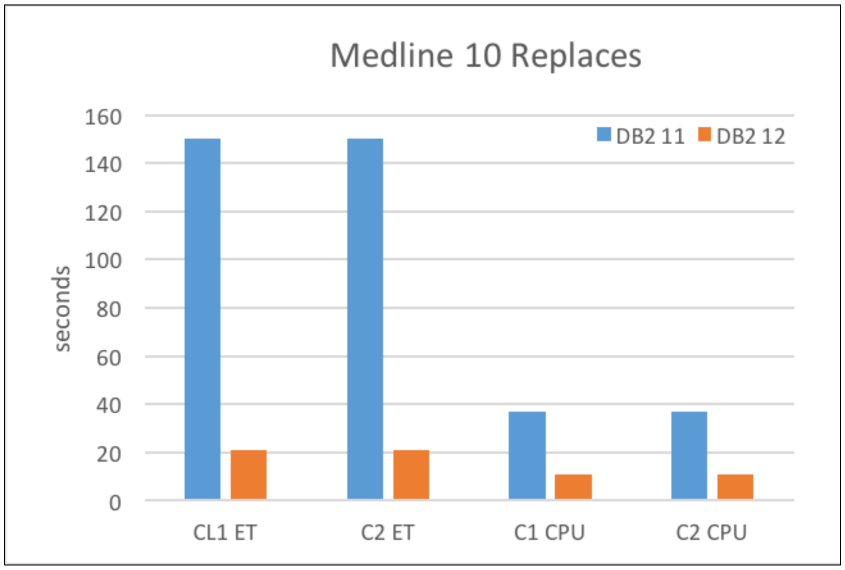


Figure 7-9 Medline 10MB with 10 replaces times

The performance results for the IBM Catalog 10MB document 10 replaces scenario are shown in Figure 7-10.

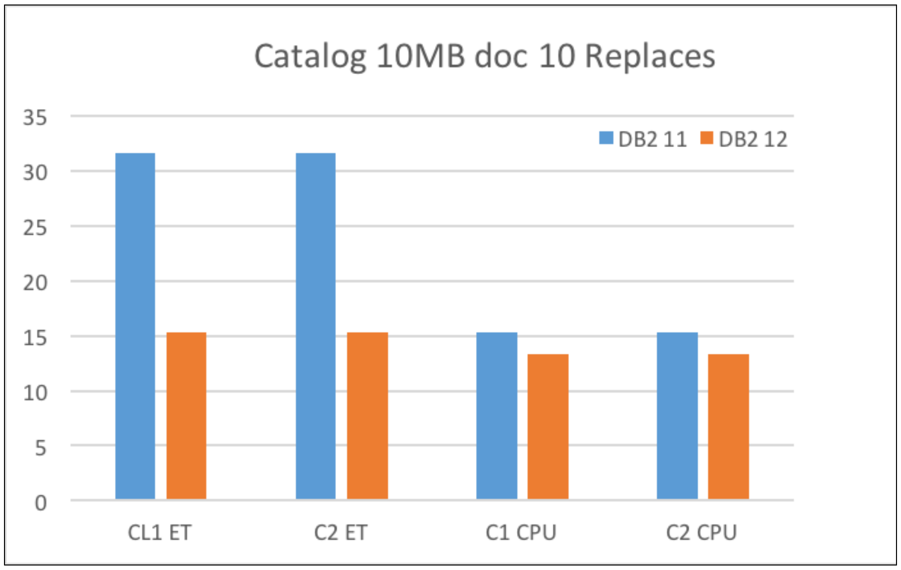


Figure 7-10 Catalog 10MB with 10 replaces times

The performance results for the Catalog 10MB 5 inserts document scenario are shown in Figure 7-11.

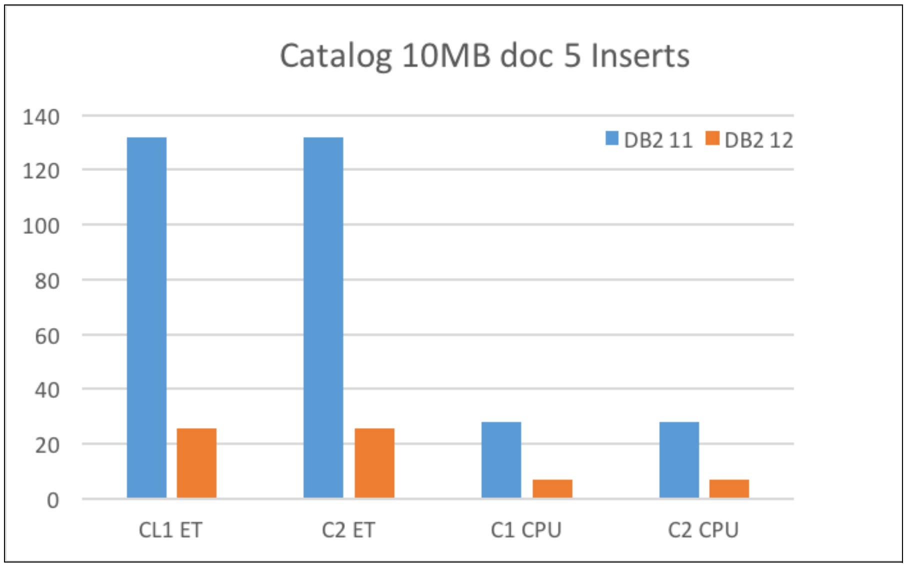


Figure 7-11 Catalog 10 MB with five inserts times

These results show a significant improvement in CPU and Elapsed time on every scenario when comparing Db2 12 to Db2 11.

FLWOR expression performance

The purpose of this test is to evaluate the performance effect of the new FLWOR functions.

The performance expectation is to have equal or better CPU time than running multiple separate XMLMODIFY statements.

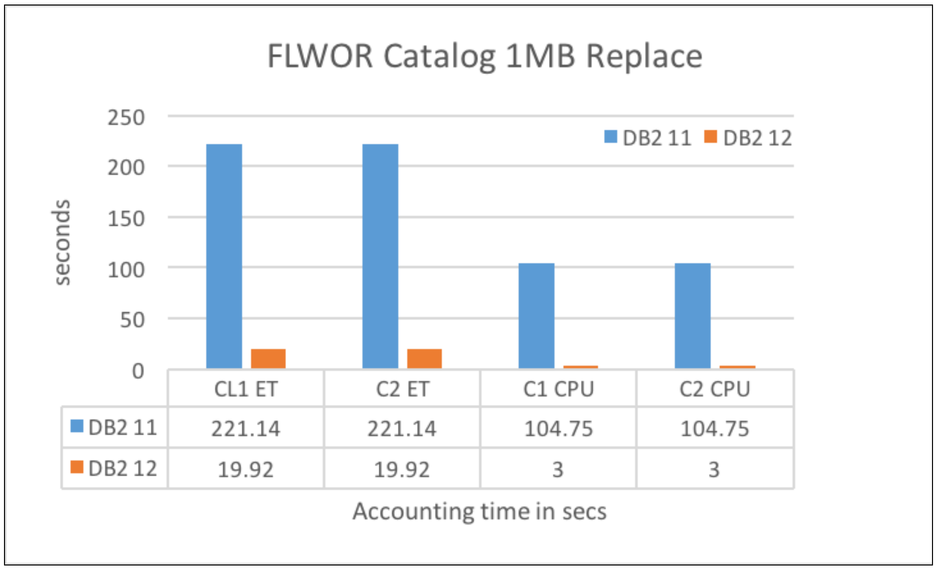
The test scenario involves 3000 1MB Catalog Docs. Eight instances of `quantity_in_stock` are found in each document.

The update XMLMODIFY statement is shown in Example 7-28.

Example 7-28   XML update XMLMODIFY statement

```
UPDATE XML2TB SET XML2 = XMLMODIFY(
  'for $i in /catalog/item
   let $c := $i/cost
   let $q := $i/quantity_in_stock
   where xs:integer($q)=10
  return
       replace value of node $c with $c *0.9'
```

The performance measurements are shown in Figure 7-12.



	CL1 ET	C2 ET	C1 CPU	C2 CPU
DB2 11	221.14	221.14	104.75	104.75
DB2 12	19.92	19.92	3	3

Figure 7-12   FLWOR Catalog 1MB replace times

The results show a significant improvement in CPU and Elapsed time on every scenario when comparing Db2 12 to Db2 11.

Validation performance

The purpose of this test is to measure the performance effects of the new method of grouping XML Update statements in a single XMLMODIFY.

It is expected to be using equal or less CPU time compared to running multiple separate XMLMODIFY statements when Schema Validation is also performed.

The following XML Docs were used for testing:

- Medline 10MB doc: 400 docs inserted and 10 deletes were issued against each document
- Catalog 10MB doc: 50 docs inserted, 2 scenarios tested:
  - 10 replaces issued against each document
  - 5 inserts issued into each document

The performance measurements for the Medline 10 MB test scenario are shown in Figure 7-13.

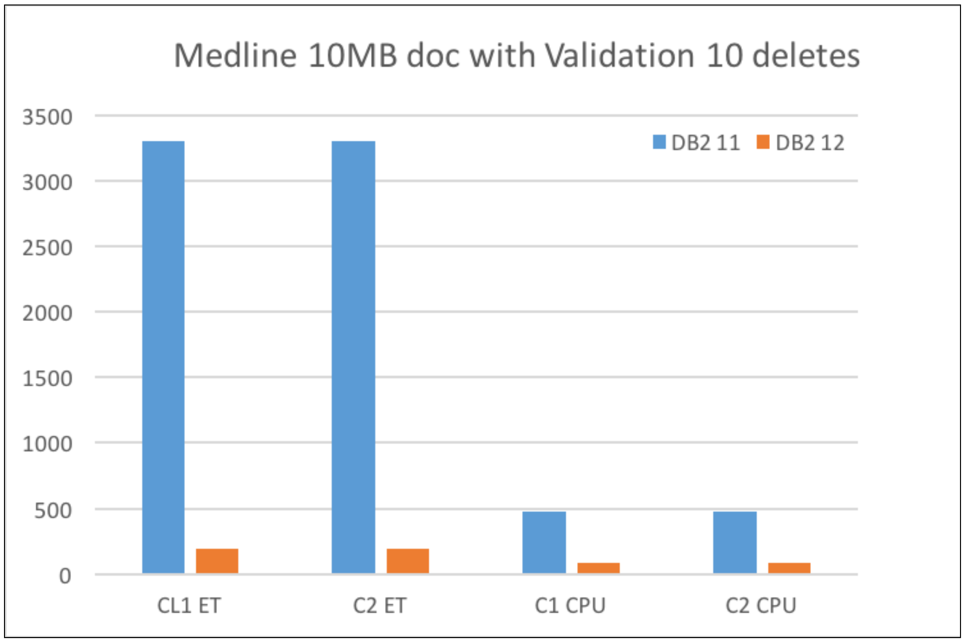


Figure 7-13 Medline 10MB times

The results for the Catalog 10MB document validation 10 replaces scenario are shown in Figure 7-14.

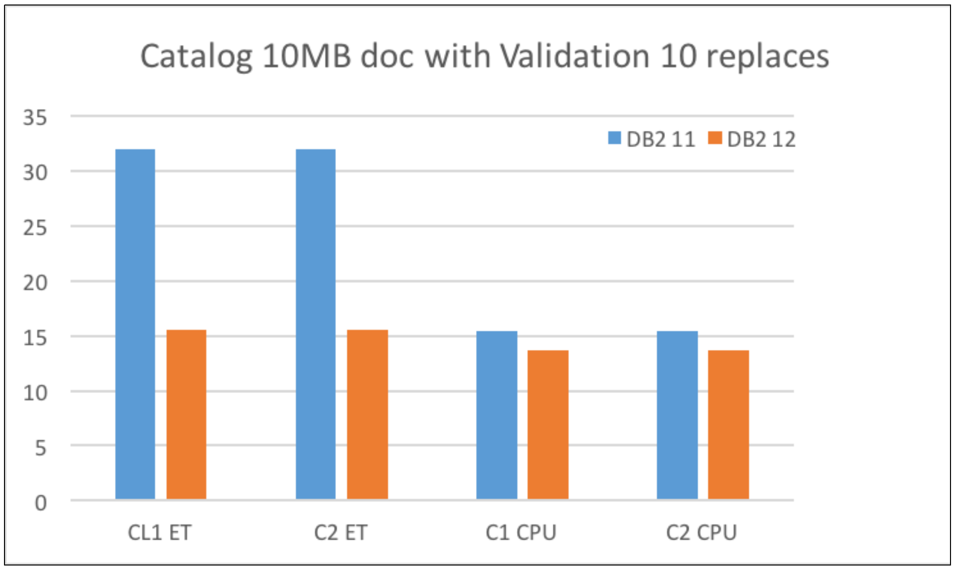


Figure 7-14 Catalog 10MB times

The results show a significant improvement in CPU and Elapsed time on every scenario when comparing Db2 12 to Db2 11. In the scenario IBM Catalog 10MB with 10 replaces, a comparable CPU time was observed, but the elapsed time reduction shows the advantages that were introduced in Db2 12 for z/OS.

7.5.2 XML access plan optimization

Db2 12 Functional Level 100 improves the performance of XML queries by optimizing the access path selection process.

To use this enhancement, new histogram and frequency statistics are collected on the leading column of the XML value indexes. The filter factor formula for the XMLEXISTS predicates is enhanced to better use these new statistics.

Tip: For more information about the Db2 12 for z/OS Function Levels, see the chapter "Continuous delivery" in *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

Performance results

In this section, we describe several tests that were performed to measure the CPU savings when these statistics are used.

First, the appropriate histogram and frequency statistics are collected by using the **RUNSTATS** command, as shown in Example 7-29.

Example 7-29 RUNSTATS command to collect histogram statistics

```
RUNSTATS TABLESPACE DBSEC3.XSEC0000 INDEX(ALL)
         FREQVAL NUMCOLS 1 COUNT 10 MOST
         HISTOGRAM NUMCOLS 1 NUMQUANTILES 5 REPORT YES
```

The performance results when queries are run after the **RUNSTATS** command is used are shown in Figure 7-15.

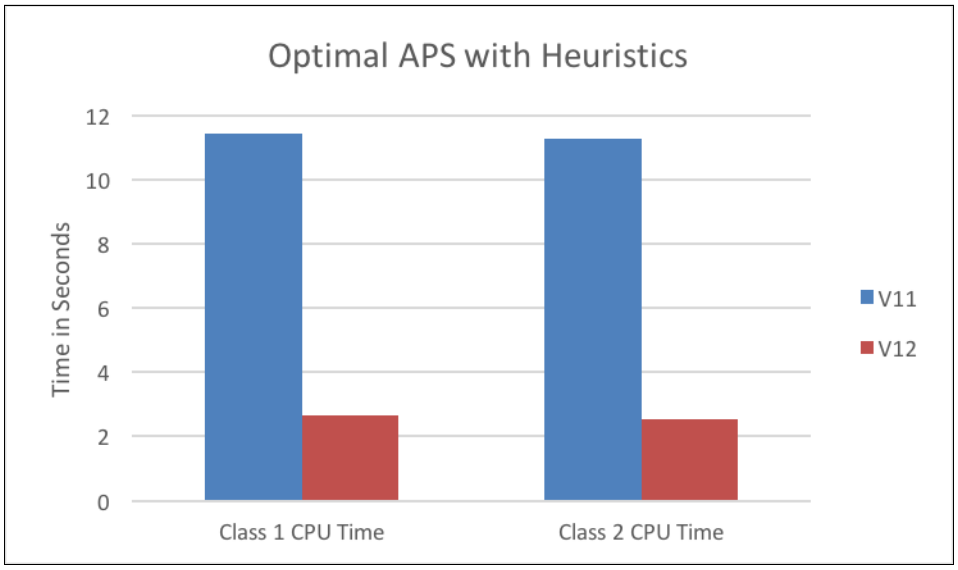


Figure 7-15 CPU time when statistics are used

The measurements show that the selected queries that matched the heuristics criteria showed a 76% reduction in the Class 1 time and a 77% reduction Class 2 time.

7.5.3 XMLTABLE enhancements

Before Db2 12, performance can be poor when XMLTABLE has many columns that are sharing the name-value pattern in their XPATHs.

In Db2 12 Functional Level 500, enhancements to the XMLTABLE function improve the performance of XMLTABLE queries, which conduct pivot-like operations to XML data with name-value pair patterns.

These enhancements are retrofitted to Db2 10 and Db2 11 with APARs PM97952 and PM98160.

Tip: For more information about the Db2 12 for z/OS Function Levels, see the chapter “Continuous delivery” in *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

Performance results

In this section, we describe several tests that were run with a different number of columns that shared the name-value pattern in their XPATHs to highlight the improvement.

The performance results are shown in Figure 7-16.

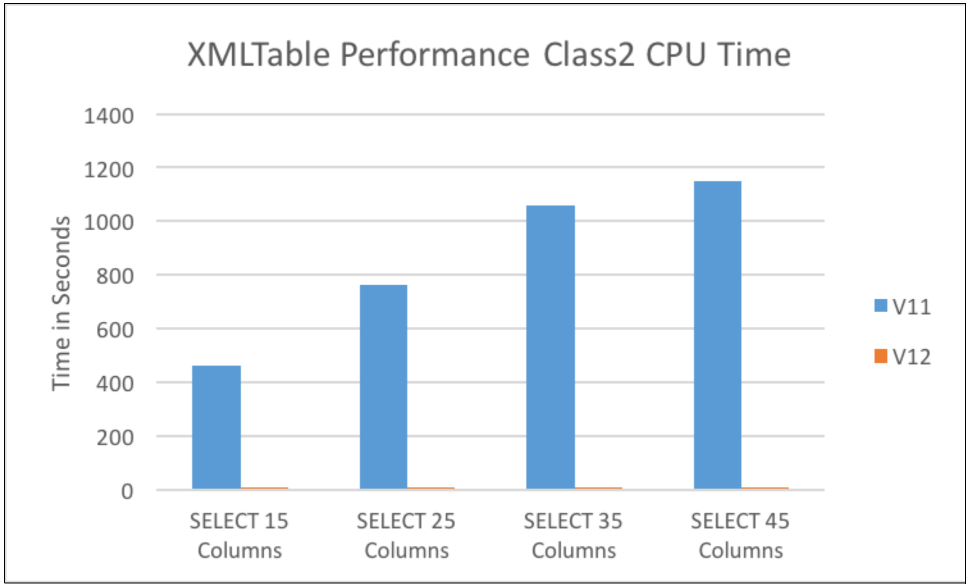


Figure 7-16 Class2 CPU time for XMLTABLE

With the new XMLTABLE enhancements, we see up to 99% savings in CPU time for queries that use this feature.

7.6 Temporal improvement

Db2 12 includes improvements in temporal referential integrity and non-deterministic expression columns for auditing.

Tip: For more information about the Db2 12 for z/OS temporal improvements, see the Temporal table enhancements section in the "Application enablement" chapter of *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

7.6.1 Temporal referential integrity

Before the activation of Db2 12 for z/OS Function Level 500, enforcing referential integrity (RI) on temporal tables with business time (application-period temporal tables) was done by using complex user-defined triggers or stored procedures. These customized routines are difficult to build and error-prone.

Db2 12 for z/OS introduces a built-in capability to define referential integrity on temporal tables with business time. Therefore, RI checking on temporal tables is not limited only to answering the question of whether a child table foreign key value is in the corresponding parent table. It also must address whether a child table foreign key value is for the specific period in the corresponding parent table.

This RI checking incurs certain overhead, but we expect that Db2 enforced RI on temporal tables will be less costly than user application enforced RI.

Performance objective

This section quantifies the overhead of RI checking (compared to no RI) in the following four main scenarios. It also compares the CPU cost of Db2 enforced RI checking by using the new Db2 12 feature with user-enforced RI checking that uses triggers:

- Update temporal parent tables
- Update temporal child tables
- Insert into temporal child tables
- Delete from temporal parent tables

Performance measurement

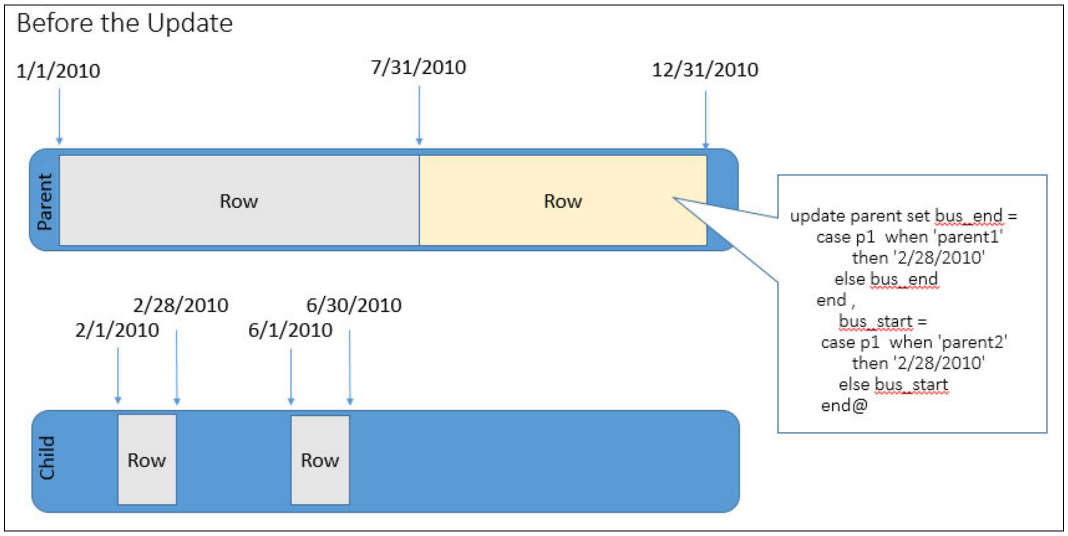
The DDL that was used to create a parent and a child temporal table that was used in the test scenarios is shown in Example 7-30.

Example 7-30 DDL for test scenario objects

```
CREATE TABLE parent
(pid INTEGER NOT NULL,
p1 VARCHAR(8),
bus_start DATE NOT NULL,
bus_end DATE NOT NULL,
PERIOD BUSINESS_TIME (bus_start, bus_end),
PRIMARY KEY (pid, BUSINESS_TIME WITHOUT OVERLAPS))@

CREATE TABLE child
(cid INTEGER NOT NULL,
pid INTEGER NOT NULL,
c1 VARCHAR(8),
bus_start DATE NOT NULL,
bus_end DATE NOT NULL,
PERIOD BUSINESS_TIME (bus_start, bus_end),
PRIMARY KEY (cid, BUSINESS_TIME WITHOUT OVERLAPS))@

CREATE INDEX idx_child_foreign_key
ON child (pid, bus_end ASC, bus_start ASC )@

Insert into parent values(1,'parent1','1/1/2010','7/31/2010')@
   Insert into parent values(1,'parent2','7/31/2010','12/31/2010')@
   Insert into child values(11,1,'chiFeb','2/1/2010','2/28/2010')@
   Insert into child values(11,1,'chiJun','6/1/2010','6/30/2010')@


-- The following ALTER statement is run only when using DB2 enforced RI

ALTER TABLE child add
FOREIGN KEY (pid, PERIOD BUSINESS_TIME)
  REFERENCES parent (pid, PERIOD BUSINESS_TIME) ON DELETE RESTRICT)@
```

Update temporal parent table

One result of updating a temporal parent table is shifting the boundary of business time periods. How the update is done is shown in Figure 7-17. The result is shown in Figure 7-18 on page 235.



```
update parent set bus_end =
  case p1  when 'parent1'
     then '2/28/2010'
    else bus_end
  end ,
    bus_start =
  case p1  when 'parent2'
     then '2/28/2010'
    else bus_start
  end@
```

Figure 7-17   Before parent table update (shifting boundaries)

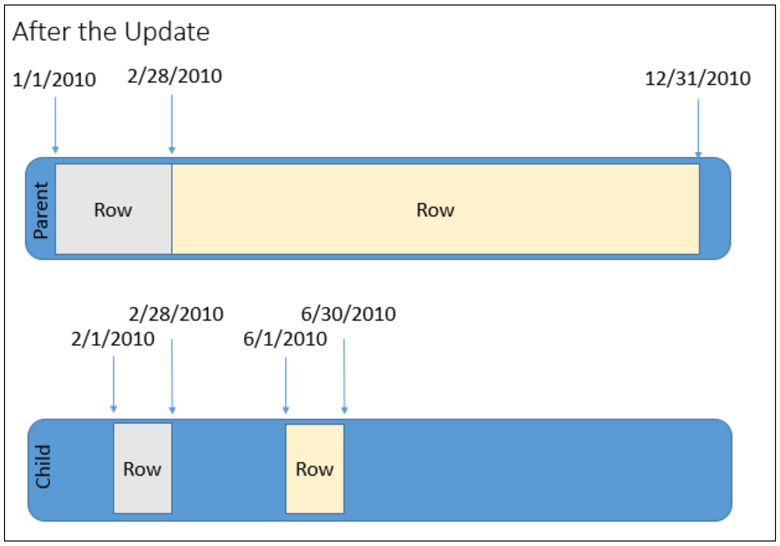
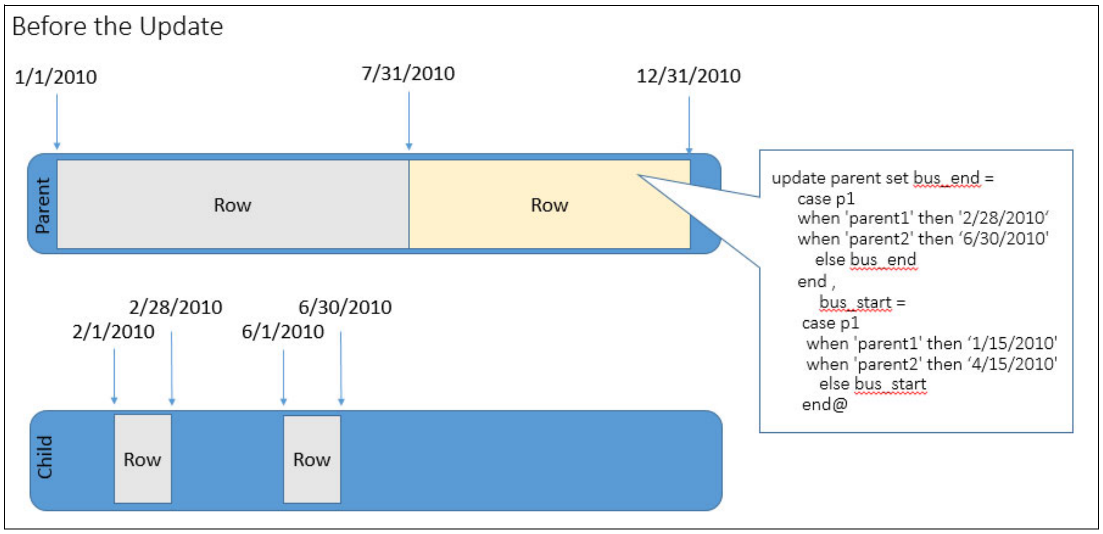


Figure 7-18 After parent table update (shifting boundaries)

Another type of parent table update results in time gaps, which triggers more Db2 processing in enforcing the parent-child relationship (see Figure 7-19 and Figure 7-20 on page 236).



```
update parent set bus_end =
    case p1
    when 'parent1' then '2/28/2010'
    when 'parent2' then '6/30/2010'
       else bus_end
    end ,
       bus_start =
     case p1
     when 'parent1' then '1/15/2010'
     when 'parent2' then '4/15/2010'
       else bus_start
     end@
```

Figure 7-19 Before parent table update (creating time gaps)

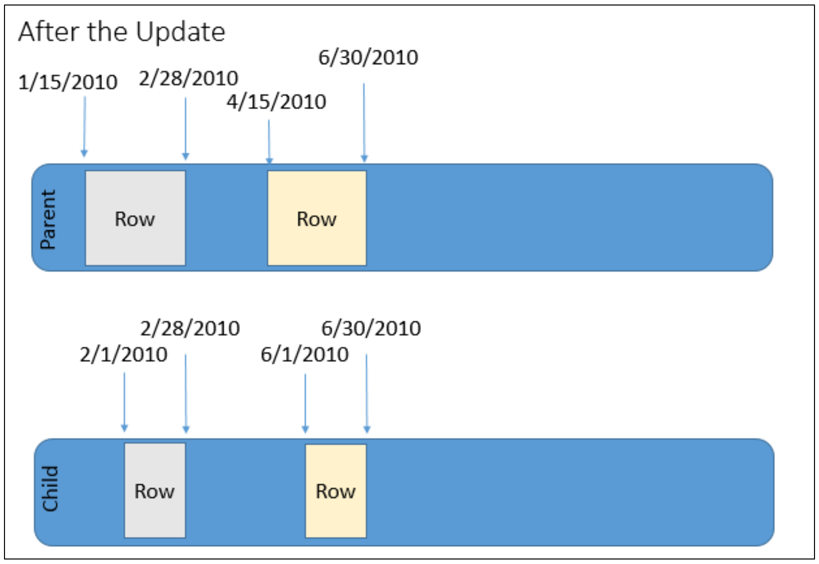
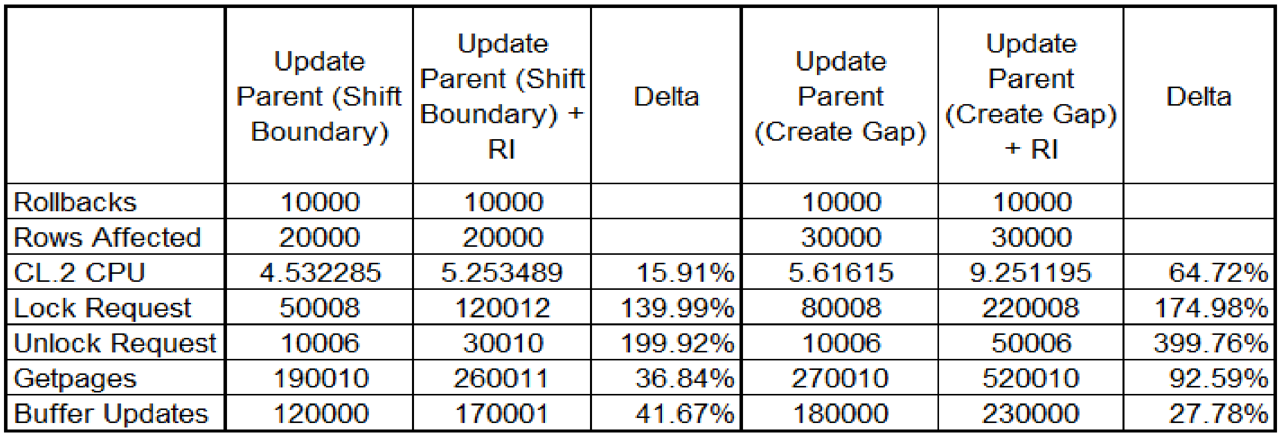


Figure 7-20 After parent table update (creating time gaps)

Our measurement of the RI overhead for the first type of boundary shifting update of the parent table is approximately 15.91% more class 2 CPU time. The second type of parent table update uses 64.72% more CPU than without RI checking to ensure that the child rows can still fit in the periods of the newly updated parent table, as shown in Figure 7-21.

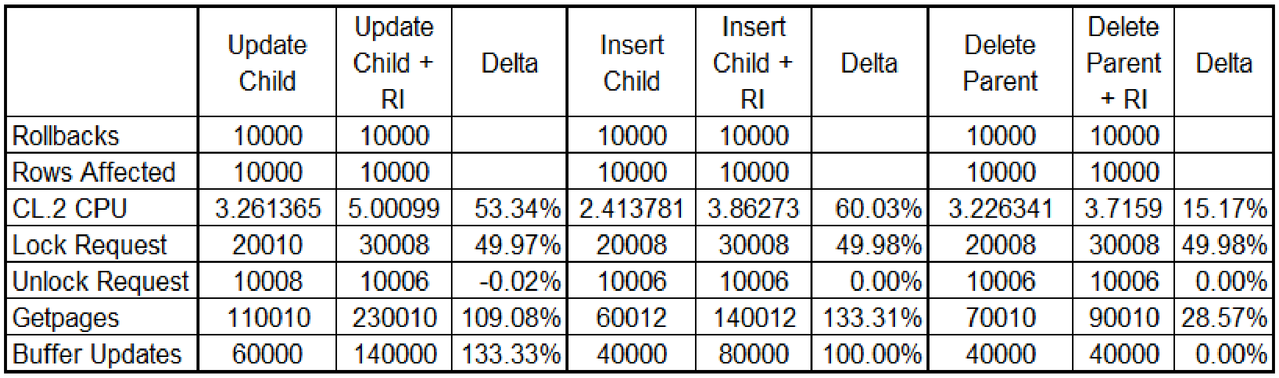


	Update Parent (Shift Boundary)	Update Parent (Shift Boundary) + RI	Delta	Update Parent (Create Gap)	Update Parent (Create Gap) + RI	Delta
Rollbacks	10000	10000		10000	10000	
Rows Affected	20000	20000		30000	30000	
CL.2 CPU	4.532285	5.253489	15.91%	5.61615	9.251195	64.72%
Lock Request	50008	120012	139.99%	80008	220008	174.98%
Unlock Request	10006	30010	199.92%	10006	50006	399.76%
Getpages	190010	260011	36.84%	270010	520010	92.59%
Buffer Updates	120000	170001	41.67%	180000	230000	27.78%

Figure 7-21 RI checking overhead when updating temporal parent table

Update and insert into temporal child table, delete from parent table

These three scenarios differ in complexity for RI checking, and as a result, the measured overhead also ranges 15.17% - 60.03%, as shown in Figure 7-22.



	Update Child	Update Child + RI	Delta	Insert Child	Insert Child + RI	Delta	Delete Parent	Delete Parent + RI	Delta
Rollbacks	10000	10000		10000	10000		10000	10000	
Rows Affected	10000	10000		10000	10000		10000	10000	
CL.2 CPU	3.261365	5.00099	53.34%	2.413781	3.86273	60.03%	3.226341	3.7159	15.17%
Lock Request	20010	30008	49.97%	20008	30008	49.98%	20008	30008	49.98%
Unlock Request	10008	10006	-0.02%	10006	10006	0.00%	10006	10006	0.00%
Getpages	110010	230010	109.08%	60012	140012	133.31%	70010	90010	28.57%
Buffer Updates	60000	140000	133.33%	40000	80000	100.00%	40000	40000	0.00%

Figure 7-22 RI checking overhead when updating and inserting into temporal child table, and deleting from parent table

Db2 enforced RI checking versus trigger enforced RI checking

To determine how efficient Db2 engine code is to enforce the RI relationship between parent and child tables, we performed an experiment in which a row was inserted into a temporal child table with Db2 enforced RI checking versus inserting that row into a child temporal table where the RI is enforced by using triggers. The trigger that was used in this scenario is shown in Example 7-31.

Example 7-31 Trigger to enforce RI

```
CREATE TRIGGER parent_child_RI_insert
   NO CASCADE BEFORE INSERT ON child
   REFERENCING NEW AS new_C
   FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC

    --  check if child row violate temporal RI
 IF NOT EXISTS (
    -- there was a row valid in PARENT when CHILD started
       SELECT *
       FROM PARENT AS P
       WHERE new_C.PID = P.PID
          AND P.BUS_START <= new_C.BUS_START
          AND new_C.BUS_START < P.BUS_END
                )
    -- there was a row valid in PARENT when CHILD ended
    OR  NOT EXISTS (
       SELECT *
       FROM PARENT AS P
       WHERE new_C.PID = P.PID
          AND P.BUS_START < new_C.BUS_END
          AND new_C.BUS_END <= P.BUS_END
                   )
    -- there are no holes in PARENT.PID during Child's period of
    -- validity
    --  the outer select find all parents convers child's range
    --    for each this parent row
    --      the subselect in the predicate find if there is no gap
    OR      EXISTS (
       SELECT *
       FROM PARENT AS P
       WHERE new_C.PID = P.PID
          AND new_C.BUS_START < P.BUS_END
          AND P.BUS_END < new_C.BUS_END
          AND NOT EXISTS (
                SELECT *
                FROM PARENT AS P2
                WHERE P2.PID = P.PID
                   AND P2.BUS_START <= P.BUS_END
                   AND P.BUS_END < P2.BUS_END
                          )
                     )
  THEN
   SIGNAL SQLSTATE '2FIOO'
   ('Child period is not covered by parent with temporal RI!');
  END IF;
END @
```

With the complexity of this trigger logic (in particular, when it must take all the different potential violations of the parent-child relationships into consideration), it is not surprising to find that Db2 enforced RI checking uses 86% less class 2 CPU compared to the user-defined trigger solution.

Temporal RI support in Db2 12 for z/OS significantly shortens the application development cycle and delivers better performance than home-grown solutions that many Db2 users depended on in the past.

7.6.2 Non-deterministic expression columns for auditing

Db2 10 for z/OS introduced a system-time versioning feature, along with temporal tables, that provides a solution to manage different versions of application data. Db2 automatically tracks when rows are created, updated, and deleted from the system-period temporal table. Columns ROW BEGIN and ROW END of the system-period temporal table are used and populated by Db2 based on the system clock.

However, this feature merely answers the question of when a change is made to the base table.

To meet the auditing requirements for regulatory compliance, starting with Db2 11 with APARs PM99683 and PI15298 applied, more columns are added to temporal tables to track who, how, and what when changes are made to the base table. These columns are automatically maintained by Db2, which avoids the need to write complex and often error-prone logic in user applications to accomplish the same goal.

Performance objective

We want to quantify the performance of populating these extra auditing columns by using this new Db2 feature, in comparison with doing so through populating non-generated columns by using triggers, which is what most users implement for their custom solution.

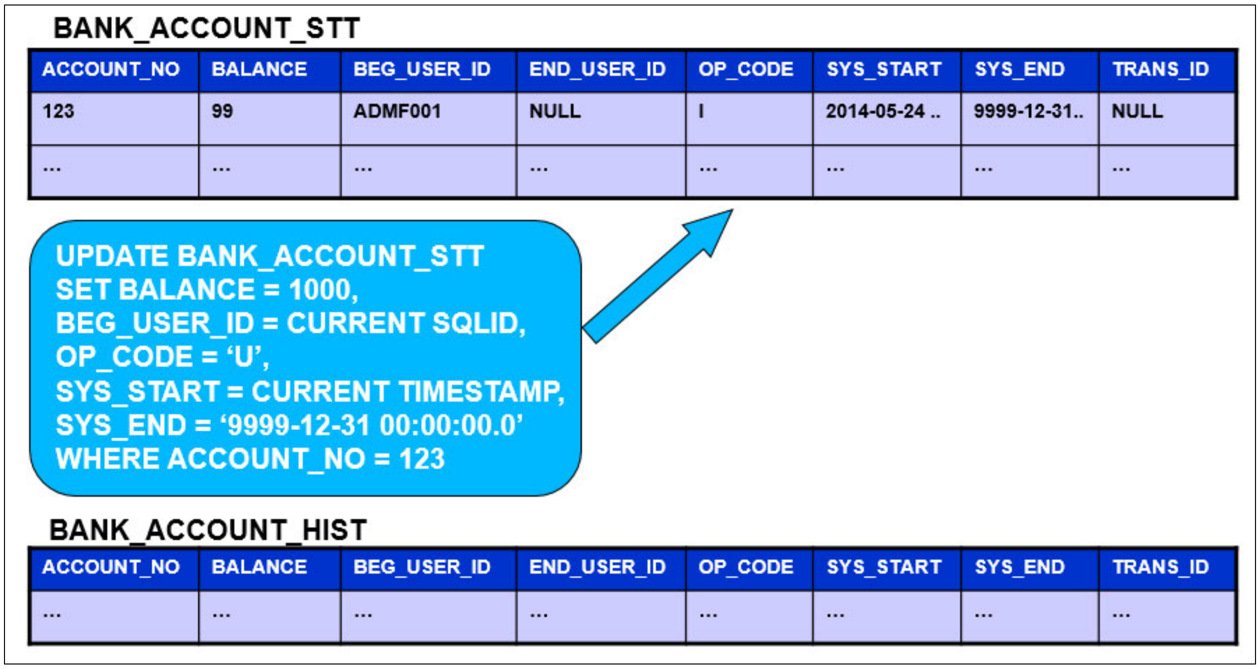
Populating auditing columns by using triggers

The DDLs that are shown in Example 7-32 are used to define a base table, history table, update trigger, and to populate the base table.

Example 7-32 DDL to define example objects

```
CREATE table bank_account_stt
(account_no INT NOT NULL,
balance INT,
beg_user_id VARCHAR(128) DEFAULT NULL,
end_user_id VARCHAR(128) DEFAULT NULL,
op_code CHAR(1) DEFAULT NULL,
sys_start TIMESTAMP(12) DEFAULT NULL,
sys_end TIMESTAMP(12) DEFAULT NULL,
trans_id TIMESTAMP(12) DEFAULT NULL)@
CREATE table bank_account_hist
(account_no INT NOT NULL,
balance INT,
beg_user_id VARCHAR(128) DEFAULT NULL,
end_user_id VARCHAR(128) DEFAULT NULL,
op_code CHAR(1) DEFAULT NULL,
sys_start TIMESTAMP(12) DEFAULT NULL,
sys_end TIMESTAMP(12) DEFAULT NULL,
trans_id TIMESTAMP(12) DEFAULT NULL)@
CREATE TRIGGER update_trig
AFTER UPDATE ON bank_account_stt
REFERENCING OLD AS OLDROW NEW AS NEWROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
INSERT INTO bank_account_hist
   (account_no,
    balance,
    beg_user_id,
    op_code,
    sys_start,
    sys_end)
VALUES (OLDROW.account_no,
        OLDROW.balance,
        OLDROW.beg_user_id,
        OLDROW.op_code,
        OLDROW.sys_start,
        CURRENT TIMESTAMP);
END@
INSERT INTO bank_account_stt (account_no, balance, beg_user_id,
end_user_id, op_code, sys_start, sys_end, trans_id)
VALUES (123, 99, 'AMDF001', NULL, 'I', CURRENT TIMESTAMP,
'9999-12-31 00:00:00.0', NULL)@
```

The base and history table that is shown in Figure 7-23 are created. Then, we run an update statement against the base table.



BANK_ACCOUNT_STT

ACCOUNT_NO	BALANCE	BEG_USER_ID	END_USER_ID	OP_CODE	SYS_START	SYS_END	TRANS_ID
123	99	ADMF001	NULL	I	2014-05-24 ..	9999-12-31..	NULL
...	...	...	...	...	...	...	...

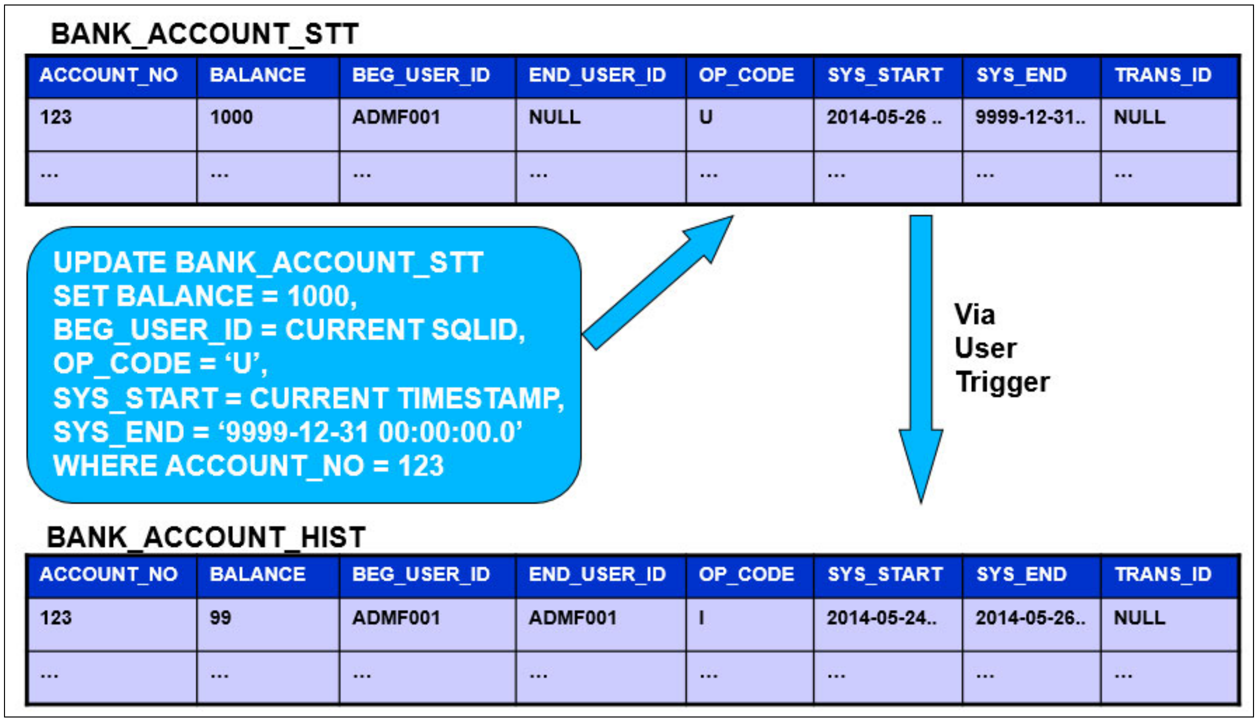
```
UPDATE BANK_ACCOUNT_STT
SET BALANCE = 1000,
BEG_USER_ID = CURRENT SQLID,
OP_CODE = 'U',
SYS_START = CURRENT TIMESTAMP,
SYS_END = '9999-12-31 00:00:00.0'
WHERE ACCOUNT_NO = 123
```

BANK_ACCOUNT_HIST

ACCOUNT_NO	BALANCE	BEG_USER_ID	END_USER_ID	OP_CODE	SYS_START	SYS_END	TRANS_ID
...	...	...	...	...	...	...	...

Figure 7-23 Before issuing update statement against the base table to start the trigger

After the UPDATE statement, the history table is populated by the trigger, as shown in Figure 7-24.



BANK_ACCOUNT_STT

ACCOUNT_NO	BALANCE	BEG_USER_ID	END_USER_ID	OP_CODE	SYS_START	SYS_END	TRANS_ID
123	1000	ADMF001	NULL	U	2014-05-26 ..	9999-12-31..	NULL
...	...	...	...	...	...	...	...

```
UPDATE BANK_ACCOUNT_STT
SET BALANCE = 1000,
BEG_USER_ID = CURRENT SQLID,
OP_CODE = 'U',
SYS_START = CURRENT TIMESTAMP,
SYS_END = '9999-12-31 00:00:00.0'
WHERE ACCOUNT_NO = 123
```

BANK_ACCOUNT_HIST

ACCOUNT_NO	BALANCE	BEG_USER_ID	END_USER_ID	OP_CODE	SYS_START	SYS_END	TRANS_ID
123	99	ADMF001	ADMF001	I	2014-05-24..	2014-05-26..	NULL
...	...	...	...	...	...	...	...

Figure 7-24 After update trigger gets started

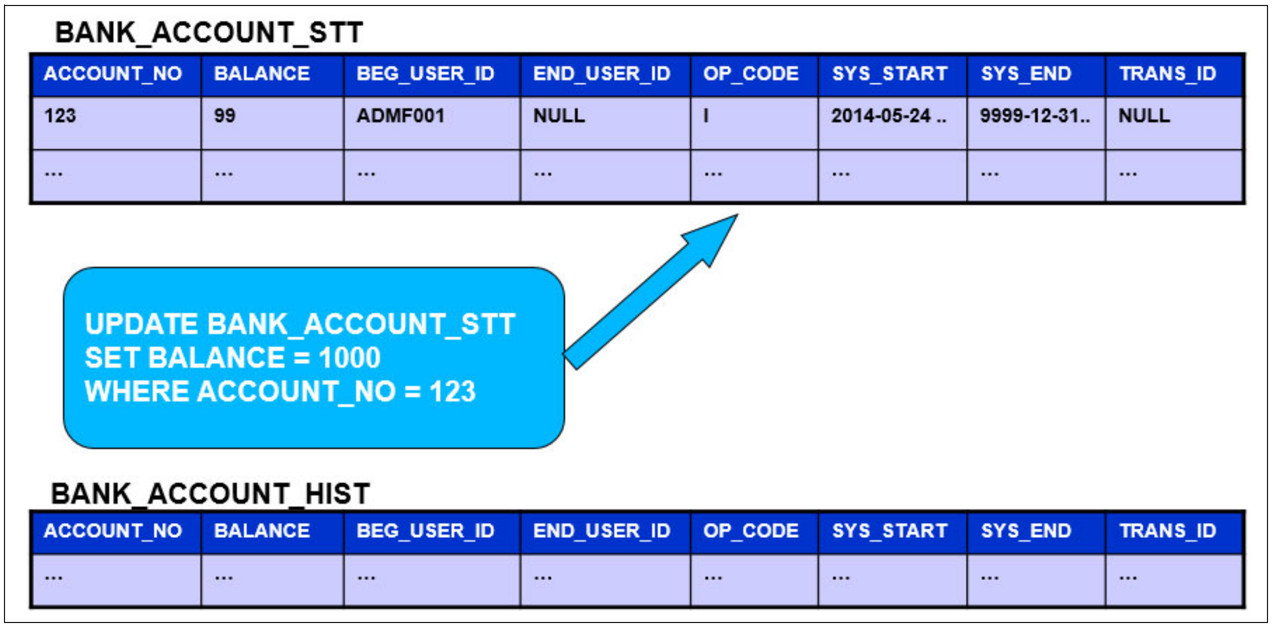
Populating auditing columns by using Db2 built-in logic

As a comparison, we use the DDLs that are shown in Example 7-33 to define a base table and history table, and to enable versioning and populate the base table.

Example 7-33 Example INSERT statements

```
CREATE TABLE BANK_ACCOUNT_STT
(ACCOUNT_NO INT NOT NULL,
BALANCE INT,
BEG_USER_ID VARCHAR(128) GENERATED ALWAYS AS (SESSION_USER),
END_USER_ID VARCHAR(128) DEFAULT NULL,
OP_CODE CHAR(1) GENERATED ALWAYS AS (DATA CHANGE OPERATION),
SYS_START TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
SYS_END TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
TRANS_ID TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
PERIOD SYSTEM_TIME(SYS_START, SYS_END))@
CREATE table bank_account_hist
(account_no INT NOT NULL,
balance INT,
beg_user_id VARCHAR(128),
end_user_id VARCHAR(128),
op_code CHAR(1),
sys_start TIMESTAMP(12) NOT NULL,
sys_end TIMESTAMP(12) NOT NULL,
trans_id TIMESTAMP(12))@
ALTER TABLE bank_account_stt ADD VERSIONING USE HISTORY TABLE bank_account_hist@
INSERT INTO bank_account_stt (account_no, balance) VALUES (123, 99)@
```

The base and history tables that are shown in Figure 7-25 are created. Then, we run an update against the base table.



BANK_ACCOUNT_STT

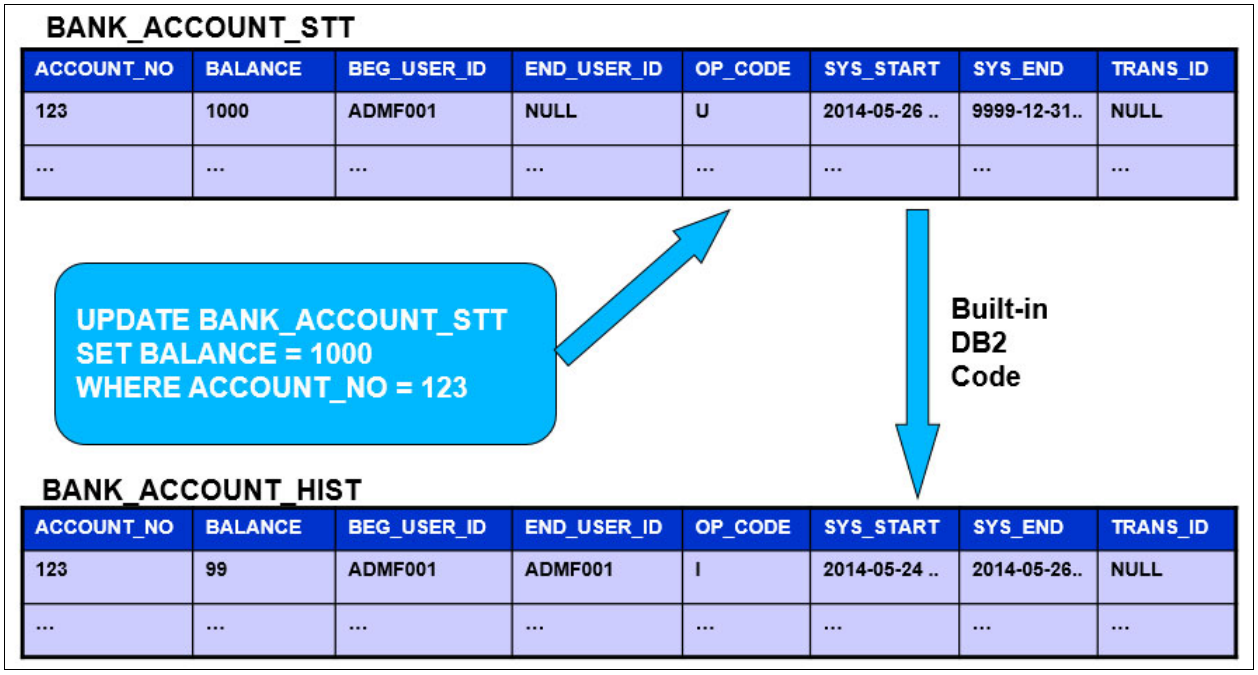
ACCOUNT_NO	BALANCE	BEG_USER_ID	END_USER_ID	OP_CODE	SYS_START	SYS_END	TRANS_ID
123	99	ADMF001	NULL	I	2014-05-24 ..	9999-12-31..	NULL
...	...	...	...	...	...	...	...

BANK_ACCOUNT_HIST

ACCOUNT_NO	BALANCE	BEG_USER_ID	END_USER_ID	OP_CODE	SYS_START	SYS_END	TRANS_ID
...	...	...	...	...	...	...	...

Figure 7-25 Before issuing update statement against the base table

Db2 populates the history table after the update statement, as shown in Figure 7-26.



BANK_ACCOUNT_STT

ACCOUNT_NO	BALANCE	BEG_USER_ID	END_USER_ID	OP_CODE	SYS_START	SYS_END	TRANS_ID
123	1000	ADMF001	NULL	U	2014-05-26 ..	9999-12-31..	NULL
...	...	...	...	...	...	...	...

```
UPDATE BANK_ACCOUNT_STT
SET BALANCE = 1000
WHERE ACCOUNT_NO = 123
```

BANK_ACCOUNT_HIST

ACCOUNT_NO	BALANCE	BEG_USER_ID	END_USER_ID	OP_CODE	SYS_START	SYS_END	TRANS_ID
123	99	ADMF001	ADMF001	I	2014-05-24 ..	2014-05-26..	NULL
...	...	...	...	...	...	...	...

Figure 7-26 After updating the base table

Performance comparison

Per the Db2 accounting report, the use of the built-in Db2 support for maintaining the extra auditing columns uses 14% less class 2 CPU than doing so by using user-defined triggers.

The performance result agrees with our initial expectation, which indicates that users can save on CPU processing time by using this new auditing column feature to reduce the complexity in user application development.

8

Query performance

Db2 12 introduces many enhancements in query performance, including enhancements in optimal access path selection, query transformation, and internal runtime processing.

This chapter includes the following topics:

8.1 UNION ALL and OUTER JOIN performance enhancements

Performance challenges are often observed when UNION ALL or OUTER JOIN is used in a part of a query. Although the UNION ALL construct and outer joins might not always be combined in the same queries, significant overlap occurs with the performance challenges of both such that it makes sense to discuss them together.

However, a conclusion should not be drawn that all queries that involve UNION ALL or outer joins include performance challenges. The performance opportunities exist where these constructs involve materializations in which filtering is not applied early in the query execution. Older releases of Db2 delivered enhancements for both constructs such that performance might be acceptable for simple use cases.

An OUTER JOIN is generally used when the join relationship between tables is optional, whereas an inner join is used to maintain a mandatory join relationship. The expectation is typically that performance is similar between inner joins and outer joins. Although it might be possible to have similar performance, comparable performance can also be challenging, most notably because of the difference in the result that is retrieved from the optional join relationship, and that outer joins dictate the sequence that tables must be joined.

UNION ALL is used to combine different tables and allow them to be represented as a consistent result. This construct is appearing in many new applications, including SAP Core Data Services and WebSphere Application Server Portal, and is also used internally by Db2 for system-period temporal tables (available since Db2 10) and the archive transparency feature (since Db2 11).

The Db2 usage drives more customers to be exposed to UNION ALL within their queries. Customers are also integrating active and archive tables in their own applications by using UNION ALL to supplement or alleviate the management challenges of large table spaces.

Customers expect outer joins to perform like inner joins. Similarly, when customers use UNION ALL to consolidate multiple objects, they expect performance similar to if they accessed a single object. This performance expectation can be far more challenging to achieve because of the increase in the number of objects that are involved when UNION ALL is introduced.

In Db2 12 for z/OS, improving UNION ALL and OUTER JOIN performance to address materialization challenges include the following the targeted enhancements:

- Reordering OUTER JOIN tables to avoid materializations
- Push predicates inside UNION ALL legs or OUTER JOIN query blocks
- Sort of outer table to ensure sequential access to the inner table of a join
- Bypass work file usage when materialization is required
- Trim unnecessary columns and tables
- Push ORDER BY and FETCH FIRST into UNION ALL legs

These targeted enhancements are described in the following sections. The reader can skip these sections to the UNION ALL and OUTER JOIN summary (8.1.5, “Targeted UNION ALL and OUTER JOIN summary” on page 257), or read through the next sections to understand each enhancement and why they were targeted in Db2 12.

8.1.1 Reorder OUTER JOIN tables to avoid materializations

Allowing Db2 to internally reorder the OUTER JOIN tables within the query overcomes a limitation that can be exposed when combining outer and inner joins in the same query. In certain instances, Db2 11 and older versions materialize some tables that can result in local or join filtering not being applied before the materialization.

Some customers who were observed this performance challenge rewrote their queries to ensure that all inner joins appear before OUTER JOINs in the query, if possible. Rewriting a query is often difficult because of the proliferation of generated queries and applications being deployed without thorough performance evaluation. Therefore, minimizing exposure to this limitation in Db2 12 provides a valuable performance boost for affected queries.

To demonstrate this enhancement, the query that is shown in Example 8-1 materializes the view in Db2 11, but Db2 12 can avoid materialization of the view, which greatly improves the performance of the query.

Example 8-1 View materialization avoidance

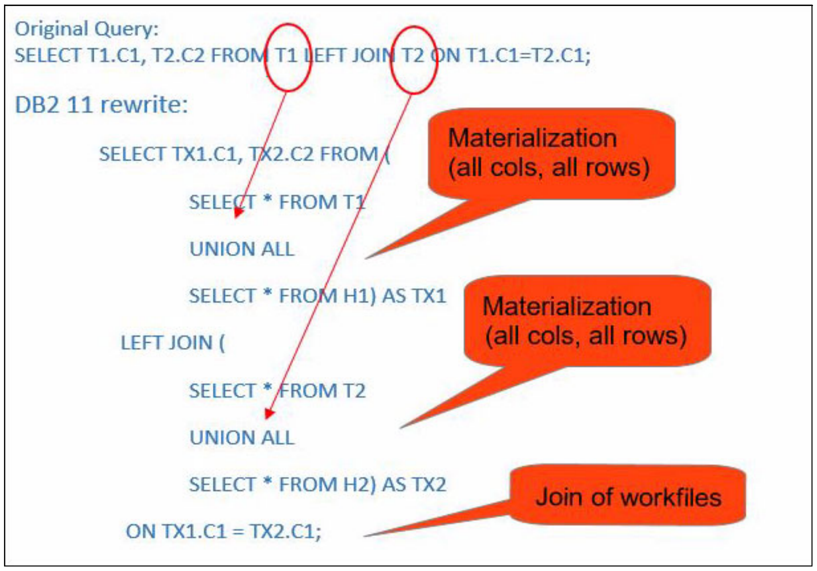
```
CREATE VIEW TPCHVLIL  AS
   SELECT *
      FROM CUSTOMER
       LEFT OUTER JOIN NATION
         ON C_NATIONKEY=N_NATIONKEY
       INNER JOIN SUPPLIER
         ON S_SUPPKEY=C_CUSTKEY
       LEFT OUTER JOIN PART
         ON C_CUSTKEY=P_PARTKEY
     WHERE S_SUPPKEY BETWEEN 0 AND 900000;

SELECT *
  FROM TPCHVLIL
   INNER JOIN LINEITEM
     ON C_CUSTKEY=L_ORDERKEY WHERE C_CUSTKEY < 900000;
```

Performance results

Performance results for reordering OUTER JOIN tables showed up to 90% reduction in CPU for queries that now avoids costly materializations.

8.1.2 Push predicates inside UNION ALL legs and OUTER JOIN query blocks

The example that is shown in Figure 8-1 demonstrates performance challenges when UNION ALL is combined with an OUTER JOIN. The original query appears at the top of Figure 8-1 and is a simple two table (left outer) join of T1 to T2.



```
Original Query:
SELECT T1.C1, T2.C2 FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1;

DB2 11 rewrite:

        SELECT TX1.C1, TX2.C2 FROM (
                SELECT * FROM T1
                UNION ALL
                SELECT * FROM H1) AS TX1
          LEFT JOIN (
                SELECT * FROM T2
                UNION ALL
                SELECT * FROM H2) AS TX2
          ON TX1.C1 = TX2.C1;
```

Materialization (all cols, all rows)
Materialization (all cols, all rows)
Join of workfiles

Figure 8-1 Db2 11 left OUTER JOIN query with archive transparency tables

In this example, T1 and T2 are “archive enabled” (which refers to the Db2 11 archive transparency feature). Therefore, Db2 rewrites the query to include the active and archive tables (with T1 circled and the arrow pointing to the first UNION ALL on the left side of the join, and T2 circled with arrow pointing to the second UNION ALL on the right side of the join).

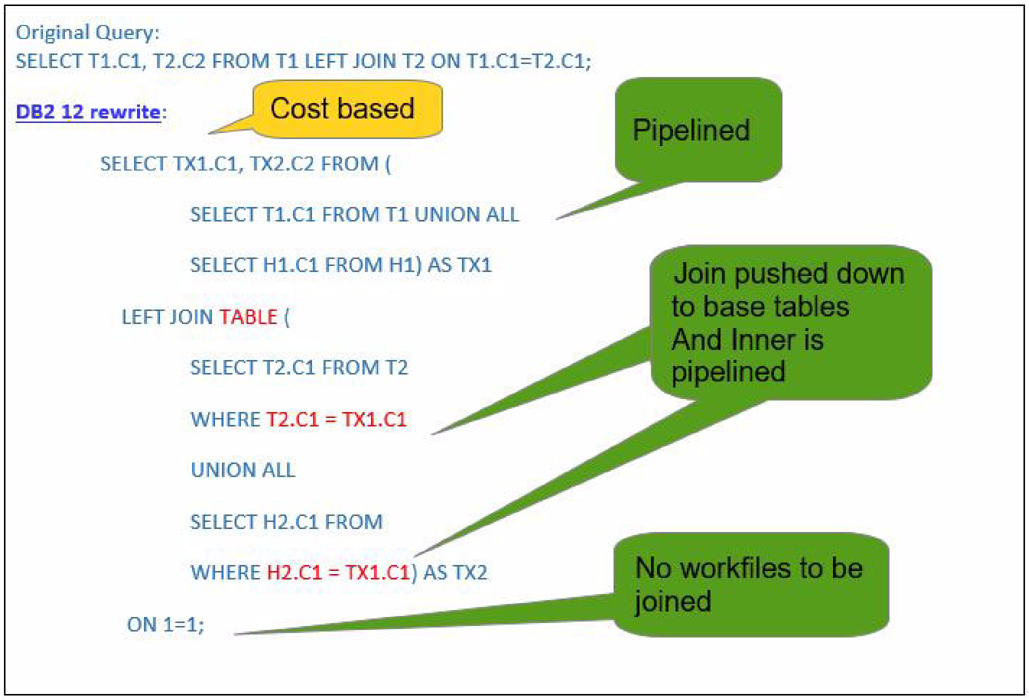
This representation is true of any UNION ALL within a view, where the view definition is replaced within the query where it is referenced. Although older releases provided similar merge (and therefore materialization avoidance) capabilities for table functions that were syntactically equivalent to views, Db2 12 improved the merge of deterministic table functions with input parameters and indexability of input parameters as predicates within the table function, as described in 8.2, “Predicate optimizations” on page 257.

The performance challenge for this query example is that Db2 runs each leg of the UNION ALL separately and combines the results from each side of the first UNION ALL. Then, it combines each side of the second UNION ALL before joining the two results as requested in the outer query block.

In the V11 rewrite of the query, no join predicates or any filtering within the UNION ALL legs are used. The term *combining* means that Db2 returns all columns and all rows from T1 in the first UNION ALL and all columns and all rows from H1 and materialize those rows to a work file. The work file then is sorted in preparation for the join. This process is repeated for the second UNION ALL; that is, all columns and all rows from T2 and H2 are materialized into a work file and sorted in preparation for the join on column C1 from both work files.

The performance of this query depends heavily on the size of the tables that are involved, with large tables that use significant CPU and work file resources to complete the process.

By using the same example from Figure 8-1 on page 246, we can use Figure 8-2 to describe how several of the UNION ALL-focused enhancements in Db2 12 can improve the performance of this query.



```
Original Query:
SELECT T1.C1, T2.C2 FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1;

DB2 12 rewrite:

        SELECT TX1.C1, TX2.C2 FROM (
                SELECT T1.C1 FROM T1 UNION ALL
                SELECT H1.C1 FROM H1) AS TX1
          LEFT JOIN TABLE (
                SELECT T2.C1 FROM T2
                WHERE T2.C1 = TX1.C1
                UNION ALL
                SELECT H2.C1 FROM
                WHERE H2.C1 = TX1.C1) AS TX2
          ON 1=1;
```

Cost based
Pipelined
Join pushed down to base tables And Inner is pipelined
No workfiles to be joined

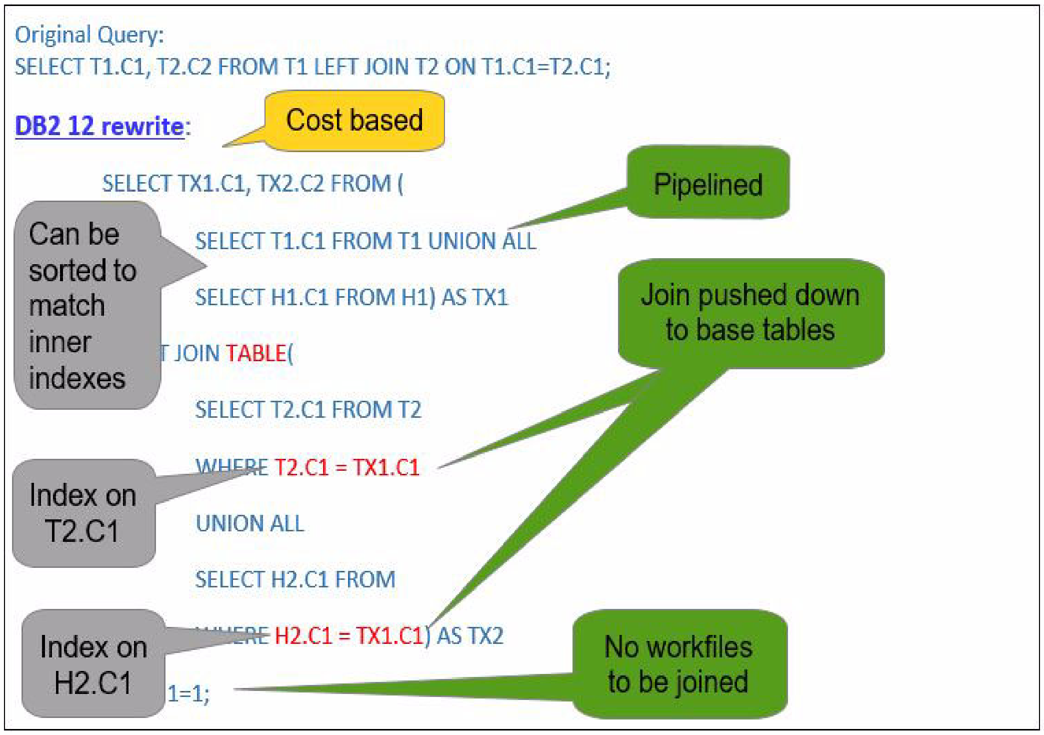
Figure 8-2 Db2 12 left OUTER JOIN query with archive transparency tables (or any UNION ALL views)

Although the internal rewrite of the tables to the UNION ALL representation remains the same, Db2 12 adds the ability for the optimizer to make a cost-based decision whether to push the join predicates into the UNION ALL legs. As shown in Figure 8-2, the optimizer chose to push the join predicates inside each UNION ALL leg. The join predicates occur only on the right side of the join because a join is "FROM" the left "TO" the right. The TABLE keyword is required externally for this example to be syntactically valid because pushing the predicates down results in the query appearing as a correlated table expression.

Having the join predicates in each UNION ALL leg allows the join to "look up" each leg of the UNION ALL for every row coming from the outer, rather than sort the full results for the join.

Another optimization to reduce work file usage and materialization, is that Db2 12 can "pipeline" the rows from the first UNION ALL (on the left side of the join). (The term *pipelining* describes a technique where Db2 transfers rows directly from one query block to another without materializing one or both query blocks.) Transferring rows from one query block to another can require the results to be written (materialized) to an intermediate work file as in the Db2 11 example. However, Db2 12 can pipeline the result from the first UNION ALL to the join without requiring this materialization.

The result for this query in Db2 12 given the (cost-based) join predicate pushdown is that the work file and materializations are avoided, and available indexes can be used for each leg, as shown in Figure 8-3.



Original Query:

```
SELECT T1.C1, T2.C2 FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1;
```

DB2 12 rewrite:

```
        SELECT TX1.C1, TX2.C2 FROM (
              SELECT T1.C1 FROM T1 UNION ALL
              SELECT H1.C1 FROM H1) AS TX1
        T JOIN TABLE(
              SELECT T2.C1 FROM T2
              WHERE T2.C1 = TX1.C1
              UNION ALL
              SELECT H2.C1 FROM
              WHERE H2.C1 = TX1.C1) AS TX2
        1=1;
```

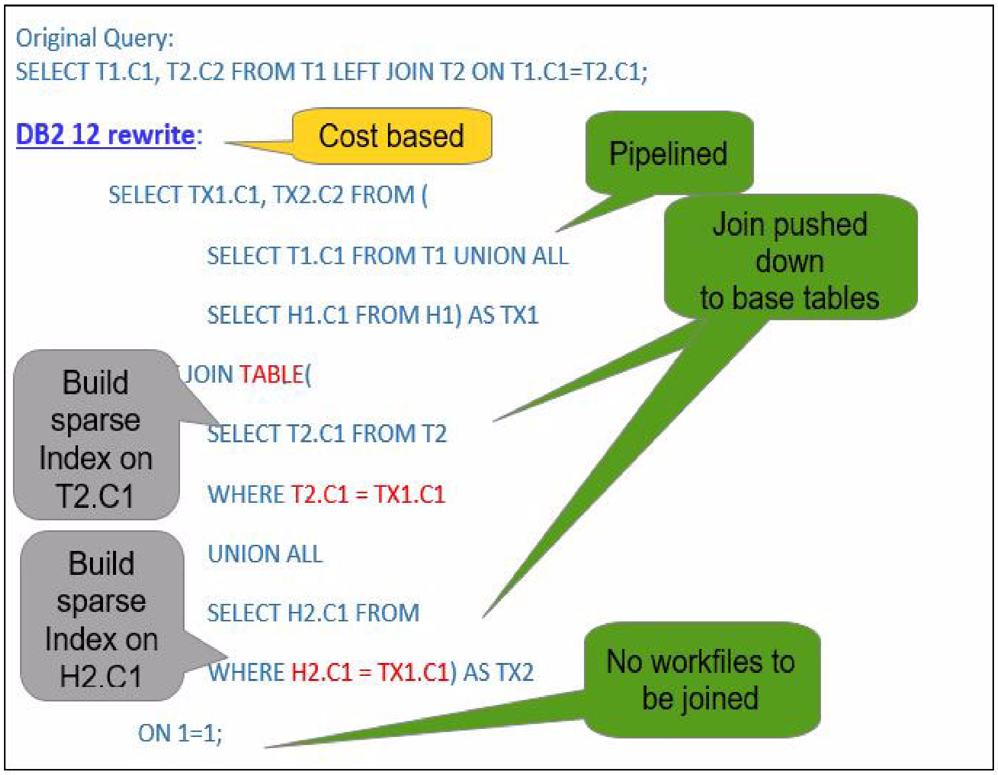
Cost based
Pipelined
Can be sorted to match inner indexes
Join pushed down to base tables
Index on T2.C1
Index on H2.C1
No workfiles to be joined

Figure 8-3 Db2 12 left OUTER JOIN UNION ALL query with sort of outer

We extend the example that is shown in Figure 8-3 further to demonstrate more Db2 12 enhancements that can improve performance closer to queries that do not use the UNION ALL infrastructure.

One option that the optimizer can use to improve join performance between two tables sequentially is for the optimizer to introduce a sort of the outer (composite, which is denoted as SORTC_JOIN='Y' in the explain output). If data from the outer table is accessed in a different sequence than the index that is used for the join to the inner, Db2 can choose to sort the outer into the sequence of the inner, which allows the access to the inner to occur in order. This approach is extended in Db2 12 when joining to views and table expressions that contain UNION ALL and is a cost-based choice for the optimizer.

Figure 8-4 on page 249 shows another variation of the same query in which no supporting indexes for the join to T2.C1 or H2.C1 are used. Db2 12 allows a sparse index to be built on an individual leg of a UNION ALL. This index applies to one or both legs, and is also cost-based and therefore can be combined with the other UNION ALL-focused enhancements.



```
Original Query:
SELECT T1.C1, T2.C2 FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1;

DB2 12 rewrite:

        SELECT TX1.C1, TX2.C2 FROM (
                SELECT T1.C1 FROM T1 UNION ALL
                SELECT H1.C1 FROM H1) AS TX1
                JOIN TABLE(
                SELECT T2.C1 FROM T2
                WHERE T2.C1 = TX1.C1
                UNION ALL
                SELECT H2.C1 FROM
                WHERE H2.C1 = TX1.C1) AS TX2
          ON 1=1;
```

Cost based
Pipelined
Join pushed down to base tables
Build sparse Index on T2.C1
Build sparse Index on H2.C1
No workfiles to be joined

Figure 8-4 Db2 12 left OUTER JOIN UNION ALL query without supporting join indexes

If sparse index is chosen, the outer does not need to be sorted into join order as shown in Figure 8-3 on page 248 because a sparse index can use hashing if the result can be contained within memory. Therefore, random I/O is not a significant concern.

Performance results for join predicate pushdown

The following performance results were observed:

- Up to 95% reduction in CPU time for qualifying queries.
- WebSphere Application Server Portal subset: An 80% CPU time reduction for qualifying queries.

Performance results for avoiding inner table materializations (pipelined inner)

The following performance results were observed:

- Crystal Reports: A 5% reduction in CPU and 65% reduction in work file getpages.
- Up to 80% reduction in CPU and elapsed time for qualifying queries.
- When materialization of the inner table is avoided because of pipelining, the PLAN_TABLE indicates this result by using a new ACCESSTYPE='O' option.

Performance results for pipelined outer

The following performance results were observed:

- A 10 - 30% CPU reduction for various workload queries.
- Up to 80% reduction in elapsed time because of improved parallelism by avoiding materialization to work file for outer table.
- When materialization of the outer table is avoided because of pipelining, the PLAN_TABLE indicates this result by using a new ACCESSTYPE='O' option.

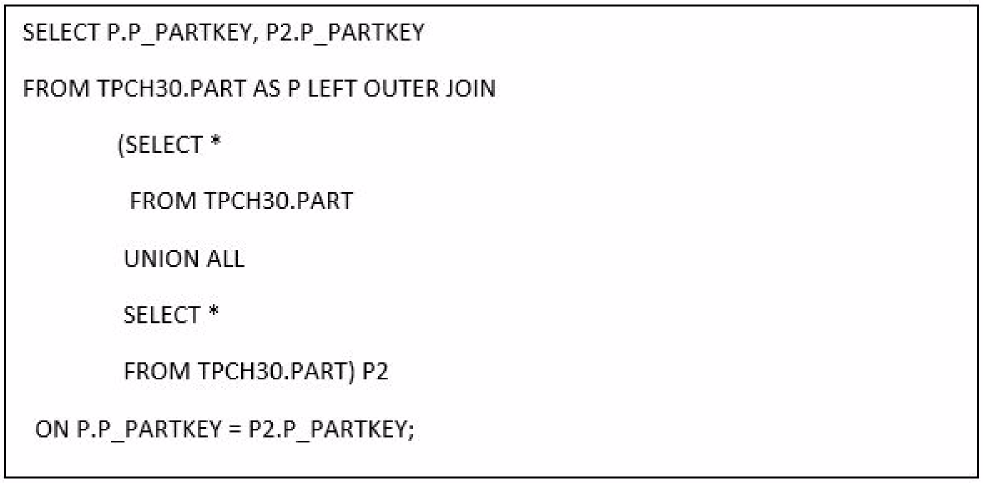
8.1.3 Trim unnecessary columns and tables

In the first UNION ALL example that is shown in Figure 8-1 on page 246, it was highlighted that all columns and all rows were materialized. The next examples focus on the optimizations to improve the available access path choices to avoid materializing all rows of the view or table expression.

However, an explanation of the all columns was intentionally avoided until the enhancement was described. But first, the problem: For materialized views or table expressions where the select list can contain more columns than are needed by the query that is referencing that view or table expression, all columns that are coded within that view or table expression are included as part of the materialization in Db2 11 and older versions when materialization occurs. Db2 removes unreferenced columns from the select list if a view or table expression is merged with the referencing query.

Column pruning

Db2 12 provides an enhancement to remove (prune) columns from the select list of the view or table expression before materialization for any columns that are not required by the referencing query. This removal reduces the size of the materialized result and the amount of work file space and sort key size, which can allow index-only access for access before materialization. Such an enhancement can also apply to join predicate pushdown in which materialization can be avoided, as shown in Figure 8-5 and Figure 8-6 on page 251.

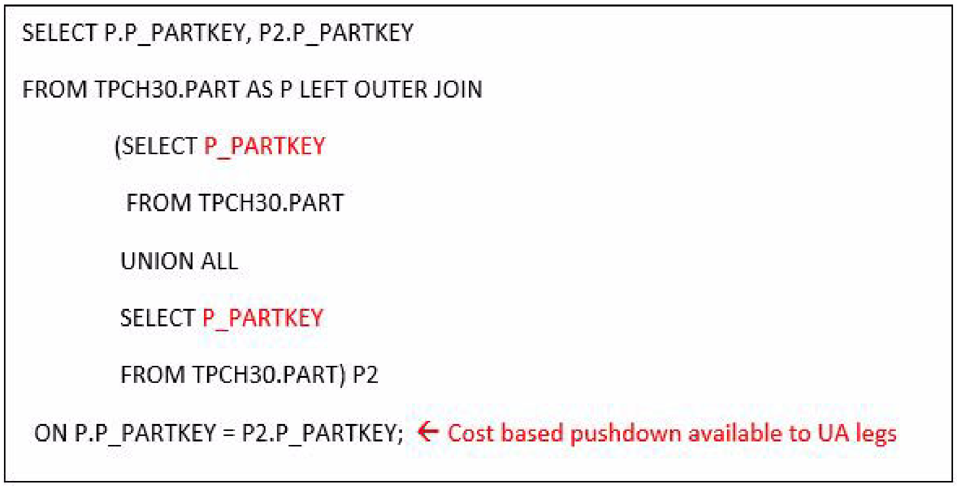


```
SELECT P.P_PARTKEY, P2.P_PARTKEY

FROM TPCH30.PART AS P LEFT OUTER JOIN

          (SELECT *

           FROM TPCH30.PART

          UNION ALL

          SELECT *

          FROM TPCH30.PART) P2

 ON P.P_PARTKEY = P2.P_PARTKEY;
```

Figure 8-5 Db2 11 processing all columns in materialized view/table expression

The query that is shown in Figure 8-5 involves a LEFT OUTER JOIN to a UNION ALL table expression. In the select list for each leg of the UNION ALL is a SELECT *, which means return all columns from that table.

However, the referencing SELECT requires P2.P_PARTKEY only (for the SELECT list and ON clause). Because of the materialization of the UNION ALL table expression in Db2 11, all columns from the PART table (for example, if the table has 10 columns) are accessed and materialized to the work file. As the outer query is retrieving from that work file, only P_PARTKEY is retrieved.



```
SELECT P.P_PARTKEY, P2.P_PARTKEY

FROM TPCH30.PART AS P LEFT OUTER JOIN

          (SELECT P_PARTKEY

           FROM TPCH30.PART

           UNION ALL

           SELECT P_PARTKEY

           FROM TPCH30.PART) P2

 ON P.P_PARTKEY = P2.P_PARTKEY; ← Cost based pushdown available to UA legs
```

Figure 8-6 Db2 12 pruning unreferenced columns with optional join pushdown

The example that is shown in Figure 8-5 on page 250 is repeated in Figure 8-6. Db2 12 prunes the unnecessary columns and requires only P_PARTKEY to be returned from each UNION ALL leg, whereas Db2 11 returned all columns. If the UNION ALL table expression is materialized, only P_PARTKEY is retrieved and written or materialized to work file compared with all (10) columns in Db2 11.

If P_PARTKEY is indexed, the optimizer can choose a non-matching index scan in Db2 12 rather than a table space scan in Db2 11. Similarly, if the join predicates were pushed down in Db2 12 and matching index access was chosen on P_PARTKEY, index-only is possible because only P_PARTKEY is required.

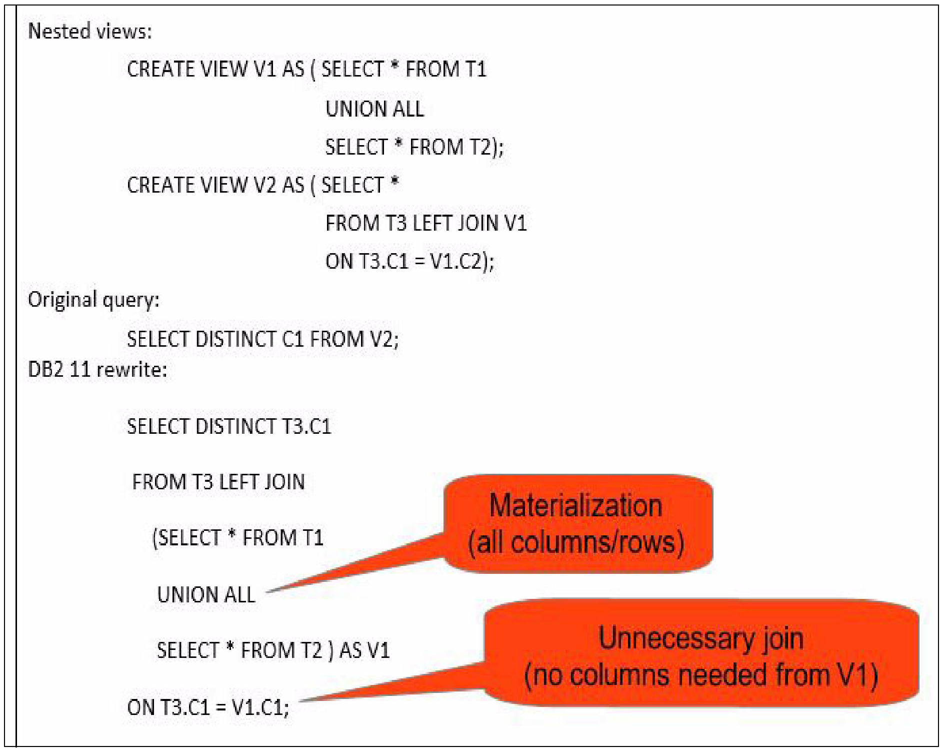
Table pruning

Another benefit of column pruning is that it can open other table pruning opportunities. In Db2 10, an enhancement was delivered to prune OUTER JOIN tables if no columns were required from that table and it was guaranteed that the table did not introduce duplicates. In Db2 12, this pruning enhancement is extended from OUTER JOINs to views or table expressions that are guaranteed not to return duplicates and no columns are required from the view or table expression.

The example that is shown in Figure 8-7 on page 252 might appear convoluted, but it is a simplified example from an actual ERP workload on Db2 for z/OS. Before explaining the example, it is an interesting question to ask why you might build a query that references tables from which columns are not actually required.

This type of construct is becoming more common as views are built to hide the complexity and present a simplified representation to the user, such that they can select from a view, rather than writing a complex query themselves. Similarly, for query generators it is required that the select from the views and the underlying table relationships can be set up or altered without affecting the query generation.

Such views can be built on top of each other as building blocks for simplicity in constructing these respresentations and to simplify future changes at any level. This ability means that the views typically contain more than what you might require, and you can select only the columns that you need and ultimately, you might need only a subset of the tables.



Nested views:

```
        CREATE VIEW V1 AS ( SELECT * FROM T1
                            UNION ALL
                            SELECT * FROM T2);
        CREATE VIEW V2 AS ( SELECT *
                            FROM T3 LEFT JOIN V1
                            ON T3.C1 = V1.C2);
```

Original query:

```
        SELECT DISTINCT C1 FROM V2;
```

DB2 11 rewrite:

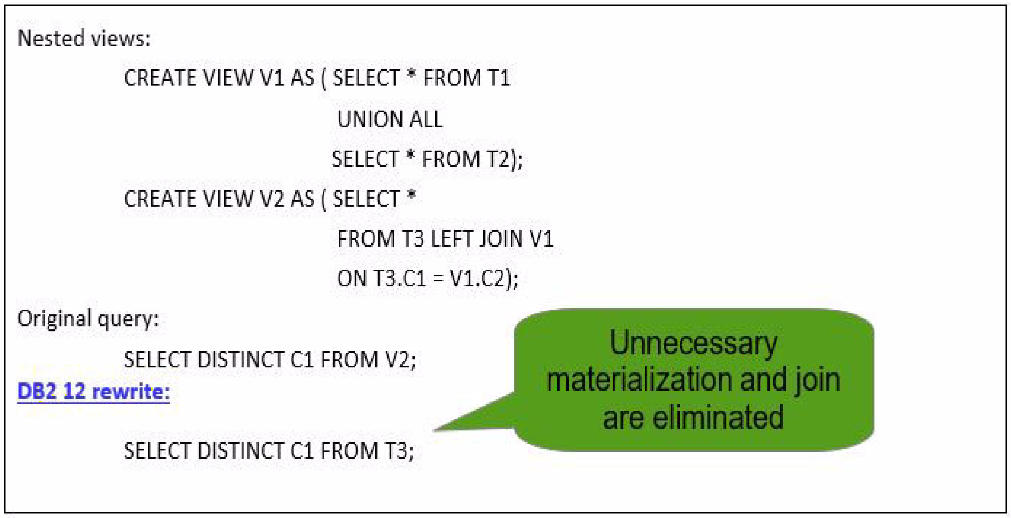
```
        SELECT DISTINCT T3.C1

         FROM T3 LEFT JOIN

           (SELECT * FROM T1

            UNION ALL

            SELECT * FROM T2 ) AS V1

        ON T3.C1 = V1.C1;
```

Materialization (all columns/rows)
Unnecessary join (no columns needed from V1)

Figure 8-7 Db2 11 LEFT OUTER JOIN to unnecessary table expression

Figure 8-7 shows an example of how Db2 11 handles this type of nested view which conceals the underlying complexity. As with past UNION ALL examples, the UNION ALL retrieves all columns and all rows from T1 and T2 and materializes these columns and rows to a work file, which is sorted for the join. T3 is then joined to this materialized result, and the result of this final join is sorted to remove duplicate C1 values (given the DISTINCT). This join is unnecessary because no columns were required from the UNION ALL of T1 and T2 and the DISTINCT removes any duplicates introduced.

The Db2 12 rewrite for the example that is shown in Figure 8-7 on page 252 is demonstrated in Figure 8-8. The view definitions and the query against the views are the same in both figures. What differs is how Db2 12 can prune out the table expression that contains the UNION ALL. The result is a SELECT DISTINCT that requires access to T3 only.



Nested views:

```
CREATE VIEW V1 AS ( SELECT * FROM T1
                    UNION ALL
                    SELECT * FROM T2);
CREATE VIEW V2 AS ( SELECT *
                    FROM T3 LEFT JOIN V1
                    ON T3.C1 = V1.C2);
```

Original query:

```
SELECT DISTINCT C1 FROM V2;
```

DB2 12 rewrite:

```
SELECT DISTINCT C1 FROM T3;
```

Unnecessary materialization and join are eliminated

Figure 8-8 Db2 12 LEFT OUTER JOIN to unnecessary table expression

Performance results for trimming unnecessary columns and tables

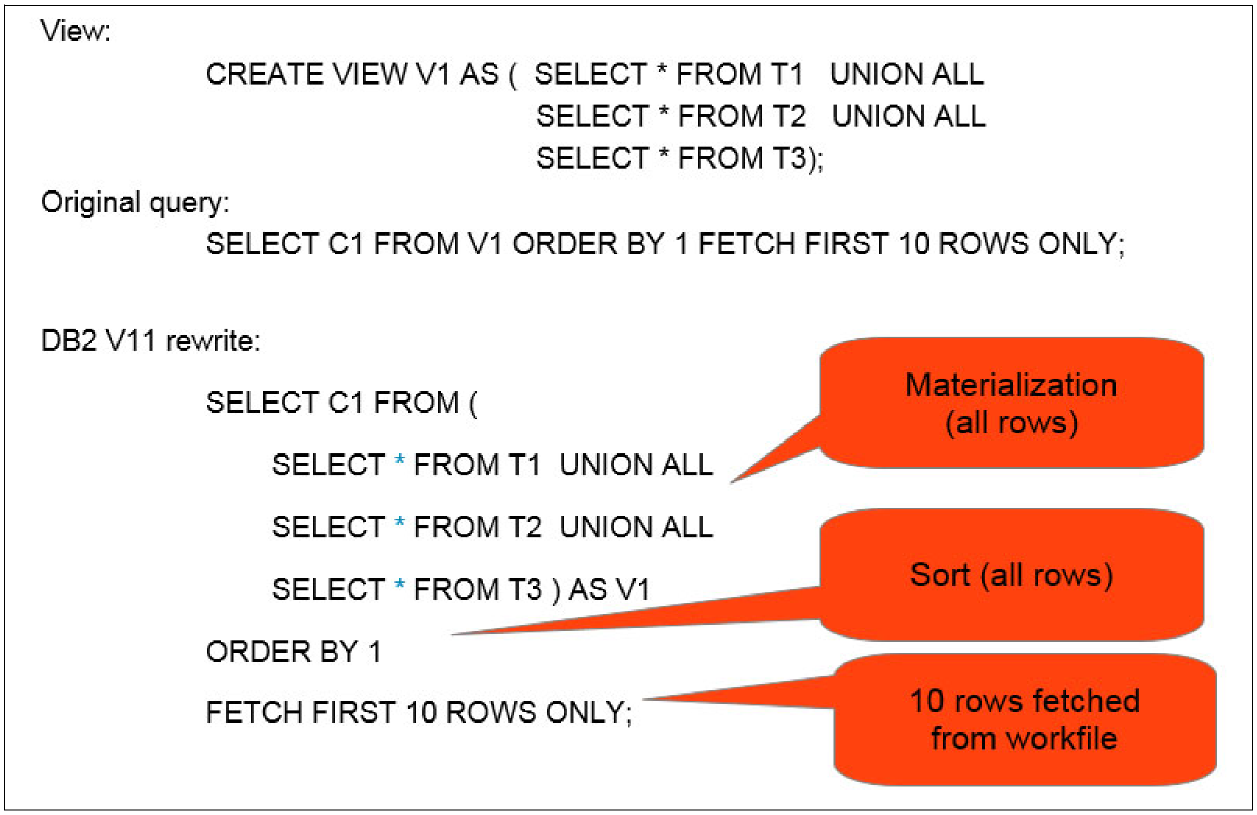
The following performance results were observed:

- Up to 5% reduction in CPU and 35% reduction in elapsed time with column pruning alone.
- Up to 30% reduction in CPU when trimmed columns result in table pruning.

8.1.4 Push down of ORDER BY and FETCH FIRST

The final Db2 12 enhancement targeted at improving UNION ALL and OUTER JOINs with views and table expressions relates to the usage of ORDER BY and FETCH FIRST clauses.

In the example that is shown in Figure 8-9 on page 254, view V1 contains three tables combined with UNION ALL, and then a query against that view includes an ORDER BY 1 (meaning the first column in the select list) and FETCH FIRST 10 ROWS ONLY. Figure 8-9 on page 254 shows the view definition, the query, and the merged query in Db2 11. From a performance perspective, all rows (and columns) from all three UNION ALL legs will be retrieved and materialized to a work file, and then sorted for the ORDER BY and from that sort, the top 10 rows will be retrieved. If each of the tables contains 1 million rows, then the materialization and sort will involve 3 million rows to retrieve just the top 10 rows.


View:

```
CREATE VIEW V1 AS (  SELECT * FROM T1   UNION ALL
                     SELECT * FROM T2   UNION ALL
                     SELECT * FROM T3);
```

Original query:

```
SELECT C1 FROM V1 ORDER BY 1 FETCH FIRST 10 ROWS ONLY;
```

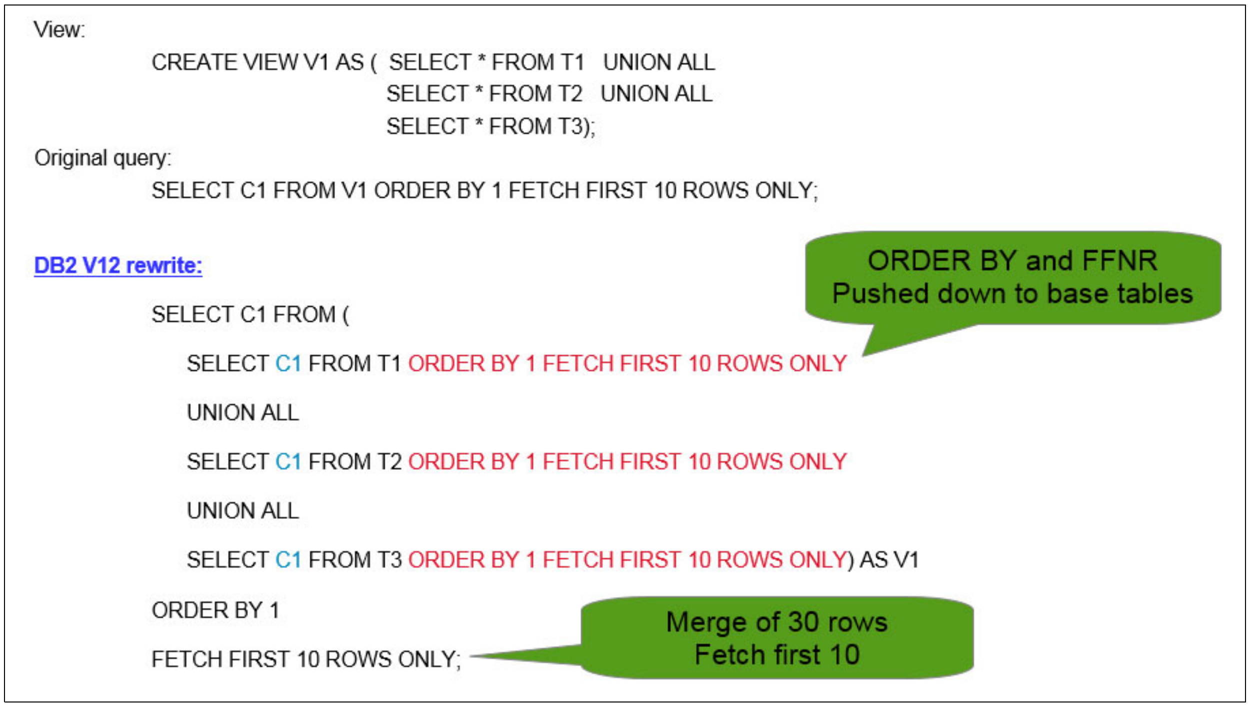
DB2 V11 rewrite:

```
SELECT C1 FROM (
     SELECT * FROM T1  UNION ALL
     SELECT * FROM T2  UNION ALL
     SELECT * FROM T3 ) AS V1
ORDER BY 1
FETCH FIRST 10 ROWS ONLY;
```

Figure 8-9 Db2 11 ORDER BY and FETCH FIRST with materialized view

In Db2 12, the ORDER BY and FETCH FIRST clauses are pushed inside each leg of the UNION ALL, as shown in Figure 8-10 on page 255. Also of note as to another difference between Figure 8-9 and Figure 8-10 on page 255, the SELECT list of each UNION ALL leg contains SELECT * in Figure 8-9 (Db2 11), which results in all columns from each table being retrieved and materialized.

As shown in Figure 8-10 (Db2 12), you see that the SELECT * from the view definition had the unnecessary columns pruned, which results in only column C1 appearing in the merged query in Db2 12.



```
View:
        CREATE VIEW V1 AS (  SELECT * FROM T1   UNION ALL
                             SELECT * FROM T2   UNION ALL
                             SELECT * FROM T3);
Original query:
        SELECT C1 FROM V1 ORDER BY 1 FETCH FIRST 10 ROWS ONLY;

DB2 V12 rewrite:

        SELECT C1 FROM (

          SELECT C1 FROM T1 ORDER BY 1 FETCH FIRST 10 ROWS ONLY

          UNION ALL

          SELECT C1 FROM T2 ORDER BY 1 FETCH FIRST 10 ROWS ONLY

          UNION ALL

          SELECT C1 FROM T3 ORDER BY 1 FETCH FIRST 10 ROWS ONLY) AS V1

        ORDER BY 1

        FETCH FIRST 10 ROWS ONLY;
```

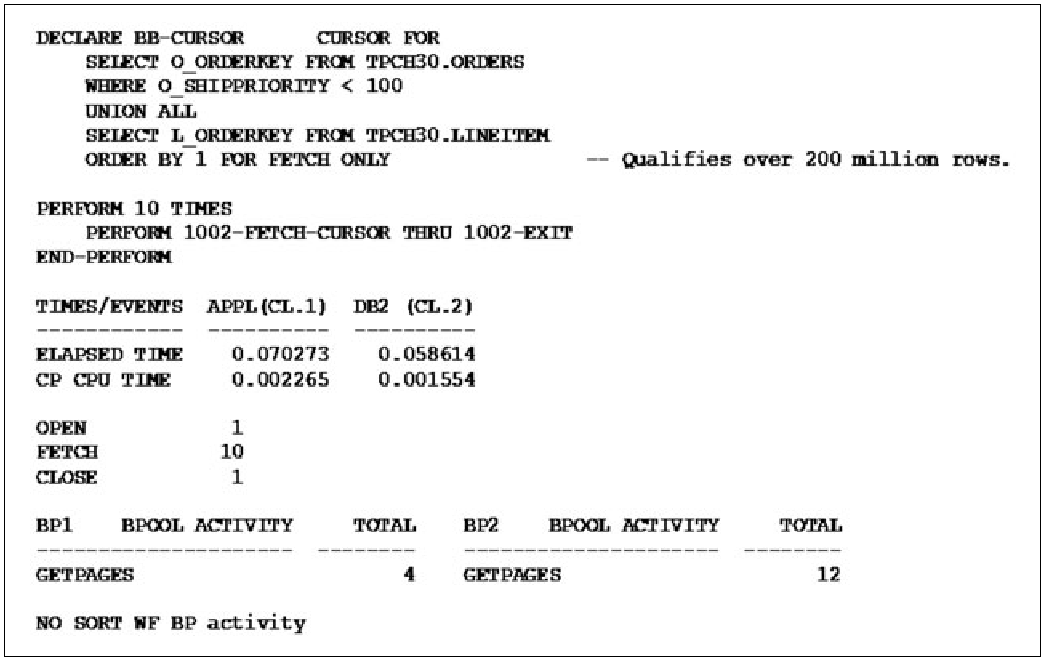
Figure 8-10 Db2 12 ORDER BY and FETCH FIRST with materialized view

With ORDER BY and FETCH FIRST pushed down to each UNION ALL leg, the Db2 optimizer can now optimize each leg independently, which includes the use of an available index to avoid the ORDER BY sort and fetch the top 10 rows without having to retrieve and sort 1 million rows from each leg. If no index exists, sort has in-memory optimizations since Db2 9 for z/OS when the number of rows is guaranteed, as is the case with FETCH FIRST.

Therefore, each leg now can optimize sort or avoid a sort (given an available index) with the ORDER BY and FETCH FIRST on each leg, and the final sort merges the three results because it is known that each leg is in order. The best-case performance comparison is Db2 11 sorting 3 million rows, and Db2 12 using indexes in each leg to retrieve only 30 rows total to retrieve the top 10 rows.

This ORDER BY (and FETCH FIRST) pushdown enhancement is also interesting to demonstrate the use of a simpler type of query that involves UNION ALL that might appear as a cursor in an OLTP or batch application.

Figure 8-11 on page 256 and Example 8-2 on page 256 shows a simple cursor with UNION ALL and a final ORDER BY, followed by a FETCH that is performed 10 times. No FETCH FIRST is available; therefore, the optimizer is not aware of the number of fetches that occur within the application.



```
DECLARE BB-CURSOR      CURSOR FOR
    SELECT O_ORDERKEY FROM TPCH30.ORDERS
    WHERE O_SHIPPRIORITY < 100
    UNION ALL
    SELECT L_ORDERKEY FROM TPCH30.LINEITEM
    ORDER BY 1 FOR FETCH ONLY              -- Qualifies over 200 million rows.

PERFORM 10 TIMES
    PERFORM 1002-FETCH-CURSOR THRU 1002-EXIT
END-PERFORM

TIMES/EVENTS  APPL(CL.1)  DB2 (CL.2)
------------  ----------  ----------
ELAPSED TIME    0.070273    0.058614
CP CPU TIME     0.002265    0.001554

OPEN            1
FETCH          10
CLOSE           1

BP1   BPOOL ACTIVITY     TOTAL    BP2   BPOOL ACTIVITY     TOTAL
---------------------  --------   ---------------------  --------
GETPAGES                      4   GETPAGES                     12

NO SORT WF BP activity
```

Figure 8-11   Db2 12 UNION ALL in an OLTP-style query

Example 8-2   Db2 12 UNION ALL in an OLTP-style query

```
V11:
|      QQP       | M | MJ |     TBNAME      |  ACC  | A_NM | IXO | JM_UJOGM |
+---------------------------------------------------------------------------
| 00032-001-001  | 3 | ?  |                 |   :   |      | N   | NN NNYNN |
| 00032-002-001  |   | ?  | ORDERS          | R :   |      | N   | NN NNNNN |
| 00032-003-001  |   | ?  | LINEITEM        | I :   | L_OK | Y   | NN NNNNN |

V12:
|      QQP       | M | MJ |     TBNAME      |  ACC  | A_NM | IXO | JM_UJOGM |
+---------------------------------------------------------------------------
| 00032-001-001  |   | ?  |                 |   :   |      |     | NN NNNNN |
| 00032-002-001  |   | ?  | ORDERS          | I :   | O_OK | N   | NN NNNNN |
| 00032-003-001  |   | ?  | LINEITEM        | I :   | L_OK | Y   | NN NNNNN |
```

The summarized trace information that is shown in Figure 8-11 shows 1.5 milliseconds of class 2 CPU time, 10 fetches, and a total of 16 getpages across BP1 (index) and BP2 (data). Although the Db2 11 results are not shown for this example, the query qualifies over 200 million rows. Because Db2 11 does not push the ORDER BY to each UNION ALL leg, all 200 million rows are sorted and written to work file for the application to fetch only 10 rows.

What also is not shown in Figure 8-11 is the access path chosen in Db2 12, although it can be deduced from the low number of getpages that both legs of the UNION ALL can use an index to avoid the ORDER BY sort. In addition to the ORDER BY pushdown (and FETCH FIRST pushdown when applicable), sort can merge the legs across the UNION ALL.

What cannot be deduced from this example is how many getpages were required across each leg. Determining this information requires a more detailed getpage trace or separating the objects into more buffer pools. All 10 rows fetched can come from the first leg, the second leg, or a combination of both.

Performance results

Performance results for ORDER BY and FETCH FIRST pushdown showed up to 99% reduction in CPU time and elapsed time was for qualifying SQL.

8.1.5 Targeted UNION ALL and OUTER JOIN summary

It was highlighted in the introduction to the UNION ALL enhancements that this construct is appearing more frequently in ERP and customer workloads, and is the foundation for recent Db2 features; that is System-Period Temporal and archive transparency. The commitment in Db2 12 to improving UNION ALL performance and how this is achieved by avoiding materialization or improving the performance of materialization by piping or merging the materialization result into the next query processing step was described.

As implied in the examples that were used in this section where the Db2 11 materialization processes all table rows, avoiding this materialization in Db2 12 can result in only the required rows being processed, which in turn results in orders of magnitude in performance improvement.

Although most of the examples demonstrated the challenge and benefit for UNION ALL queries, the Db2 12 optimizations are equally applicable to other materialized views or table expressions with outer join. Many of the same materialization performance challenges exist for optimal performance of OUTER JOIN and UNION ALL-based queries. It is expected that Db2 12 will provide significant performance improvement for these query patterns.

8.2 Predicate optimizations

The stage at which Db2 can evaluate predicates can have a significant effect on SQL performance. It is well-understood that indexable and stage 1 predicates allow filtering to occur at an earlier stage and can use indexes to restrict the search range.

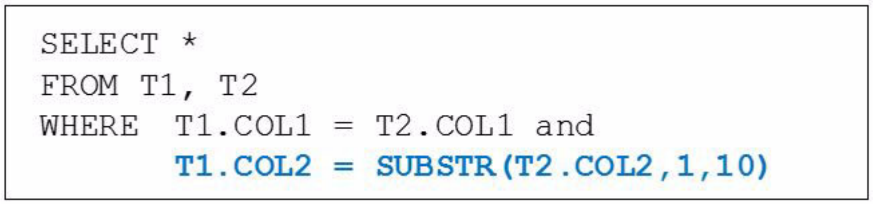
Although limiting the search range can provide orders of magnitude of performance improvements over scanning the entire object, if the only filtering comes from a stage 2 predicate, the performance of that query or application is not acceptable and deployment of that query or application usually fails.

Improving the performance of stage 2 predicates or allowing predicates to become indexable/stage 1 can enable successful application deployments. Applications or queries in which filtering from other indexable or stage 1 predicates occurs are less affected by stage 2 predicates.

8.2.1 Sort for stage 2 join expressions

The first use case of stage 2 predicates that is targeted in Db2 12 involves stage 2 join predicates. Numerous enhancements were available in recent releases that improve the performance of stage 2 predicates. However, stage 2 join predicates, particularly on the inner table of a join, remain a performance challenge.

Figure 8-12 shows a simple example of a stage 2 join predicate.



```
SELECT *
FROM T1, T2
WHERE  T1.COL1 = T2.COL1 and
       T1.COL2 = SUBSTR(T2.COL2,1,10)
```

Figure 8-12 Stage 2 join predicate

Db2 12 can improve the performance of stage 2 join predicates by allowing the sort to evaluate the function, and allowing a sparse index to be built on the result for the join. This candidate then becomes an optimal candidate on the inner table when the result can be contained in-memory or when there is no other viable index to support the filtering of the join. Alternatively, if the table with the join expression is the outer table of the join, a sort on the join order can allow access to the inner table to be performed sequentially.

The use of this enhancement does not show up explicitly in the PLAN_TABLE, but it can be inferred from the use of SORTN_JOIN='Y' and PRIMARY_ACCESSTYPE='T' (sparse index usage) when the stage 2 predicated is on the inner table, or the use of SORTC_JOIN='Y' when the stage 2 predicate is on the table that was selected as the outer table of the join operation.

Stage 2 join predicates are often observed if tables are not designed with consistent data types for joined columns (which can occur if more applications are integrated later), if business information is embedded within columns, or if time stamp columns are used within each table and the join is by the consistent date portion of those columns.

For example, the inserted time stamps do not match between two tables as the rows were inserted at a slightly different time, but you are looking only for rows that match the date portion of the time stamp column WHERE DATE(T1.TIMESTAMPCOL) = DATE(T2.TIMESTAMP). Therefore, Db2 12 can improve performance significantly for these situations without requiring a targeted index on expression to be built.

Performance results for sort on stage 2 join predicates

Although we observed these constructs in multiple customer workloads, our standard IBM Retail Data Warehouse workload is well-designed and does not use this type of stage 2 join predicates. To evaluate the performance effect of this enhancement, the IBM Retail Data Warehouse workload was modified to include several stage 2 join predicates. A 20 - 90% reduction in CPU time was observed for qualifying queries.

8.2.2 User-defined table function predicate optimizations

User-defined table functions (also known as table UDFs, table functions, or TUDFs) were initially targeted to allow an application program to be called from within an SQL statement. This ability provided the flexibility to access non Db2 objects and represent them as a table within SQL to be joined with Db2 tables. Inline table UDFs were a further extension to Db2's UDF support, which allows the definitions to contain native SQL.

There was an increase in the use of table functions as an alternative to views because of the capability to create a table function with input parameters, whereas parameterized views are not supported in Db2 for z/OS.

Although older releases provided similar merge (and thus materialization avoidance) capabilities for table functions that were syntactically equivalent to views, Db2 12 improved the merge of deterministic table functions (defined as DETERMINISTIC and NO EXTERNAL ACTION) with input parameters and the indexability of input parameters as predicates within the table function, as shown in Example 8-3.

Example 8-3 Table function with input variables

```
Simplified table function definition:
CREATE FUNCTION TPCHUDF(in_date1 CHAR(10), in_date2 CHAR(10))…
SELECT O_ORDERPRIORITY, COUNT(*)
FROM ORDERS
WHERE  O_ORDERDATE >= DATE(in_date1)    <<< Was STAGE 2. Now indexable
   AND O_ORDERDATE <  DATE(in_date2)    <<< Was STAGE 2. Now indexable

Query referencing table function:
SELECT * FROM table(TPCHUDF('1993-07-01', '1993-10-01')) as TAB1;
```

As with other cases where a merge occurs, such as views and table expressions, the table UDF's name no longer appears in the PLAN_TABLE. As shown in Example 8-3, the table function is merged and the PLAN_TABLE shows only the ORDERS table (that uses index access) and no longer the TPCHUDF table function.

Performance results for table UDF predicate optimizations

The following performance results were observed:

- Modified table UDF version of our IBM Retail Data Warehouse: 44% reduction in CPU for the workload.
- SAP CDS Fiori workload containing table UDFs: 10% reduction in CPU.
- Up to 99% reduction in CPU, elapsed time, and getpages for sweet spot queries. (Sweet spot queries used a table space scan before and now can use matching index access.)

8.2.3 VARBINARY indexability improvements

Variable-length binary (VARBINARY) is a more recent data type that was added to Db2 for z/OS. It increased the adoption of any new feature and typically clarifies customer usage patterns; therefore, it identifies opportunities for improvement.

Db2 12 adds indexability support for mismatch length comparisons of BINARY and VARBINARY data types, which were previously stage 2. Comparisons of the same length were indexable prior to Db2 12. Example 8-4 compares the pre Db2 12 predicate as stage 2 to the same predicate with a CAST expression added in Db2 12 to support indexability of mismatched length between both VARBINARY columns. (Mismatched length BINARY is not shown.)

Example 8-4 Indexability for mismatched length VARBINARY

```
SELECT A.C_CUSTKEY, HEX(A.C_NAME)
FROM CUSTOMER A, CUSTNULL B
WHERE A.C_NAME > B.C_NAME;
CUSTOMER.C_NAME: VARBINARY(25) NOT NULL
CUSTNULL.C_NAME: VARBINARY(30)
Pre DB2 12:      A.C_NAME > B.C_NAME                           (stage 2)
DB2 12:     CAST(A.C_NAME AS VARYING BINARY(30)) > B.C_NAME (Indexable)
```

Although many customers might identify that VARBINARY or BINARY data types are not used within their environments, improving indexability is important because Db2 scalar functions are available that return the result of the function as BINARY or VARBINARY. The improvements to the underlying support of BINARY and VARBINARY indexability were necessary to allow index on expressions to be built on those scalar functions and to be used for matching index access. Example 8-5 shows a scalar function as an index on expression that is now indexable in Db2 12.

Example 8-5 Index on expression example for VARBINARY-based expression

```
CREATE INDEX EMPLOYEE_NAME_SORT_KEY ON EMPLOYEE
(COLLATION_KEY(LASTNAME, 'UCA410_LDE', 600));
SELECT  *
FROM   EMPLOYEE
WHERE  COLLATION_KEY(LASTNAME, 'UCA410_LDE', 600) = < ?
```

The example that is shown in Example 8-5 uses the COLLATION_KEY scalar function with a parameter that is tailored to German. The COLLATION_KEY scalar function returns the result as VARBINARY.

You can create an index on expression by using COLLATION_KEY in Db2 11, but it is not used by predicates as the index used in the sample query that is shown in Example 8-5 because the predicate is not indexable in Db2 11. However, in Db2 12 it is indexable, so the query that is shown in Example 8-5 can use the EMPLOYEE_NAME_SORT_KEY index on expression index.

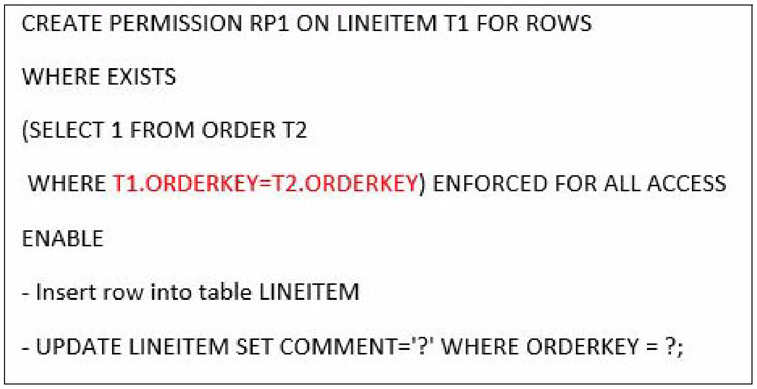
Performance results

Performance results for improved VARBINARY indexability showed up to a 99% reduction in CPU and elapsed time for sweet spot queries that switched from table space scans to matching index access.

8.2.4 Row permission indexability for correlated subqueries

Row permissions improve data security at a lower level of granularity and become an attractive solution for integrating that security into the database system rather than requiring application control or implemented by using views. Efficiency of security validation is paramount; therefore, this enhancement in Db2 12 to resolve indexability of correlated subquery predicates within a row permission.

Before Db2 12, correlated subquery predicates within a row permission are stage 2 predicates during insert and update operations. A correlated subquery example of a row permission is highlighted in Figure 8-13.



```
CREATE PERMISSION RP1 ON LINEITEM T1 FOR ROWS

WHERE EXISTS

(SELECT 1 FROM ORDER T2

 WHERE T1.ORDERKEY=T2.ORDERKEY) ENFORCED FOR ALL ACCESS

ENABLE

- Insert row into table LINEITEM

- UPDATE LINEITEM SET COMMENT='?' WHERE ORDERKEY = ?;
```

Figure 8-13   Correlated subquery predicates in a row permission

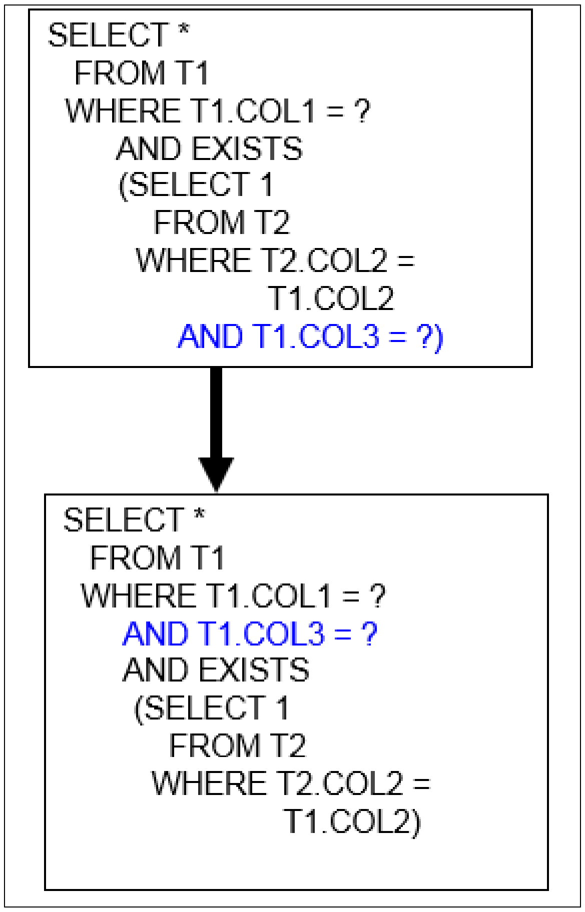
Db2 can now use an index for the EXISTS part of the row permission during insert and update.

Performance results

Performance results for improved row permission indexability included the IBM Retail Data Warehouse workload, which featured a modified TPCH queries that used row permissions. Up to a 99% reduction in CPU and elapsed time was observed.

8.2.5 Predicate bubble up enhancements

Db2 V12 opens more opportunities for predicate indexability by bubbling up predicates, which was restricted in past releases. For the example that is shown in Figure 8-14, predicate `T1.COL3 = ?`, which was "misplaced" in the subquery block, is bubbled up to the parent query block by Db2 so that any indexes that are qualified for the predicates can be used. The bubbled up predicate is visible in the rewritten query (by Db2) after query transformation. This information is stored in XML format in the DSN_QUERY_TABLE explain table.



```
SELECT *
  FROM T1
 WHERE T1.COL1 = ?
      AND EXISTS
      (SELECT 1
         FROM T2
        WHERE T2.COL2 =
                 T1.COL2
          AND T1.COL3 = ?)
```

```
SELECT *
  FROM T1
 WHERE T1.COL1 = ?
     AND T1.COL3 = ?
     AND EXISTS
      (SELECT 1
         FROM T2
        WHERE T2.COL2 =
                 T1.COL2)
```

Figure 8-14 An example of a predicate bubbling up

Performance results for predicate bubble up enhancements

Sweet spot queries can have up to 99% CPU time and elapsed time improvement because of access path changes from table space scan to index access for table T1. Sweet spot queries are queries in which the predicate that is bubbled up provides a good index matching predicate where none were available before.

8.2.6 IN-list predicate enhancements

Db2 12 provides improved performance for long IN-lists and IN-list direct table access.

Improved performance for long IN-lists

Db2 V10 introduced streamlined processing for non-matching IN-list predicates that were applied against an index or against data. These changes resulted in a dramatic improvement in the performance of non-matching IN-lists.

This enhancement included a limitation in that it applied to only IN-lists of 128 elements or fewer in Db2 V10 and Db2 11 when the IN-list was applied as an index screening predicate. Db2 12 removes this limitation so that large IN-lists with more than 128 elements can also benefit from the V10 performance enhancements and scale linearly beyond 128 elements, regardless of whether they are applied against an index or data page.

The limit is now typically greater than 3000 IN-list elements and is the same for index and data access. This enhancement was retrofit to Db2 V10 and Db2 11 by using APAR PI41598. A query that can benefit from this enhancement is shown in Example 8-6.

Example 8-6 Long IN-list predicate performance enhancement

```
Example of the new feature in use:
SELECT * FROM TAB1
WHERE COL1 = ?
AND COL3 IN (?,?,?,…?);  - non matching IN list with 100, 200, etc. elements
INDEX on (COL1, COL2, COL3)
```

Performance results for large IN-lists

Sweet spot queries showed up to a 99% reduction in CPU time and equivalent reduction in elapsed time (as a result of the CPU time reduction) for index screening IN-list predicates with more than 128 elements. Although the CPU time reduction is expected to be greater when the IN-list is larger, any IN-list of more than 128 elements is expected to see a significant CPU time reduction for the non matching IN list processing part of the query.

Improved performance for IN-list direct table access

IN-list direct table access occurs when Db2 uses in-memory tables to process one or more IN-list predicates as matching predicates. This access type is indicated in the PLAN_TABLE by ACCESSTYPE='IN'. Db2 supports matching on multiple IN-list predicates if indexes exist on the necessary columns.

Before Db2 12, IN-list direct table access was limited to the use of a nested loop join with list prefetch. This limitation can result in poor performance when the number of qualifying RIDs per list prefetch request is small. For example, the performance suffers if few rows qualify per inner table join probe or even if more rows qualify per probe but these rows span many partitions that results in few RIDs per list prefetch request because a single prefetch request cannot span partitions.

Db2 12 removes this limitation and chooses between nested loop join and hybrid join based on estimated cost. Hybrid join can be more efficient because of the way the qualifying RIDs are first accumulated before making the list prefetch requests. When Db2 12 chooses to use hybrid join with IN-list direct table access, the qualifying RIDs are not sorted (SORTN_JOIN=N) to avoid possible CPU regressions. Nested loop join with list prefetch is still selected by optimizer for IN-list direct table access if the inner table access is estimated to contain enough qualifying RIDs per probe that the use of nested loop join is more cost effective than the use of hybrid join.

Performance results for ACCESSTYPE='IN' with hybrid join

The following performance results were observed:

- Internal insurance type workload: Up to a 7% reduction in CPU for qualifying queries.
- Sweet spot queries: Up to a 30% reduction in CPU and elapsed time. Sweet spot queries are cases where probing the inner table takes up a large portion of the query's runtime, and the number of table RIDs qualifying per probe of the inner table (when nested loop join is used) is small.

8.3 Optimizer cost model enhancements

As we move through the internal Db2 components where the performance enhancements in Db2 12 are the focus, next we consider the optimizer cost model. Within Db2, this component is the access path selection (APS) component. The optimizer cost model (access path selection) is responsible for choosing the lowest-cost access path based on the query after query transformation performed view or table expression merge and predicate transformations (for example, pushdown and rewrites).

Referring to “Query regression workloads ” on page 284, in which the internal Db2 performance workload results are described, most queries and workloads that achieved the largest performance gains were those queries and workloads that chose a new access path in Db2 12 compared with Db2 11. This result might occur because of the following reasons:

- A new access path became available because of a query transformation (such as a predicate rewrite, predicate pushdown, or view or table expression merge).
- A new execution path became available.
- An existing access path was optimized to be more efficient (such as Db2 12 Runtime Adaptive Index).
- The optimizer cost model has access to more accurate inputs or improves upon its cost formulas.

It is incorrect to infer from the groupings of topics in this overview that each of the Db2 components operates independently. In fact, each component plays a role in providing an efficient execution for a specific SQL statement. With the descriptions of query transformations and runtime optimizations in this chapter (“UNION ALL and OUTER JOIN performance enhancements” on page 244 and “Runtime enhancements” on page 268), we can assume that the optimizer cost model played its role in ensuring an efficient access path that is based on those enhancements. Therefore, we can now turn our attention to the improvements that are specific to the cost model that use access path choices.

8.3.1 Extending NPGTHRSH to default statistics

A discussion of the NPGTHRSH DSNZPARM is unlikely to garner the same immediate understanding compared with a discussion of the VOLATILE table attribute. Both attributes are trying to effectively solve the same problem in that there are instances where RUNSTATS collected on an object might be unreliable because that object is early in an application rollout (meaning it is small but likely to grow quickly), or the object size grows and shrinks regularly, which makes it difficult to collect representative statistics at the right time.

The NPGTHRSH DSNZPARM was delivered in Db2 V7 to preference matching index access over other access paths at the subsystem level. Db2 12 extends the effectiveness of this zparm.

When NPGTHRSH is set, this zparm value is compared to the number of pages for a table; NPAGESF catalog statistic, or at the partition level, it is the number of pages in the partition that is compared to NPGTHRSH. If NPAGESF is less than NPGTHRSH, the optimizer preferences matching index access for access to that table, if possible.

NPGTHRSH is disabled by default, although there is at least one major ERP vendor that recommends Db2 for z/OS customers set their default to 10, which was effective for many years. A complaint was not ever received by a customer that wanted a table space scan and Db2 chose matching index access for a table with nine (or less) pages.

In Db2 11 and earlier, if no statistics were collected on a table and NPAGESF=-1, Db2 did not use -1 for the NPGTHRSH comparison. Instead -1 becomes 501 for all optimizer costing, including the comparison to NPGTHRSH.

Db2 12 instead allows default stats (-1) to apply the NPGTHRSH rules, if enabled. Enabling NPGTHRSH in Db2 12 also applies the rule for preferencing matching index access if the index being considered includes default statistics. This scenario is possible if an index was recently created but statistics are not yet collected.

Note: Compared to the NPGTHRSH DSNZPARM, the VOLATILE table attribute favors only index usage for single table queries or for the leading table of join queries. Therefore, VOLATILE might not address the index choice for multi-table joins, or might not adequately influence the table join sequence.

Performance results

Performance results for NPGTHRSH with default statistics showed that, table space scans were reduced and index access was selected for qualifying objects as expected.

8.3.2 List prefetch and hybrid join cost improvements

List prefetch and hybrid join are not new to Db2. However, what is new are improvements to the optimizer to encourage list prefetch and also hybrid join for poorly clustered objects when sort avoidance is not a viable candidate as one of the lower-cost access paths.

The largest complaint with list prefetch is that, if chosen for an online application's SQL statement, list prefetch degrades performance significantly because it accumulates all qualified RIDs before fetching the rows to return to the application. To be more specific, when the query contains an ORDER BY and an index exists to avoid the sort, list prefetch can be a "dangerous access path" to choose for an online screen-type application.

The other complaint is if the optimizer underestimates the filtering of a predicate and most of the table qualifies, list prefetch is less efficient compared with a table space scan. The introduction of the *runtime adaptive index* enhancements for multi-index access and list prefetch, allows the optimizer to improve the costs that are associated with list prefetch, multi-index access, and hybrid join. The optimizer now more appropriately reflects the improved I/O performance that is available for such choices when a sort avoidance plan is not one of the viable lower-cost plans.

In addition to the cost model improvements that is associated with list prefetch, hybrid join support (which uses list prefetch) was extended as a viable optimizer choice in more situations when parallelism is enabled. Hybrid join has the benefit of more efficiently accumulating list prefetch requests to access the inner table of a join as compared with nested loop join with list prefetch.

Performance results for list prefetch and hybrid join cost improvements

Several internal workloads and a customer-based workload were used to evaluate this enhancement. The following results were observed:

- A 5% reduction in CPU time for our IBM Retail Data Warehouse workload.
- A 12% reduction in CPU time for our Crystal Reports workload.
- A 50% reduction in CPU time for a poorly clustered version of a customer workload.

8.3.3 Improved filter factor for CURRENT DATE/TIMESTAMP predicates

Range predicates with parameter markers or host variables are among the most difficult predicates for a query optimizer to estimate accurately. At execution time, a range can be specified that qualifies anywhere in the range 0 - 100% for a simple range predicate.

Although the available catalog statistics for a column identify the ranges that exist within the table, how much of that range qualifies cannot be known until the literal value is supplied at runtime. A predicate ORDER_DATE < ? is more difficult to accurately estimate compared with predicates with literal values, such as ORDER_DATE < '2016-01-01' or ORDER_DATE < '9999-12-31'.

Db2 12 supports the resolution of predicates that involve CURRENT DATE and CURRENT TIMESTAMP to use the actual current value at the bind and prepare time. This support includes date/time stamp arithmetic incorporating those special registers.

In Db2 11 and older, ORDER_DATE < CURRENT DATE use the same filter factor estimate as that of a parameter marker or host variable (ORDER_DATE < ?). Example 8-7 shows predicate examples that resolve the predicate values at bind and prepare time to provide a more accurate filter factor estimation in Db2 12.

Example 8-7 Predicates involving date special registers

```
SELECT * FROM T1
WHERE ORDER_DATE > CURRENT_DATE - 7 DAYS
  AND ORDER_DATE < CURRENT_DATE;
```

The obvious question from this enhancement is whether the filter factor estimate becomes stale because the values are resolved at bind and prepare, and it is possible a static bind was performed six months ago. Although it is true that the current date six months ago is not the same as the current date today, it is likely that the data in the table also continued to change over time such that a six-month move in current date also corresponds to six months of more rows that were added or updated to the table.

8.3.4 Improved resolution of filter factors at bind and prepare using index probing

Index probing was added in Db2 10 for dynamic prepares with literals, including REOPT(ONCE) and static binds that use REOPT(ALWAYS). For index probing, Db2 uses the predicate value to probe the index non-leaf pages if a table is volatile, qualifies based on the NPGTHRSH z-parm value, or if the original predicate estimate is assumed to qualify zero rows. Index probing allows the optimizer to validate the estimated filter factor based on the data in the index. This index probing process also accesses the RTS information for clarity on the current object sizes.

In Db2 12, index probing is moved to an earlier stage of the access path selection process. This way, index probing is performed only once for any index where probing applies and adjusts the filter factors for all applicable predicates. For example, this process allows the multi-column index probing results to be used for local and join predicates. This enhancement improves the performance of index probing and allows the resultant filter factor to be used more consistently for the benefit of improved cost estimation across all affected indexes.

8.3.5 Improved bind and prepare performance with many indexes

Recent deployments of ERP applications included many indexes per table. This type of schema is typically based on a design where each function of the application maps to a set of columns of the table; therefore, it depends on the features that are used as to which indexes are beneficial to the user.

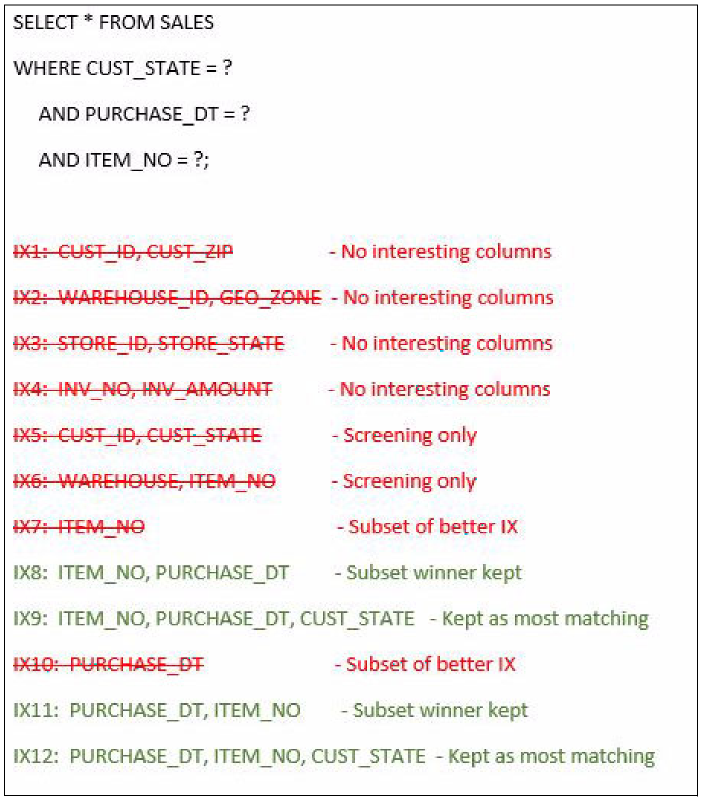
A query optimizer reads in all available catalog information, including object definitions and statistics for a specific SQL statement, as input to the bind and prepare process to allow cost comparison of available choices to determine the lowest-cost access path.

In Db2 11 and older, the optimizer evaluates every available index for each object. This approach presented performance challenges with multi-table joins involving hundreds of indexes per table. Such performance issues were not a concern because most of the tables feature 10 indexes or less.

In Db2 12, the optimizer first evaluates all available indexes for each table and ranks indexes based on the existence of matching predicates, screening predicates, and clustering attributes. Indexes are pruned from consideration for the access path if there is no filtering value.

Indexes are also pruned when there are better filtering indexes available. This feature can considerably improve bind and prepare performance when many indexes are on a table. It also can improve the chances that the index with the most filtering is chosen by the optimizer.

An example of an SQL statement with 12 indexes on the table is shown in Figure 8-15 on page 268. In Db2 11, all 12 indexes are evaluated by the optimizer throughout the access path selection process. In Db2 12, eight of those indexes are discarded before beginning the access path selection process.



```
SELECT * FROM SALES
WHERE CUST_STATE = ?
   AND PURCHASE_DT = ?
   AND ITEM_NO = ?;
```

~~IX1: CUST_ID, CUST_ZIP~~	- No interesting columns
~~IX2: WAREHOUSE_ID, GEO_ZONE~~	- No interesting columns
~~IX3: STORE_ID, STORE_STATE~~	- No interesting columns
~~IX4: INV_NO, INV_AMOUNT~~	- No interesting columns
~~IX5: CUST_ID, CUST_STATE~~	- Screening only
~~IX6: WAREHOUSE, ITEM_NO~~	- Screening only
~~IX7: ITEM_NO~~	- Subset of better IX
IX8: ITEM_NO, PURCHASE_DT	- Subset winner kept
IX9: ITEM_NO, PURCHASE_DT, CUST_STATE	- Kept as most matching
~~IX10: PURCHASE_DT~~	- Subset of better IX
IX11: PURCHASE_DT, ITEM_NO	- Subset winner kept
IX12: PURCHASE_DT, ITEM_NO, CUST_STATE	- Kept as most matching

Figure 8-15 Db2 12: Discarding unnecessary indexes from consideration by the optimizer

Performance results for bind and prepare with many indexes

The following performance results were observed:

- A 70 - 80% reduction in bind and prepare CPU time was observed for targeted customer queries that contained 50 - 200 indexes per table. In a typical scenario, the number of indexes that must be evaluated by the optimizer was reduced from over 200 to less than 20 as a result of this enhancement.
- No access path or CPU regressions for other queries measured.

8.4 Runtime enhancements

The Db2 12 target area that is described in this section are those enhancements that take advantage of intelligent runtime adaptations or a more efficient runtime code path. The predominant Db2 component that delivers the enhancements that is described in this section is the *Db2 runtime component* within RDS (the grouping under this heading).

As with most of the query performance and optimization enhancements in Db2 12, these enhancements are complementary to other features, such that the same query can benefit, for example, from a UNION ALL enhancement, a runtime optimization, and several other enhancements.

8.4.1 Runtime adaptive index processing

The first runtime optimization enhancement is called *runtime adaptive index processing*, which extends upon RID-based access paths (list prefetch and multi-index access) and adds runtime enhancements to adapt at execution time based on index filtering.

The simple SQL statement that is shown in Example 8-8 can be used to highlight the challenge that generic (search) queries place on a query optimizer. Numerous coding is used for this type of SQL. It is commonly used for search screens, where the user can complete various fields and choose any or all combinations.

Example 8-8 Generic search SQL challenge

```
SELECT *
  FROM CUSTOMER
  WHERE LASTNAME  LIKE ?
    AND FIRSTNAME LIKE ?
    AND ADDRESS   LIKE ?
    AND CITY      LIKE ?
    AND ZIPCODE   BETWEEN ? AND ?
```

Regardless of the technique that is used for the SQL, the premise is the same in that the programmer wants to code a single SQL statement that can be generally used. Database administrators fight this battle to have programmers construct dynamic SQL at execution time that is specific to the input request, or to limit the combinations that are possible and code a subset of targeted SQL. Typically, this battle is lost, and it becomes the responsibility of the database to address the performance challenges.

Despite the option to bind the package with the REOPT(ALWAYS) bind option, this binding might not be possible if the requests are coming into Db2 through DRDA because too many SQLs use the same package, and similarly for static SQL, all SQL in the package are affected.

The SQL that is shown in Example 8-8 is used as a generic search query by enabling or disabling each predicate that is based on the user input.

For example, if LASTNAME and CITY are entered, these predicates are populated with their search values, as shown in the following example:

```
LASTNAME LIKE 'SMITH' AND CITY LIKE 'SAN JOSE'
```

All other predicates are populated with the following values:

```
LIKE '%%%%'
```

If the query was coded with BETWEENs for character columns, the following predicates are used for numeric columns:

```
BETWEEN 'AAAAAAAA' AND 'ZZZZZZZZ')
```

or

```
BETWEEN 0 AND 99999
```

The Db2 optimizer (or any optimizer) cannot choose one access plan that best matches the filtering predicates because those predicates can change every execution.

Runtime adaptive index processing is a Db2 12 optimizer (runtime) enhancement that targets this type of SQL performance challenge. To describe this feature further, assume that one separate index is available for each column in the WHERE clause. For this example, the optimizer chose a multi-index access plan that uses each of those five indexes for each of the predicates in the query that is shown in Example 8-8 on page 269.

At execution time, the Db2 runtime component first looks at the values that are provided for each predicate to determine whether the predicate is likely or unlikely to be filtering. LIKE predicates that contain wildcards, such as %, are considered unlikely to be filtering. BETWEEN or other range predicates that appear to cover the entire range of values are also considered unlikely filtering candidates.

At this stage, runtime moves unlikely filtering indexes to the end of the multi-index execution sequence and begins processing the first likely filtering index based on the original index operation sequence, which is identified by the value of the Multi-Index OPeration SEQuence (MIXOPSEQ) column in the PLAN_TABLE, after unlikely filtering indexes are moved to the end.

After a certain threshold is reached when processing the first filtering index, processing of that index is paused, and the other remaining indexes are also searched (up to a similar threshold), which allows Db2 to determine the approximate filtering of each index. The threshold that is associated with the first index is implemented to ensure that short-running queries are not affected by any cost that is associated with validating the filtering of other indexes.

After the filtering of each remaining index is determined, runtime can reorder the index execution sequence, discard any non-filtering indexes, or revert to table space scan if no indexes provide sufficient filtering. If an index plan is to continue, the processing of each index continues where it was left off, and in the order of best filtering.

Because this enhancement is built upon the existing multi-index access path, each leg is index-only as it is accumulating qualified RIDs that are then intersected with the RID list from each leg, and finally the data rows are accessed.

Runtime adaptive index processing can adjust am adapt at execution time to changes in actual filtering without requiring REOPT(ALWAYS) because REOPT results in a reoptimization of the access path for each execution.

At execution time, a first quick peek at the literal values and adjustment of the index execution order or reversion to table space scan based on determination that certain index legs are not filtering occurs. Also, a more accurate filtering estimate is determined at execution after a threshold is reached. This adaptability is applicable to all list prefetch and multi-index access plans.

For multi-index ANDing, adaptive index processing can reorder the indexes from most to least filtering, early-out of non-filtering indexes, and revert to table space scan earlier than past releases with insufficient filtering.

For multi-index ORing or list prefetch-based plans, this early determination of filtering at two stages (first from peeking at the literals, and second after the internal threshold is reached) allows an earlier determination of the decision to fall back to table space scan compared to past releases if there is no adequate filtering from the chosen indexes.

The use of runtime adaptive index processing is not limited to the generic search SQL challenge. The optimizer cannot identify the intent of an SQL statement. The adaptive capabilities that are described here are applicable whenever the optimizer chooses a RID-based access path, such as list prefetch or multi-index access.

As this enhancement can bring significant runtime improvements, the obvious question is whether this means that Db2 12 sees a significant increase in multi-index access plans. This result is not expected because the optimizer increases only its consideration of multi-index access when the candidate predicates have high uncertainty in their filter factor estimates.

Predicate uncertainty was added in Db2 10 to supplement the optimizer's filter factor estimate, and by design, the predicates that have the highest uncertainty in their filtering are range predicates, which can filter anywhere 0 - 100% of the range. Other predicates that can be difficult to estimate include JSON, spatial, and index on expression predicates.

Example 8-9 shows a simple example of three range predicates, each with an available index, and filtering with a high degree of uncertainty until execution time when the literal values are known. This type of query becomes a good candidate for multi-index access and its adaptive capabilities in Db2 12.

Example 8-9 SQL with high predicate uncertainty

```
SELECT *
  FROM TAB1
  WHERE COL1 < ?
    AND COL2 < ?
    AND COL3 < ?;
INDEXES: IX1(col1), IX2(col2), X3(col3)
```

Note: IFCID 125 (RID list processing usage) was enhanced with several new fields for tracking these new features.

Adaptive index processing is available in Db2 12 at function level V12R1M100 after rebinding the application package. The use of REBIND with APREUSE(ERROR) enables adaptive index processing, provided Db2 can successfully reuse the access path.

Performance results using runtime adaptive index processing

The following performance improvements were observed:

- A 20% CPU time saving for list prefetch or multi-index ORing when failover to table space scan is the most appropriate choice.
- Orders of magnitude performance improvement for multi-index ANDing when a mix of filtering and non-filtering indexes is used.
- For an internal workload that uses many multiple index accesses, a 95% reduction in CPU time was observed, and elapsed time was reduced by 50% when compared to Db2 11 for same multi-index workload. The workload uses a 10 million row table with six single column indexes. It consists of 200 SQL statements with range predicates on those indexed columns that use varying value ranges to provide the Db2 optimizer and runtime with a large set of different selectivity challenges.

8.4.2 Internal runtime and complex expression enhancements

In addition to the use of adaptive index processing, Db2 12 features several other enhancements that help to improve Db2 runtime performance in certain areas.

More use of machine code generation (gencode)

In each recent Db2 release, efforts were made to improve the execution performance of expensive operations by using machine code generation.

SPROCs (and other xPROCs) are the most notable examples of machine code generation in Db2. These SPROCs and xPROCs are relied upon by customers for noticeable CPU time reductions after static SQL undergoes a REBIND (regardless of the APREUSE option) in each new release.

Db2 is also using gencode in the following examples:

- (non-matching) IN-list and OR predicate evaluation in Db2 10
- DECFLOAT data type processing
- SUBSTR scalar function evaluation in Db2 11

Db2 12 applies the technique of machine code generation to the following constructs:

- CASE expressions without any other embedded scalar functions or other expressions, for example:

  ```
  CASE WHEN PC.CODE = '1' THEN 'Open'
       WHEN PC.CODE = '2' THEN 'Closed'
       ELSE 'Unknown'
  ```

- More SUBSTR enhancements (in addition to the Db2 11 changes), which allows for a variable length string of any size (not limited to 256), and enables this machine code generation for SUBSTR that uses binary, varbinary, and vargraphic string expressions.
- External data type conversions for DATE-based data types (DATE, TIME, and TIMESTAMP).

Performance results of using more machine code generation

The following performance results were observed:

- Sweet spot query showed up to 20% reduction in CPU for SUBSTR(). Sweet spot queries are cases where the SUBSTR processing is the predominant cost of running the query, as shown in the following example:

  ```
  SELECT MAX(SUBSTR(COMMENTS,15,10))
     FROM INVOICES
     WHERE SUBSTR(COMMENTS, 1,5) = 'EXTRA'
  ```

- Up to 30% reduction in CPU for DATE-based data types was observed for sweet spot queries. For example, a query retrieving many rows from a table and the columns in the SELECT list are mostly date-time data types.

8.4.3 Db2 12 expression sharing

The next runtime optimization is referred to as *expression sharing* and is best described by using an example.

When an expression is coded within a view or table expression and a referencing select is merged with that view/table expression, that outer reference is replaced with the expression.

Example 8-10 shows a SELECT from a table expression (TX) that references an arithmetic expression four times. When the referencing SELECT is merged with the table expression, the result is that the expression is repeated four times (once for each outer reference).

Example 8-10 Db2 12 expression merge

```
SELECT SUM(C1), AVG(C1), MAX(C1), MIN(C1)
  FROM (SELECT DECIMAL(C1,9,2) * 0.75/DECIMAL(C2,9,2) AS C1
          FROM T) AS TX
Is merged to:
SELECT SUM(DECIMAL(C1,9,2) * 0.75/DECIMAL(C2,9,2))
      ,AVG(DECIMAL(C1,9,2) * 0.75/DECIMAL(C2,9,2))
      ,MAX(DECIMAL(C1,9,2) * 0.75/DECIMAL(C2,9,2))
      ,MIN(DECIMAL(C1,9,2) * 0.75/DECIMAL(C2,9,2))
  FROM T
```

From a performance perspective, the expression is run four times in Db2 11.

In Db2 12, when the merge occurs, Db2 also recognizes that the expression is repeated. At execution time, the expression is run once only and the result is shared across the four references.

This enhancement applies only to expressions where Db2 merges the query. If the query is written with multiple occurrences of the same expression, Db2 does not detect and share the expression.

Performance results when expression sharing is used

The following performance results were observed:

- Sweet spot query: Up to 10% reduction in CPU for simple expressions. The more often the expression is repeated as part of the merge operation, the more benefit can be expected.
- Sweet spot query: Up to 35% reduction in CPU for reducing UDF calls. Although, the amount of savings depends on how often the shared UDF is repeated, larger benefits can be expected compared to sharing simple expressions as UDF calls tend to be more expensive.

8.4.4 UDF result caching

The term *expression* is often used to reference any type of arithmetic, scalar function, or user-defined functions (UDFs). Of these expressions, UDFs are often the most expensive because of the flexibility in what can be contained within, including native SQL that references other tables or external UDFs that contain an external application program.

Db2 12 enhances the performance of deterministic UDFs by supporting caching of the UDF result given the same inputs. This process is achieved by using an internal hash table that is entirely managed by Db2; that is, there is no user control.

For a UDF that is defined as DETERMINISTIC and NO EXTERNAL ACTION, Db2 builds a hash key from the input variables and stores the output result. Each UDF invocation first performs a look-up in the cache to see whether those values were processed. Db2 monitors the cache at regular intervals, and if no situations exist where a cached value was found, the cache is disabled to ensure that no performance regression occurred. The cache persists only within a single query run and is not shared by other queries that are running the same UDF, or for the next run of the same SQL.

Performance results when using UDF result caching

To evaluate the benefits for this enhancement, several internal measurements were conducted. The following results were observed:

- A 40% reduction in CPU time for complex SQL with table UDF calls by using the SAP CDS Fiori workload.
- Up to 95% reduction in CPU time for repeating expensive UDF calls for sweet spot queries. Those are cases where the SQL is simple, the UDF invocation is expensive and the UDF is started many times with recurring input values, so the query can greatly benefit from the results being stored in, and retrieved from, the UDF cache.

8.4.5 Internal block fetching

Db2 12 provides an internal optimization to perform a “block fetch” operation when moving qualified rows from the Data Manager (stage 1) component to RDS (stage 2). When the optimizer enabled this internal block fetch feature, it is recorded in the newly added BLOCK_FETCH column of the DSN_DETCOST_TABLE table. This feature takes effect only when you are in V12R1M500 and packages are rebound.

Performance results

To evaluate this enhancement, several tests were conducted that used queries that return large variations in the number of rows (100 - 500,000). In general, when more rows were returned, the amount of improvement increased. The various types of queries were tested showed the following CPU time improvements when this feature is used:

- Index and data access: Up to 22% improvement
- Index only access: Up to 14% improvement
- List prefetch: Up to 20% improvement
- Table space scan: Up to 18% improvement
- Index access on a CHAR (255) column: Up to 4% improvement

8.5 Sort space reductions and in-memory use

There is a continual focus on improving performance and reducing resource consumption for sort and work file usage. The work file buffer pool and data sets can be a point of contention for workload scalability because OLTP, batch, and other queries might all converge on the same resources. For this reason, Db2 continually seeks to improve sort avoidance, optimize sort processing, and have more (smaller) sorts processed in-memory.

Materializations also converge on the work file buffer pool and data sets; therefore, the reduction in materializations for UNION ALL and OUTER JOIN queries can help to significantly reduce this contention for candidate workloads.

The (ORDER BY and) FETCH FIRST pushdown also is one of the optimizations that are related to UNION ALL, and pushing these down into the UNION ALL leg can result in Db2 taking advantage of other new or existing optimizations. The use of the FETCH FIRST clause (when combined with DISTINCT, GROUP BY, or ORDER BY) included numerous sort performance improvements since Db2 9, which are made possible because Db2 is providing precise knowledge about how many rows must be sorted and returned. Sort was implemented an in-memory replacement technique for sort when the FETCH FIRST clause is used if the result is guaranteed to be in-memory.

When a sort cannot be avoided, the use of memory to process the sort or reducing the length of the sort record can result in that the sort can be contained in-memory, or at a minimum reducing the amount of work file resources that are needed to complete the sort.

8.5.1 Increasing the number of sort tree nodes

In Db2 11, the sort tree size is limited to 32,000 nodes. Although the amount of memory for sort can be controlled by using the SRTPOOL DSNZPARM, the limit of 32,000 nodes remained.

However, in Db2 12, sort allows up to 512,000 nodes for non-parallelism sorts, or 128,000 nodes for a sort within a parallel child task, which is still capped by the value of the SRTPOOL DSNZPARM. With the ability to contain sorts within the sort tree rather than writing the result to a work file, an obvious benefit exists to increasing the value of the SRTPOOL DSNZPARM. This allocation is a per-query allocation, and it depends on the size of each sort and the number of concurrent queries that dictates how much memory is required across an entire system for sorting.

Increasing the number of nodes in the sort tree in Db2 12 not only has the benefit of containing a sort in-memory, but GROUP BY and DISTINCT sorts can take further advantage of many nodes. Db2 9 added hashing support as input to sort for duplicate removal (GROUP BY/DISTINCT), such that duplicates can be collapsed before going through sort.

In Db2, the number of hash entries is tied to the number of nodes of the sort tree. Therefore, increasing the number of nodes can result in higher cardinality GROUP BY/DISTINCT results in consolidating the groups as rows that are input to sort. Consolidating the groups as rows can increase the chance that the sort can be contained in memory, or at least reduce the amount of work file space required to consolidate duplicates during the final sort merge pass.

Increasing the value of the SRTPOOL DSNZPARM in Db2 12 can have a greater benefit to improving sort performance than in past releases. Also, increasing the SRTPOOL value in Db2 11 might not result in that memory being used if the 32,000 limit is reached before the full memory (SRTPOOL) request is allocated. Therefore, Db2 12 might use the memory up to the SRTPOOL value that might not be used previously. That is, even with the same SRTPOOL allocation in Db2 11 and 12, Db2 12 can use more memory because it is not constrained by the 32,000 nodes limit of the sort tree.

Performance results for increasing the number of sort nodes

As with all performance enhancements, several measurements were conducted to evaluate the potential benefits that can be obtained from increasing the number of sort nodes. The following observations were made:

- SAP CDS Fiori: A 5% CPU time reduction was observed for several queries. A 1% CPU time reduction across the entire workload was measured.
- SAP CDS FIN: A 1.8% reduction in CPU time for the entire workload and a 12% reduction in the total number of work file getpages were measured.
- IBM Retail Data Warehouse workload: Two queries showed a 14% and a 6% CPU time reduction. The CPU time reduction largely results from a 40% reduction in the number of work file getpages for these two queries.
- Up to 75% reduction in CPU time for sweet spot queries that became in-memory sorts in Db2 12 where they previously required work files and sort merge processing.

- Up to 50% reduction in CPU and elapsed time for queries that benefitted from increasing the SRTPOOL size ZPARM in Db2 12. The largest improvements were for queries that can now complete their group by or distinct processing in one sort pass because of the larger sort tree size.

Note: The domain of queries that benefit from this Db2 12 enhancement can be increased by increasing the size of the sort pool (SRTPOOL ZPARM).

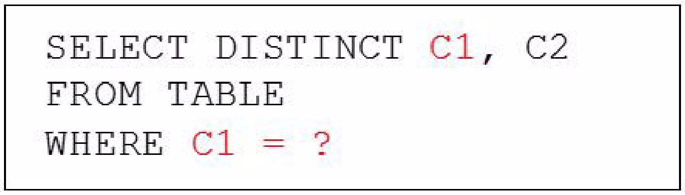
8.5.2 Reducing the length of the sort rows

The next sort enhancement is related to reducing the length of sort rows for internal sorts. Reducing the sort row length includes the benefit that it also reduces the total amount of memory and work file space needed for sort.

It is common that predicates that are coded in the WHERE clause are also (often redundantly) included in the SELECT list. Any redundancy in the sort key columns or data row columns (SELECT list) has a negative effect on sort performance and resource consumption.

ORDER BY sort already removes columns from the sort key if they are covered by equal predicates in the WHERE clause. DISTINCT or GROUP BY already remove redundant columns to allow for sort avoidance.

However, if a sort cannot be avoided for DISTINCT or GROUP BY, such redundant columns remain as part of the row to be sorted in Db2 versions before Db2 12. Starting with Db2 12, these redundant columns are removed from the sort key, as shown in Figure 8-16.



```
SELECT DISTINCT C1, C2
FROM TABLE
WHERE C1 = ?
```

Figure 8-16 Removal of redundant columns from the sort key

Because C1 has an equal predicate in the WHERE clause, all sorted rows are ensured to contain that same value; therefore, only column C2 is needed for the (DISTINCT) sort.

Similarly, sort avoids duplicating the sort key from the data portion in Db2 12 if they match each other.

Sort operates on a fixed-length concatenated key of the columns that are required to support the ORDER BY, GROUP BY, DISTINCT sort, or sort for a join operation. The requested order of the sort key might not match the order of the columns in the SQL statement's SELECT list.

When the sequence of columns in the sort match the leading columns in the SELECT list and contains only fixed-length columns in Db2, sort does not replicate the key. Instead, it uses the data portion for the sort, which can result in a dramatic reduction in the length of the sort row.

8.5.3 Reducing contention on 4 K work file pages

Another goal in Db2 12 is to continue to focus on minimizing the effect that larger sorts have on OLTP applications. The use of more memory for sorting greatly assists in reducing contention on sort buffer pools and sort work files as a shared resource. Although Db2 9 added the option for sort to use 32 K work file pages for longer sort rows (greater than 100 bytes), large sorts of smaller-length rows are possible to dominate the 4 K work file buffer pool and work file data sets.

In Db2 12, many larger sorts of small rows (less than 100 bytes) now use 32 K work file pages instead of 4 K. “Larger sorts” refers to sorts where the number of records input to sort exceeds the number of sort nodes. For these large sorts of small rows, Db2 compacts the rows and stores them as one physical row per 32 K work file page. This configuration allows the Db2 sort component that understands this compacted format to process the data more efficiently.

This process might result in an increased requirement for 32 K work file allocations compared to 4 K. However, as this enhancement targets larger sorts (of small rows), it does help to improve separation of longer-running (larger) sorts in 32 K work files from OLTP (smaller) sorts in 4 K work files.

It also can improve performance as it reduces getpage activity by using an eight times larger page size.

Note: Customers might need to increase their 32 K work file allocations; both work file data sets and work file buffer pools because of DB12 using 32 K work files where 4 K work files were used previously.

Performance results from improved use of 32 K work files

To evaluate the potential benefits of this enhancement, several performance tests were conducted that used different types of workloads. The following results were observed:

- By using the IBM Retail Data Warehouse workload, a 6% reduction in Db2 CPU time and a 6% reduction in the total number of getpages was achieved.
- A 10% reduction in Db2 CPU time and total number of getpages was measured for several other real-world workloads, such as WebSphere Portal, SAP BW, and workloads provided by customers.

Note: The reduction in getpage activity is for the entire workload. The reduction in work file buffer pool getpage activity was much greater than 10%. A maximum eight times reduction (because we go from 4 K to 32 K pages) is not expected as typically not all queries qualify for this enhancement.

8.5.4 Reading sort results directly from the sort tree

Db2 11 can avoid the final write to a work file for the last sort in the query if that result can be contained in the sort tree. In Db2 12, this feature is extended to many types of intermediate sorts, such as sorts for joins. This enhancement results in reduced CPU time, work file getpage activity, and contention for shared sort resources (sort work file page sets and sort buffer pools).

Performance results for reading results from the sort tree

To evaluate the potential benefits of this enhancement, several performance tests were conducted by using different types of workloads. The following results were observed:

- By using the IBM Retail Data Warehouse workload, a 1% reduction in Db2 CPU time and a 5% reduction in the total amount of work file getpage activity was achieved.
- A 10% reduction in work file getpage activity was measured when the WebSphere Application Server Portal and SAP BW workloads were used.

8.5.5 Extensions to sort avoidance for OLAP functions

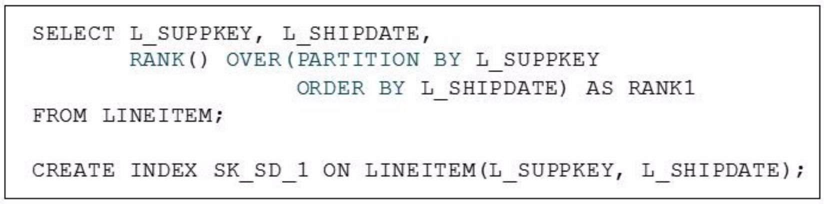
Online Analytical Processing (OLAP) functions return ranking (RANK and DENSE_RANK), row numbering (ROW_NUMBER), and aggregation information as a scalar value in the result of a query. Db2 11 supports sort avoidance if an index matches the ORDER BY clause with an OLAP function, but not if a PARTITION BY is involved. Starting in Db2 12, sort avoidance is extended to OLAP functions that combine PARTITION BY and ORDER BY.

Support for sort avoidance for OLAP functions that use a PARTITION BY clause includes the following restrictions:

- For ROW_NUMBER, RANK, and DENSE_RANK, Db2 applies only sort avoidance to one function in the SELECT list.
- With GROUP BY the query must have the same order for the GROUP BY and the OLAP function.
- The optimizer can reorder PARTITION BY columns to match an index order.

Example

The example that is shown in Figure 8-17 shows a query that combines PARTITION BY and ORDER BY in an OLAP function. Db2 12 can avoid the sort by using an appropriate index (SK_SD_1). Its definition is shown below the query.



```
SELECT L_SUPPKEY, L_SHIPDATE,
       RANK() OVER(PARTITION BY L_SUPPKEY
                   ORDER BY L_SHIPDATE) AS RANK1
FROM LINEITEM;

CREATE INDEX SK_SD_1 ON LINEITEM(L_SUPPKEY, L_SHIPDATE);
```

Figure 8-17 Sort avoidance for OLAP functions with PARTITION BY clause

Performance results

Performance tests conducted to evaluate this enhancement showed queries that used RANK(), DENSE_RANK(), and ROW_NUMBER() can have up to 50% improvement in elapsed and CPU time in Db2 12 (compared to Db2 11) when sort avoidance is used.

8.5.6 Sparse index enhancements

Sparse index usage continually enhances in recent Db2 releases to provide similar support to the hash join functionality that is available in most competitive DBMSs.

Db2 12 adds some incremental improvements to extend sparse index support to the VARGRAPHIC data type, and allows for better memory use by sparse indexes.

The first memory-related sparse index enhancement includes improved allocation of memory across multiple usages of a sparse index within the same query. In past releases, the optimizer estimate was highly depended on to determine the allocation of memory for each sparse index. In Db2 12, the sort component improved its algorithms to adjust the type of sparse index that is built to optimize memory usage, and to reduce getpage activity when a sparse index must overflow to a work file.

Another memory-related space index enhancement in Db2 12 is that when building a sparse index, sort attempts to trim the information that must be stored. Similar to the sort key length reduction enhancement, a sparse index can avoid repeating the key from the data if values are equal and of fixed length.

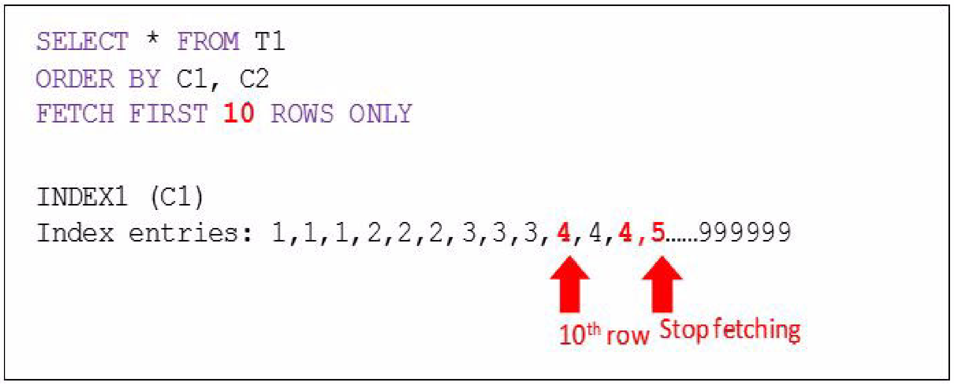
Also, when the key prefix is the same for all rows in the sparse index, the key is truncated to avoid key duplication. In addition, for variable length rows, trailing blanks can be truncated to reduce the amount of memory that is required to contain the sparse index key.

Sort work files can be an area of contention that can become a bottleneck for performance. By improving sparse indexes in Db2 12 where they use fewer work file resources, Db2 can help to reduce the demands on sort work files, and therefore allow for greater scalability in cases where contention for sort work files was previously a bottleneck.

8.5.7 Partial sort avoidance for FETCH FIRST

Although Db2 can avoid many sorts with an index that provides the required order, Db2 12 introduces the concept of partial sort avoidance with FETCH FIRST n ROWS when the order is provided by an index for the leading columns only. This performance enhancement allows the sort to stop if a partial order is provided by the index into the sort. This feature is particularly useful because more columns often that match the ORDER BY than are in the index, or trailing index columns that do not match the ORDER BY. This feature is available in V12R1M100 and requires a rebind.

In previous Db2 releases, a sort is avoided completely if the ORDER BY matches an index. However, consider the sample query that is shown in Figure 8-18 that includes an index on C1, a query with ORDER BY C1, C2 FETCH FIRST 10 ROWS ONLY. A sort cannot be avoided as the index does not contain all ORDER BY columns. In Db2 12, this feature allows the sort to stop requesting rows from runtime after Db2 has fetched past the FETCH FIRST n value (10 in this example), and hits the next change in the value of C1.



```
SELECT * FROM T1
ORDER BY C1, C2
FETCH FIRST 10 ROWS ONLY

INDEX1 (C1)
Index entries: 1,1,1,2,2,2,3,3,3,4,4,4,5......999999
```

Figure 8-18 Partial sort with ORDER BY an FFNR

In previous releases, all rows were fetched and sorted into a C1 and C2 sequence, which resulted in potentially millions of rows before fetching the top 10 rows. In Db2 12, after the 10th row is reached (the FFNR limit), Db2 continues to fetch until the value of C1 changes.

Because only 10 rows are returned in the result, the process can stop at that point. The change in C1 value ensures all possible values that can sort into the top 10 of the result set were retrieved from the table. In this case, only 12 rows are fetched and sorted instead of the entire table.

Performance results

Consider the following queries (as shown in Example 8-11) on the LINEITEM table:

- L_ORDERKEY has an index
- L_TAX has no index
- L_DISCOUNT has no index
- Approximately 45% of the rows have a value of L_DISCOUNT > 0.05

Example 8-11 FFNR partial sort enhancement

```
Q1: SELECT *
FROM LINEITEM
ORDER BY L_ORDERKEY, L_TAX
FETCH FIRST 10 ROWS ONLY
Q2: SELECT *
FROM LINEITEM
WHERE L_DISCOUNT > 0.05
ORDER BY L_ORDERKEY, L_TAX
FETCH FIRST 10 ROWS ONLY
Q3: SELECT *
FROM LINEITEM, ORDERS
WHERE L_DISCOUNT > 0.05 AND L_ORDERKEY=O_ORDERKEY
ORDER BY L_ORDERKEY, L_TAX
FETCH FIRST 10 ROWS ONLY
```

When running these queries with different values for the FFNR clause, orders of magnitude of improvement in CPU and elapsed time were observed. The improvements increase as the number of rows returned by FFNR clause decrease, as listed in Table 8-1.

Table 8-1 CPU and elapsed time for queries

	#rows in FRNR clause	**Elapsed time**		**Times improvemen t**	**CPU time**		**Times improvemen t**
		Db2 11	**Db2 12**		**Db2 11**	**Db2 12**	
Q 1	10,000	74.7480	0.0393	1903	61.4434	0.0386	1590
	1,000	75.0092	0.0211	3547	61.6551	0.0210	2935
	100	74.5297	0.0003	230030	40.4655	0.0003	134437
	10	74.5230	0.0002	325428	40.4831	0.0002	203433
Q 2	10,000	74.8117	0.0512	1461	40.2551	0.0506	796
	1,000	74.7322	0.0314	2379	40.4485	0.0312	1298
	100	74.6439	0.0003	220840	30.3317	0.0003	95684
	10	74.7781	0.0002	415434	30.4168	0.0002	186606

Q3	10,000	111.6439	0.0838	1333	109.4705	0.0831	1318
	1,000	110.5426	0.0555	1992	108.4232	0.0553	1962
	100	101.4485	0.0004	226448	97.6504	0.0004	226043
	10	101.2295	0.0002	565528	97.3445	0.0002	586412

Table 8-1 on page 280 also lists the following results for Query 2 (Q2):

- Fetching 10,000 rows results in over 796 times improvement
- Fetching 1,000 rows results in over 1,298 times improvement
- Fetching 100 rows results in over 95,684 times improvement
- Fetching 10 rows results in over 186,606 times improvement

8.5.8 Avoiding sort for ordered RIDs

Before Db2 12, no checking in the RID sort code was available to identify cases where the input to RID sort is in the proper order. (It is common for RIDs that are collected from a clustering index to be in the correct order.)

Similarly, RIDs that are collected from matching access to any index with a low full key cardinality relative to the table cardinality result in a long list of qualified RIDs that are ordered. The same can be said for matching access to a skewed key value on a high full key cardinality index. RID chains for a single key value are guaranteed to be in order with Db2 (type 2) indexes, as was not the case with type 1 indexes. However, the support for type 1 indexes was removed in Db2 Version 6 and type 2 indexes are the only supported index type since.

Db2 12 was enhanced to identify and avoid sorting pre-sorted RIDs on input to RID sort.

Example 8-12 shows a use case for this enhancement. The index on COL2 has a low cluster ratio of only 5% and list prefetch access is selected. Before Db2 11, running this query required a RID sort of all 50,000 qualifying RIDs.

Example 8-12 RID list sort avoidance for ordered RID lists

```
SELECT MAX(COL5)
FROM TABLE
WHERE COL2 = ?;   <<<<<  Predicate qualifies 50,000 rows
```

In Db2 12, Db2 can avoid most of the overhead that is involved with the RID sort by recognizing whether a “chunk” of RIDs that is passed to the sort component are ordered on input to RID sort.

Performance results for ordered RID list sort avoidance

The following performance results were observed:

- Sweet spot queries showed up to 20% reduction in CPU time. A good example of such a sweet spot query is shown in Example 8-12, which features an equal predicate selecting a single value by using an index where the RIDs for that value are in the correct order when they are retrieved from the index.

- Custom multi-index workload: Up to 20% reduction in CPU time for single index RID list access by using the clustering index. Queries in this workload that benefitted most had pre-ordered RID-list ranging 10,000 - 1 million entries.
- Custom multi-index workload: Up to 10% reduction in CPU for multi-index RID list access when one leg uses the clustering index.

8.6 Migration from Db2 11 to Db2 12

After migrating your Db2 11 systems to Db2 12, you can start to explore new features and take advantage of the Db2 performance enhancements that were introduced in Db2 12.

8.6.1 IBM Query workload performance after Db2 12 migration

To evaluate Db2 12 query performance, a Db2 system was migrated from Db2 11 to Db2 12 by using a typical migration scenario. The workload used was a 30 GB IBM Retail Data Warehouse benchmark, with queries started from SQLPL stored procedures. The 22 IBM Retail Data Warehouse queries are written to use literals and parameter markers, which resulted in a total of 44 queries in 44 stored procedures. The queries were run in the following migration stages:

- Db2 11 in NFM
- After migrating to Db2 12 at function level V12R1M100 without rebind
- After migrating to Db2 12 at function level V12R1M100 after a rebind with APREUSE(ERROR)
- After migrating to Db2 12 at function level V12R1M100 after a rebind with default options i.e. APREUSE(NONE)
- After migrating to Db2 12 at function level V12R1M500 after a rebind with default options i.e. APREUSE(NONE)

As expected, all queries use the same access path when running the workload in Db2 11, Db2 12 without rebind, and Db2 12 after rebind with APREUSE(ERROR). After allowing Db2 to select new access paths by rebinding by using the APREUSE(NONE) option, approximately 50% of the queries used a different access path compared to Db2 11.

These new access paths resulted in more performance improvements, as shown in Figure 8-19.

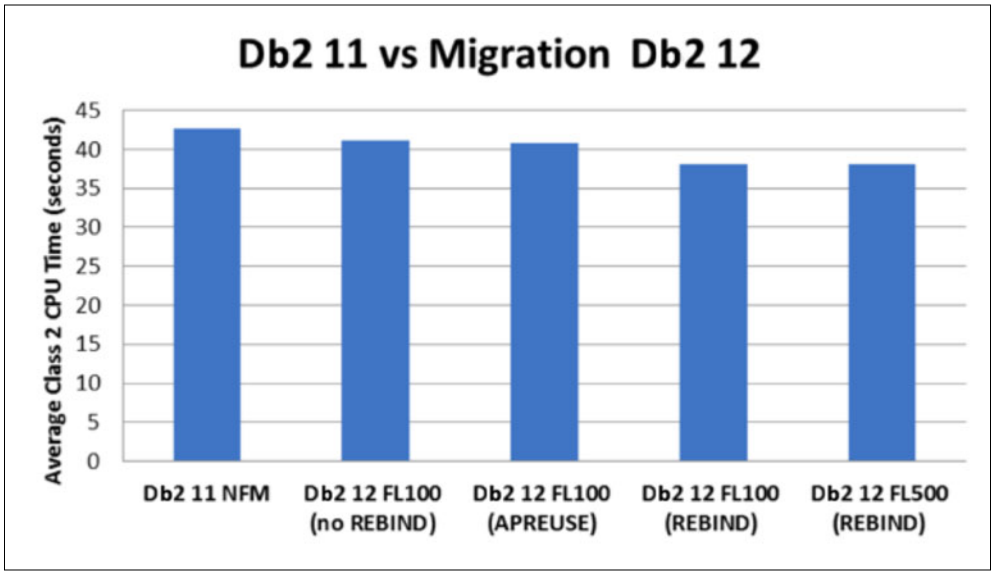


Figure 8-19 Comparison of Class 2 CPU time for each of the five scenarios

Comparing the Db2 class 2 CPU time to the run in Db2 11, the following results were observed:

- An approximately 3.8% CPU time improvement after migrating to Db2 12.
- Migrating to Db2 12 and rebinding with APREUSE(ERROR) resulted in a 4.3% improvement.
- An approximate 11% CPU time improvement at function level V12R1M100 and V12R1M500 after rebinding with default options; for example, APREUSE(NONE). This result allowed Db2 to go through complete access selection, which opened up to new or enhanced access paths.

8.7 Query regression workloads

A total of 14 Db2 internal query regression workloads comparing Db2 12 to Db2 11 were measured. Some new workloads were added during Db2 12 development cycle to highlight some of the new workload growth opportunities in complex transactional processing and real-time analytics in Db2.

As shown in Figure 8-20, the performance results vary significantly for each of the measured workloads. Most of the performance enhancements are the result of the use of new access paths in Db2 12. In addition, 2 - 10% CPU time improvement was realized for queries without any access path changes in various workloads because of Db2 engine performance enhancements.

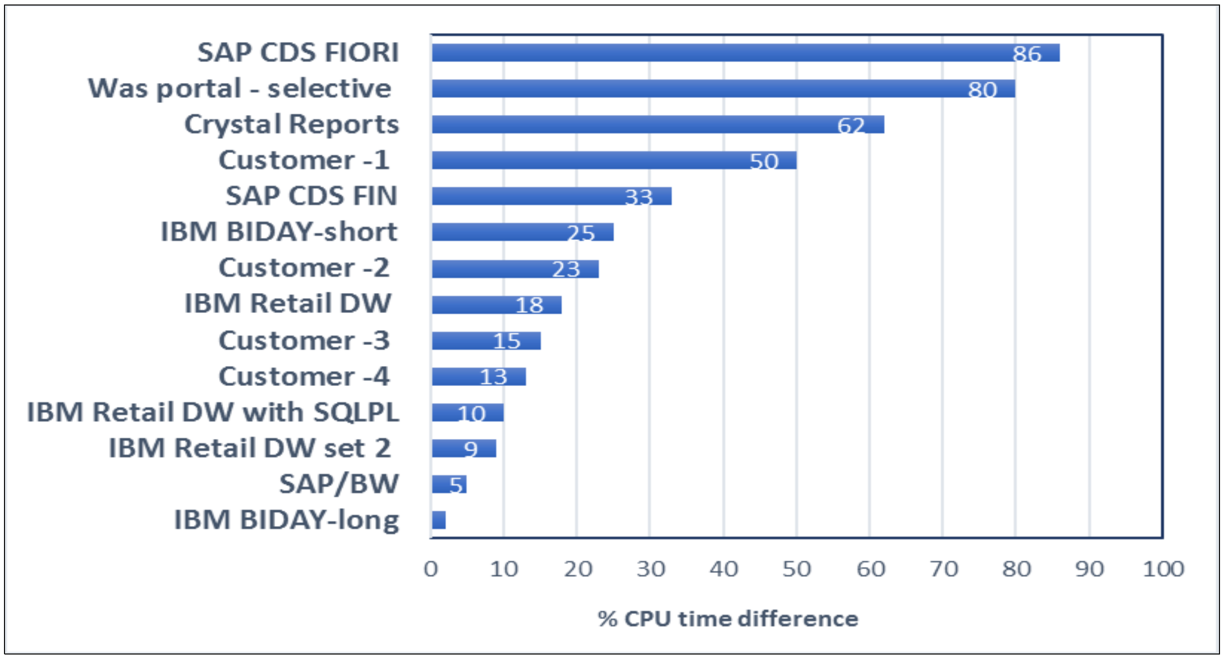


Figure 8-20 Internal Db2 query regression workload results for Db2 12 for z/OS

8.7.1 Characteristics of query workloads and improvement

This section describes the characteristics of the query workloads and how the workloads benefit from the various query enhancements that were described in this chapter.

Figure 8-21 on page 285 shows the query workloads or set of queries that demonstrate 30 - 86% CPU and elapsed time reduction compared to Db2 11. These query workloads take advantage of many Db2 12 enhancements and include the following general characteristics:

- Complex OJs, UNION ALL, UDFs, and Table UDFs
- Combinations of Table Expressions, Views, and OJs
- VARGRAPHIC data type
- Disorganized data, poor CR indexes
- Nearly 100% NEW Access Paths versus Db2 11

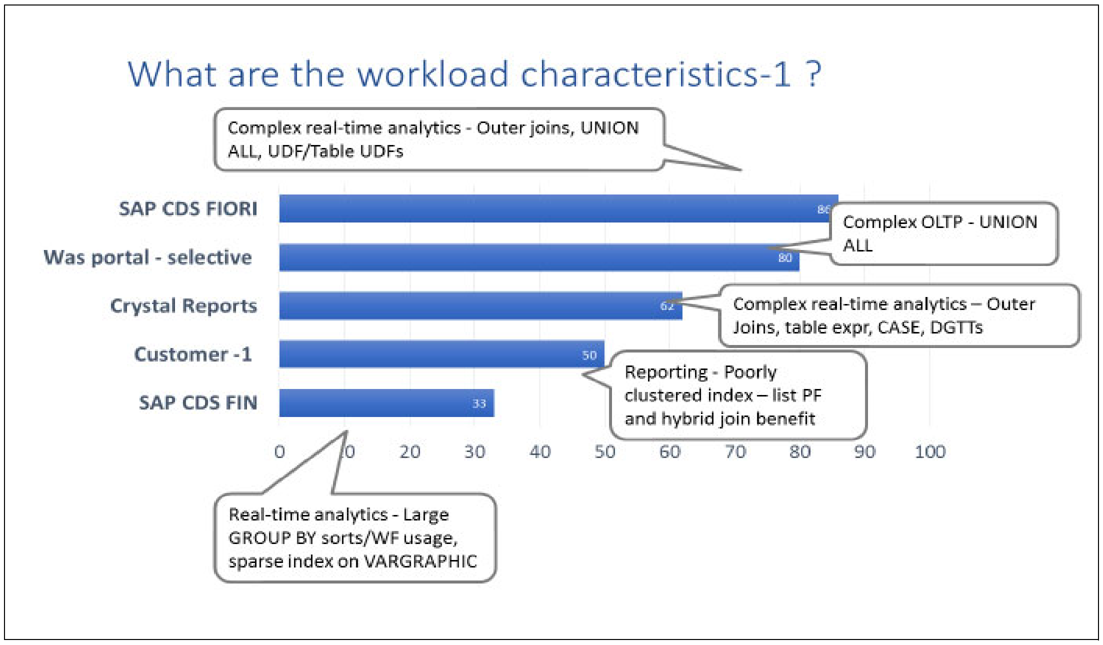


Figure 8-21 Workload characteristics

Figure 8-22 shows the query workloads or set of queries that demonstrate 10 - 25% CPU time reduction compared to Db2 11. These workloads are traditional data warehouse workloads with sort intensive queries or queries with lots of aggregations.

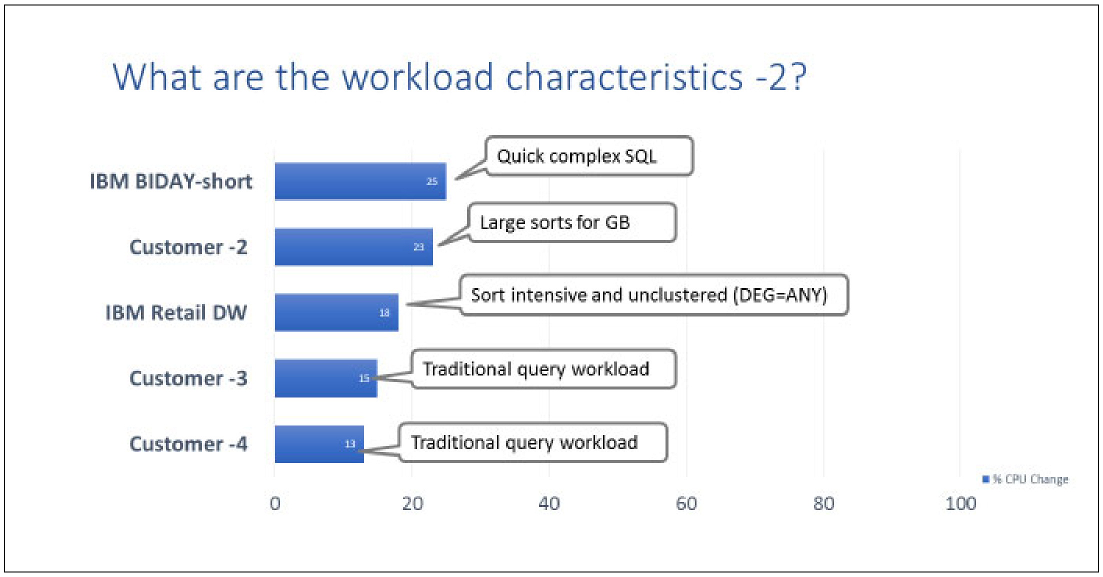


Figure 8-22 Workload characteristics for IBM BIDAY

Query workloads in Figure 8-23 show 0 - 10% range of CPU time reduction with Db2 12 compared to Db2 11. These workloads fall into two categories: Simple queries and queries where most of the processing is because of data scanning. The data scanning queries are good candidates for IBM Db2 Analytics Accelerator for z/OS and were generally not targeted for Db2 12 performance improvements. Fewer queries in this set of workloads see access path changes with Db2 12.

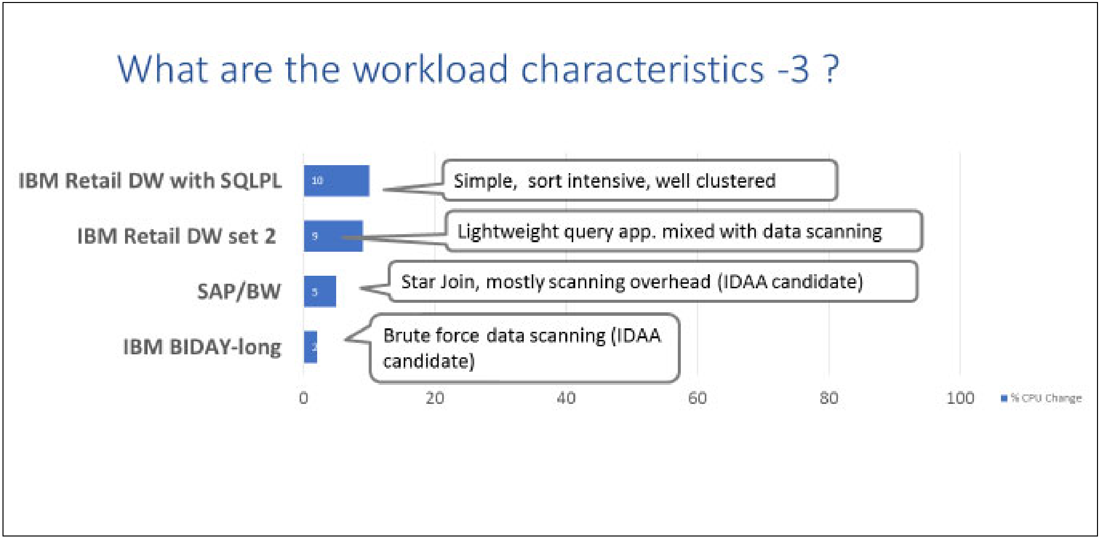


Figure 8-23 Workload characteristics for IBM Retail Data Warehouse

8.7.2 Access path change and performance improvement

In the IBM query workloads that were described in this section, the queries with new access paths tended to see larger improvements compared to the queries without access path changes.

Typical CPU and elapsed time reduction from the queries with new access paths is 10 - 99%, while the queries without access path changes resulted in the improvements of 0 - 20%.

Many Db2 users tend to use APREUSE for their static SQL statements to keep their known stable access paths during the migration of Db2. This approach is still recommended to reduce the risk from the migration.

However, after the successful migration of Db2 12, we strongly recommend rebinding without APREUSE in a controlled manner to use the full potential of Db2 12.

9

Utilities

Db2 12 includes multiple utilities enhancements and features, such as new behavior in LOAD and REORG, and new options in UNLOAD and RUNSTATS.

This chapter includes the following topics:

9.1 LOAD and REORG enhancements

Several enhancements were made in Db2 12 LOAD and REORG to improve their performance and usability. Performance evaluations of these enhancements are described in this section.

9.1.1 LOAD REPLACE improvements for empty PBR partitions

In older Db2 versions, LOAD REPLACE of an empty partition scans all indexes of that table. In Db2 12, partition LOAD into an empty partition skips the index scan of non-partition indexes, which reduces CPU and elapsed time.

Performance measurements

Measurements were performed on a UTS table of 100 million rows and 20 partitions with one, two, and five non-partitioning indexes on a four-way zEC12. They showed almost 100% CPU time and 97% elapsed time reduction, with no more getpages for indexes when loading into an empty partition with DUMMY.

Another measurement was performed on the same UTS table, where 100 million rows were loaded into an empty partition. Loading into an empty partition with data showed up to 10% CPU time and 6% elapsed time reduction. It also showed an 11% reduction in the number of getpages for indexes. The improvement is from the sort and index build phases.

9.1.2 zIIP offload during RELOAD phase for LOAD and REORG

In older Db2 versions, part of the unload, sort, and index build CPU time of the LOAD and REORG utilities was eligible for offload to zIIP. In Db2 12, the CPU time of the RELOAD phase of the LOAD and REORG utilities becomes almost 100% eligible for zIIP offload. This feature is available starting with V12R1M100.

Performance measurements

Measurements that use a UTS table space with 100 million rows and 20 partitions with no index on a four-way zEC12 showed that up to 99% of LOAD CPU is offloaded to zIIP, and up to 59% of REORG CPU is offloaded to zIIP in Db2 12. The zIIP eligible CPU in LOAD is from the RELOAD phase. The zIIP eligible CPU in REORG is from the UNLOAD and RELOAD phases.

9.1.3 LOAD SHRLEVEL CHANGE PARALLEL for PBG table spaces

Db2 11 implemented parallelism support for LOAD RESUME SHRLEVEL CHANGE with PBR and classic partitioned table spaces from a single input file, but not for PBG table spaces. This enhancement in Db2 11 showed a significant reduction in elapsed time with increases in class 2 CPU time because of contention between parallel tasks. This contention depends on free space, row versus page level locking, and many other factors. In Db2 12, parallelism support for LOAD RESUME SHRLEVEL CHANGE is added for PBG table spaces. This enhancement is available starting with V12R1M100.

Performance measurements

Measurements were conducted by using a UTS PBG table space with 100 million rows and 20 partitions with six indexes, random input (regarding cluster key), page level locking, default free space parameters, and an initially empty table.

The results showed a 76% reduction in elapsed time and 48% increase in CPU time with 24 degrees of parallelism used (default). The CPU time increased from 25.2 seconds with no parallelism to 37.6 seconds with parallelism.

9.2 Statistics enhancements

Many enhancements were made in Db2 12 RUNSTATS and inline statistics to improve its performance and usability. The performance evaluations that are related to some of these enhancements are described in this section.

9.2.1 RUNSTATS COLGROUP performance improvement

Before Db2 12, when statistics are collected on a COLGROUP with matching indexes, redundant statistics collection from the table side and the index side was used. In Db2 12, an algorithm was added to detect that COLGROUP columns match with the index columns and Db2 collects only the statistics from the index side.

Performance measurements

Measurements that use a UTS table space with 100 million rows and 20 partitions with six indexes on a four-way zEC12 showed up to 25% improvement in CPU time and up to 14% improvement in elapsed time for Db2 12 RUNSTATS COLGROUP INDEX compared to Db2 11.

9.2.2 LOAD PARALLEL STATISTICS on COLGROUP

Although the new PARALLEL option for LOAD was introduced in Db2 11, LOAD PARALLEL STATISTICS did not support inline statistics for the COLGROUP option. Db2 12 improves LOAD PARALLEL STATISTICS by supporting the COLGROUP option. This improvement allows LOAD PARALLEL STATISTICS to provide more information to Db2 by collecting more distribution statistics on columns that customers expect to use in predicates.

This enhancement to LOAD PARALLEL STATISTICS greatly improves the usability and accuracy of the filter factors that are determined by Db2, which leads to better optimization choices. Therefore, query performance is improved with better filter factor information in the Db2 catalog. COLGROUP statistics summarize the data distribution in the table.

The following COLGROUP statistics are collected:

- Frequency distribution for the group of columns
- Cardinality values for the columns that are specified in the COLGROUP
- Histogram statistics on the COLGROUP

The COLGROUP keyword has the following options: FREQVAL COUNT MOST | LEAST | BOTH.

This feature is available starting with V12R1M100.

Performance measurements

Measurements that use a UTS table with 100 million rows and 20 partitions with no index on a four-way z196 processor showed no regression for LOAD PARALLEL STATISTICS TABLE COLRGROUP.

Compared to running LOAD and subsequent RUNSTATS TABLE COLGROUP to collect distribution statistics, this new enhancement in LOAD PARALLEL STATISTICS TABLE COLGROUP showed up to 46% reduction in elapsed time with up to 18% CPU increase because of more parallelism.

9.2.3 Frequency statistics on indexes during inline statistics

In Db2 11, RUNSTATS must be run to collect MOST, LEAST, and BOTH frequency statistics on indexes. This function is supported in running Db2 12 inline statistics after V12R1M500 or above is activated.

Performance measurements

Measurements the use a UTS table space with 100 million rows and 20 partitions with two or five non-partitioning indexes on a four-way zEC12 showed no regression in inline statistics with LEAST, MOST, or BOTH options.

Comparing the execution of the LOAD/REORD/REBUILD utility followed by RUNSTATS with the LEAST/MOST/BOTH option against running these utilities with Db2 12 with inline statistics with the LEAST/MOST/BOTH option, the following performances improved significantly:

- REBUILD: Up to 29% CPU and 59% elapsed time reduction
- REORG: Up to 26% CPU and 39% elapsed time reduction
- LOAD: Up to 24% CPU and 37% elapsed time reduction

9.2.4 PROFILE support for inline statistics

Db2 12 also provides further integration of the Db2 optimizer’s recommendations with RUNSTATS. In Db2 11, the optimizer externalizes information to a catalog table and optionally to an explain table about RUNSTATS recommendations based on the identification of missing or conflicting statistics that might benefit the SQL being bound, prepared, or explained. An extra manual step was also required to convert those recommendations into RUNSTATS statements to be run.

In Db2 12, the RUNSTATS PROFILE automatically is updated when the optimizer makes new statistics recommendations. If a profile does not exist, a new profile is created by merging the existing statistics in the catalog with the new recommendations.

Updates to a profile trigger a recommendation from the DSNACCOX stored procedure that RUNSTATS should be run for that table. Automatically creating or updating the statistics profiles is performed only when the value of the new STATFDBK_PROFILE DSNZPARM is set to YES.

These enhancements are a significant step forward in RUNSTATS simplification and integration with maintenance processes.

An overview of the RUNSTATS integration steps with the optimizer in Db2 11 and Db2 12 is shown in Figure 9-1.

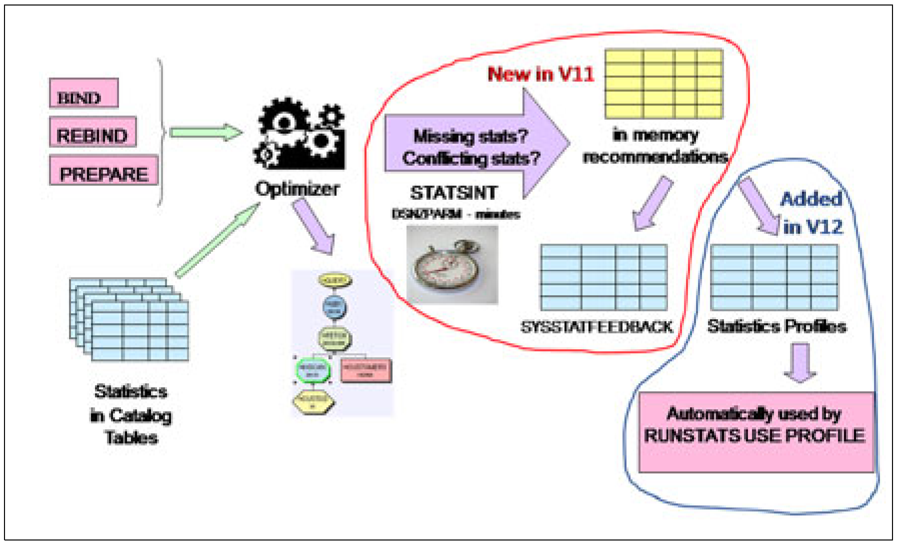

BIND
REBIND
PREPARE
Optimizer
Statistics in Catalog Tables
New in V11
Missing stats? Conflicting stats?
STATSINT
DSNZPARM - minutes
in memory recommendations
SYSSTATFEEDBACK
Added in V12
Statistics Profiles
Automatically used by RUNSTATS USE PROFILE

Figure 9-1 Optimizer externalization of missing statistics

To enable the usage of the optimizer's recommendations, the customer specifies USE PROFILE, which is now also supported for inline stats in Db2 12. With this enhancement, all methods of Db2 RUNSTATS collection now support the use of profiles. For more information, see *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

The USE PROFILE support allows inline statistics to use a previously stored statistics profile to gather statistics for a table. The statistics profile is created by using the RUNSTATS option SET PROFILE and is updated by using the UPDATE PROFILE option.

USE PROFILE support is added to the inline statistics functionality for the following utilities:

- REORG TABLESPACE STATISTICS
- LOAD STATISTICS

This feature is available in V12R1M500 or above.

Performance measurements

Measurements that use a UTS table space with 100 million rows and 20 partitions with two or five non-partitioning indexes on a four-way zEC12 showed no regression in inline statistics by using the USE PROFILE option.

Compared to running LOAD/REORG and subsequent RUNSTATS TABLE USE PROFILE, running LOAD or REORG STATISTICS USE PROFILE showed up to 26% reduction in elapsed time with 2% CPU increase because of more parallelism.

9.2.5 Statement cache invalidation in RUNSTATS and inline stats

The dynamic statement cache invalidation behavior changes for RUNSTATS in Db2 12. Before Db2 12, RUNSTATS and inline STATISTICS always invalidated the dynamic statement cache. In Db2 12, the default behavior for RUNSTATS and STATISTICS is to *not* invalidate the dynamic statement cache, except in the following cases:

- When RUNSTATS is run with the REPORT NO and UPDATE NONE options
- When RUNSTATS is run with RESET ACCESSPATH option

This new behavior is intended to reduce the cost of new prepares after RUNSTATS is run. Similarly, utilities, such as LOAD, REORG, or REBUILD INDEX, invalidate only statements in the cache if the object was in a pending state before the utility was run. The theory behind these behavior changes is that it is expected that a REORG or RUNSTATS is not necessarily run because of a bad access path; therefore, utilities that are run for other purposes should not disrupt the access path.

To invalidate the cache, the user must use the new keyword option INVALIDATECACHE YES in RUNSTATS or inline STATISTICS. The keyword INVALIDATECACHE can be used in V12 after the activation of V12R1M500 or higher.

Performance measurements

Measurements were performed by using a UTS table space with 100 million rows and 20 partitions with six indexes on a four-way zEC12. The statement cache was populated when running RUNSTATS. These measurements showed no significant difference in Db2 12 RUNSTATS, or LOAD and REORG, with INVALIDATECACHE YES compared to Db2 11.

9.2.6 RUNSTATS TABLESPACE LIST INDEX improvement

Before Db2 12, when a LIST is defined with the PARTLEVEL keyword and the following RUNSTATS TABLESPACE LIST INDEX statement is used, statistics are collected on all the specified indexes for each partition for every table space part processed. This configuration results in excessive elapsed time and contention during the update of the catalog tables.

In Db2 12, statistics for the indexes are collected only on the first part that is scanned for a specific table space in the defined list. No index statistics are collected on any other parts for the same table space scanned. This enhancement is available in V12R1M100.

Performance measurements

Measurements that used a UTS table space with 100 million rows and 20 partitions with six indexes on a four-way zEC12 showed up to 83% improvement in CPU and elapsed time comparing Db2 12 against Db2 11 when the RUNSTATS TABLESPACE LIST TABLE ALL INDEX ALL statement was used.

9.2.7 Other RUNSTATS enhancements

In addition to the enhancements that were described in this chapter, RUNSTATS features the following enhancements:

- Cluster ratio formula change

  Db2 12 includes a minor improvement to the cluster ratio formula that is used by the RUNSTATS utility. It reduces the effect that unclustered inserts have on the cluster ratio of the clustering index. This enhancement improves the alignment that the cluster ratio formula has with the running behavior of dynamic prefetch. It can also extend the time between table space REORGs recommended by stored procedure DSNACCOX based on the cluster ratio. No requirement is needed to collect statistics by using the new formula before static rebinds or dynamic SQL in Db2 12 is run.

- Automatic determination of the number of frequencies to collect

  Db2 12 offers automatic determination of the number of frequencies to collect when RUNSTATS is used. The first concern to address when RUNSTATS is used is to determine which statistics to collect. After it is determined which statistics are to be collected, the question of how many frequencies to collect remains. The default is to collect the top 10, but the truth is that only the top one can be skewed, or the top 20 or 50.

No single default is applicable to all columns and all tables; it is entirely data-dependent. Db2 12 provides the capability to allow RUNSTATS to continue collecting frequency statistics until the data is no longer skewed, with an upper limit of the top 100 values to avoid over-collection. This capability avoids the situations where too few values are collected for the optimizer to estimate filtering correctly. It also removes the burden on the DBA to specify an appropriate value. To trigger this enhancement, specify the FREQVAL option without the COUNT keyword.

9.3 REGISTER NO option in UNLOAD and RUNSTATS

Db2 12 adds a REGISTER NO/YES option to the UNLOAD and RUNSTATS utilities. The new option applies only to a Db2 data sharing system and specifies whether pages that are read by the UNLOAD and RUNSTATS utilities are registered in the coupling facility. This option is in effect only when ISOLATION UR and SHRLEVEL CHANGE behavior are used.

The UNLOAD and RUNSTATS utilities read table spaces or partitions sequentially. If the target objects are GBP-dependent, UNLOAD or RUNSTATS can drive a large volume of register page list operations to the coupling facility.

The Register Page List (RPL) operation is a list command that is used to register a set of pages in a GBP structure to keep the data consistent between data sharing members. Although RPL is a more efficient operation than a single page registration, many concurrent RPL requests can cause high Coupling Facility (CF) CPU utilization. In the worst case, this issue can also affect the performance of other structures in the same CF.

When the new keyword REGISTER NO is used with ISOLATION UR and SHRLEVEL CHANGE, GBP accesses are avoided. The pages that are read by the UNLOAD and RUNSTATS utilities are not registered in the CF.

Pages that are in the coupling facility are not processed by the UNLOAD and RUNSTATS utilities. This option aims to reduce the overall CF activity when retrieving the latest and consistent data through UNLOAD or RUNSTATS is not required.

When REGISTER YES is used, the behavior is the same as in Db2 11. That is, the UNLOAD and RUNSTATS utilities register pages to the CF and read the pages from the CF as needed. The default behavior is NO for UNLOAD with ISOLATION UR, except for the base table space with XML and LOB columns. The default is YES for the RUNSTATS utility. The new keyword REGISTER YES/NO is available after the migration to Db2 12 (Function Level 100).

9.3.1 Performance measurements

Performance measurements were done by using 20 concurrent utility jobs against 20 tables (each table is approximately 5 GB) with REGISTER YES and NO in a two-way data sharing environment. The objects that were used are GBP-dependent.

When running with the REGISTER NO option, the CF utilization in RMF Activity Reports dropped to close to 0%. This result occurred because GBP-dependent getpage requests and RPL requests are greatly reduced, as shown in Table 9-1 on page 294 and Table 9-2 on page 294. DBM1 SRB time in the statistics trace is reduced because the registration is done under SRB that is running in DBM1 address space.

Table 9-1 UNLOAD utility with REGISTER YES and NO

UNLOAD	Register YES	Register NO
Accounting Report		
Average Class 1 Elapsed Time	2:20.23722	2:20.38254
Average Class 1 CPU Time	46.162989	46.146585
Address Space TCB Time	0.3676	0.385115
Address Space PREEMPT SRB	28.434019	16.481655
Address Space NONPREEMPT SRB	0.1603	0.172527
#getpage BP3	266155 K	266155 K
#GBP dep getpage BP3	12095 K	0
Page registration per sec	762.76	0

Table 9-2 RUNSTATS utility with REGISTER YES versus NO

RUNSTATS	REGISTER YES	REGISTER NO
Accounting Report		
Average Class 1 Elapsed Time	1:30.30762	1:40.14576
Average Class 1 CPU Time	1:30.30762	1:07.66963
Address Space TCB Time	0.169684	0.158226
Address Space PREEMPT SRB	11:35.291251	11:07.972913
Address Space NONPREEMPT SRB	0.045811	0.0321
#getpage BP3	26587 K	26587 K
#GBP dep getpage BP3	4189.9 K	0
Page registration per sec	1176.97	0

9.3.2 Usage considerations

The main purpose of REGISTER NO option is to reduce the effect on the CF, although you can also expect that some CPU time is also reduced.

Although this option can greatly reduce the requests to GBP structures, pages are always read from the disk and the latest updates might not be reflected, depending on the timing of castout operations.

As expected when the ISOLATION UR process is used, pages that were read might not be committed. When you are running the RUNSTATS utility immediately after many DELETE or INSERT operations, it is recommended to use REGISTER YES (default for RUNSTATS) to reflect the latest information.

9.4 Backup and RECOVER

In Db2 12, RECOVER always uses one parallel task to overlap read and write during the RESTORE phase, even if the PARALLEL keyword is not specified. This configuration results in more efficient processing because reading the pages from the image copies and restoring pages to the buffer pool processes can occur asynchronously.

The use of hardcoded DD statements to pre-allocate the image copy data sets on tape in the RECOVER job step is no longer supported and causes allocation errors.

This feature is available in V12R1M100.

9.4.1 Performance measurements

RECOVER performance of a single table space and a list of partitions of a partitioned table space showed up to 27% CPU improvement in the RESTORE phase compared to Db2 11.

Because of this improvement in the RESTORE phase, RECOVER showed no regression overall.

9.5 Decompression improvement in utilities

In Db2 11, decompression improvement was introduced to access rows in a compressed table space. This feature substantially reduced CPU usage in Db2 11 when many compressed rows are processed by SQL queries because Db2 partially decompresses only the necessary data and applies software algorithm. In Db2 11, this feature was limited to SQL access and not used by utilities.

In Db2 12, the software algorithm improvement during decompression is also enabled for utilities. This enablement demonstrated noticeable improvement in utilities under zEC12. This feature is available in V12R1M100. Around the same time, the system z13 processor also provides better compression performance by using hardware improvement. Because z13 improvement applies for Db2 11 and 12, we observed less improvement in DB12 that used z13 compared to the case that used zEC12.

9.5.1 Performance measurements

Measurements were run against a PBR table space. This table space was defined with COMPRESS YES, 20 partitions, and contained 100 million rows and six indexes. REORG, REBUILD INDEX, RUNSTATS, and UNLOAD utilities were included in these measurements. The results are listed in Table 9-3 and Table 9-4.

Utility measurements on zEC12 show more consistent 3 - 8% CPU time improvement in Db2 12 compared to Db2 11 while the measurements on z13 show less improvement than the cases of zEC12.

Table 9-3 Percentage CPU improvement in v12 versus v11 on zEC12

Utility	Delta % CPU
REORG	-3%
REORG KEEPDICTIONARY	-3%
REBUILD PI	-5%
REBUILD PI PART 1	-5%
REBUILD NPI	-1%
REBUILD NPI PART 1	0%
REBUILD 2 INDEXES	-1%
UNLOAD	-5%
UNLOAD FORMAT INTERNAL	-8%

Table 9-4 Percentage CPU improvement in v12 versus v11 on z13

Utility	Delta % CPU
REORG	1%
REORG KEEPDICTIONARY	0%
REBUILD PI	0%
REBUILD PI PART 1	-2%
REBUILD NPI	0%
REBUILD NPI PART 1	-2%
REBUILD 6 INDEXES	2%
RUNSTATS TABLESPACE	1%
RUNSTATS TABLESPACE TABLE	-7%
UNLOAD	1%
UNLOAD FORMAT INTERNAL	-2%

10

IBM Db2 Analytics Accelerator for z/OS

IBM Db2 Analytics Accelerator for z/OS (IDAA) continues to deliver enhancements in functionality and performance.

This section describes IDAA performance features that are introduced in IDAA V4 to IDAA V6. Most of the measurements that are provided in this chapter are done by using Db2 11.

This chapter includes the following topics:

10.1 Performance Enhancements for IBM Db2 Analytics Accelerator for z/OS

During recent years, the IBM Db2 Analytics Accelerator for z/OS (IDAA) was improved with new functionalities through several PTFs in version 4 and version 5. With version 6, IBM introduced cloud IDAA capabilities.

In *DB2 11 for z/OS Performance Topics*, SG24-8222, topics related to IDAA V4.1 and Db2 11 were covered. This chapter covers the enhancements starting with IDAA V4 PTF2 to V6. The performance results in this chapter are based on Db2 11, unless otherwise noted.

IDAA delivered the following features and fixes that greatly improve the load and load-related performance:

- The native loader allows Netezza® Performance Server (NPS®) on the accelerator to parse Db2 internal data directly, so data conversion is eliminated and load throughput is significantly improved.
- The time-optimized reload feature reduces most of the delete phase cost for a full table reload.
- Empty partitions and tables are skipped from UNLOAD to save elapsed and CPU time.
- The mutual exclusive lock enhancement allows more concurrency against Netezza catalogs and improves the end-to-end time of concurrent load and reload operations.
- The Db2 UNLOAD utility can now bypass an expensive BSAM process to reduce z/OS CPU cost.
- A limitation to TCP send buffer size is removed to allow higher load throughput when running UNLOAD in parallel.

Real-time analytics is one of the goals of IDAA. Thus, it is important to reduce the replication speed of the operational data from the host to IDAA to make the data available for the analytical queries.

IDAA added SQL functions, such as accelerating local static SELECT INTO statements, which support system temporal and bi-temporal tables and inline scalar UDFs, among others. IDAA also implemented the accelerator-only table concept to enable acceleration for analytics tools that use temporary data, and enabled in-database transformation for ETL processes.

IDAA also encrypts data in motion by using IPSec to protect network traffic between the z/OS system and the accelerator.

10.2 IDAA Version 4 PTF-2

This section describes the effect of enabling query acceleration on rebind packages.

10.2.1 Effect of QUERYACCELERATION ENABLE on Rebind Static Packages

After the support for accelerating static SQLs becomes available in IDAA V4, users might have the following question:

> *I have too many static packages to identify which are suitable to enable query acceleration. What is the performance affect on rebind packages, especially the ones for OLTP queries, if we blindly enable query acceleration?*

To measure the effect on OLTP workloads, experiments were conducted that used an IBM Brokerage transaction workload that performed a rebind of 230 static packages (each package containing one query) with the queries being too short to be accelerated. Without enabling query acceleration, the average elapsed time of the rebinds was 23.74 ms per package. The average Db2 CPU time of the rebind was 21.77 ms per package.

Having query acceleration enabled, only a minor performance effect was observed. The CPU and elapsed time increase from the rebind was less than 2%. Queries that were run in this workload were mixtures of simple to medium complexity. Yet, none of the queries are qualified for acceleration.

However, by using more complex query workloads, such as an IBM Retail Data Warehouse workload, a rebind of 140 accelerated static packages shows nearly 6% saving in Db2 elapsed time and 7% saving in Db2 CPU time. This saving is because after Db2 decides to accelerate the query, some Db2 bind time processes are skipped, such as access path selection, and generating runtime structures.

But when we used simple, short packages that contain one query with FETCH FIRST 1 ROWS ONLY phrase, we observed an 11% Db2 elapsed time increase and a 16% Db2 CPU time increase. These simple packages or queries had shorter average Db2 elapsed time (9.4 ms) and CPU time (7.6 ms) than the packages or queries in the IBM Brokerage transaction workload. Therefore, the effect of rebinding with enabling query acceleration is more obvious. Again, none of the simple queries are qualified for acceleration.

The effect of enabling query acceleration for simple statements can be seen at the bind or rebind. The effect is likely negligible at typical OLTP environment, nand the packages that contain simple queries might see visible cost. For the complex queries that qualify the acceleration, rebind time can be reduced.

10.3 IDAA version 4 PTF-3

This section describes the performance enhancements that are introduced in V4 PTF-3: Acceleration of local static SELECT INTO queries, and LOAD and RELOAD enhancements.

10.3.1 Acceleration of local static SELECT INTO

Local static SELECT INTO queries were not included in the initial support for static query acceleration that was delivered in IDAA V4. This feature was added in version 4 PTF-3 so that local static SELECT INTO statements are eligible for acceleration. Tests of 26 accelerated singleton select statements show overall 7.8 times speedup in Db2 elapsed time and a negligible CPU cost when compared with running in Db2.

10.3.2 Load enhancement

IDAA v4 PTF-3 provides enhancement during initial load and in handling empty partitions during load.

Initial load

Before IDAA v4 PTF-3, one of the bottlenecks during load was the conversion of data from the Db2 for z/OS internal format into Netezza's text-delimited format.

With IDAA v4 PTF-3, a new parser for Db2 for z/OS internal format (Db2 for z/OS parser) was implemented and integrated into the Netezza Performance Server (NPS). The Db2 for z/OS parser enables NPS to load data that is based on the Db2 for z/OS internal format and was unloaded from Db2 for z/OS using UNLOAD utility.

The new Db2 for z/OS parser also allows IDAA to load data into NPS without having to convert it into Netezza's text-delimited format, which saves CPU time on the Netezza host. Although the Db2 for z/OS parser is in NPS 7.1, IDAA support begins with IDAA V4.1 PTF-3.

Performance of initial load

Comparing initial load on IDAA v4 PTF-2 to IDAA v4 PTF-3 shows the following results:

- For a partitioned table space:
  - A 20 - 24% decrease in overall elapsed time for a partitioned table, depending on the numbers of columns and types of columns.
  - No major change in Db2 CPU time.
- For a non-partitioned table space:
  - Over 70% decrease in overall elapsed time for a non-partitioned table.
  - No major change in Db2 CPU time.

Note: A higher reduction was observed in non-partitioned table space. This result occurred because LOAD test against a partitioned table space was done by using 12 concurrent streams while LOAD test with a non-partitioned table space was done with single stream. When there are fewer streams, NPS requires to wait more frequently for converting input data before processing the records.

Load with empty partitions

Load performance might be negatively affected when loading tables that include many empty partitions or many empty non-partitioned tables. This affect was because of many unnecessary DSNUTILU calls to unload empty partitions or tables.

When environment variable AQT_SKIP_UNLOAD_EMPTY_PARTS=SET is specified, IDAA uses real-time statistics (RTS) to detect and skip DSNUTILU calls for empty partitions or tables. In our controlled test of loading a partitioned table with 4096 partitions that were all empty, the elapsed time was reduced from 95 seconds to 21 seconds and CPU cost was reduced from 39 seconds to close to 0 seconds, compared to the measurement with and without this enhancement.

10.3.3 Reload enhancements

IDAA v4 PTF-3 also provides performance improvements during reload.

The current implementation of reloading table data consists of the following major steps:

1. Delete old table data.
2. Insert new table data.

Therefore, the overall time for reloading table data is the sum of the data deletion and data load time.

For non-partitioned tables, the following steps are taken in one transaction:

1. Load data into a new table by using one connection, which is called the shadow table.
2. Drop the old table.
3. Rename the new shadow table to old table’s name.

Because this process avoids the delete phase entirely, a significant improvement in reload performance is gained.

For the reload of a partitioned table, the process is the same as for a non-partitioned table; however, we use multiple connections for the load phase.

Reload performance

Performance tests were conducted by using a Striper machine with full rack (N2001). The partitioned table that was used is the 1 GB of Lineitem table from IBM Retail Data Warehouse database. Loads and reloads were done by using 12 streams.

The Lineitem table is also used as a non-partitioned table. The table was loaded until the maximum size of ~64 GB (compressed) was reached. During load, this process results in ~120 GB of uncompressed data being transferred to IDAA.

Results

Comparing full table reload on IDAA v4 PTF-2 to IDAA v4 PTF-3 shows the following results:

- For the partitioned table:
  - A total elapsed time improvement of 32% where a 17% improvement comes from the use of the shadow table.
  - No major change in CPU time.

- For the non-partitioned table:
  - A total elapsed time improvement of 72% where only a 4% improvement comes from the use of a shadow table because the delete phase is not a major component of the load.
  - No major change in CPU time.

Improvements can also be expected for partial table reload on IDAA v4 PTF-3 compared to IDAA v4 PTF-2, in accordance to how much of the table is reloaded.

10.4 IDAA version 4 PTF-4

This section describes dynamic statement cache support for accelerated queries that was added in IDAA V4 PTF-4.

10.4.1 Dynamic statement cache support for accelerated queries

Before this feature was available, Db2 did not support dynamic statement caching for accelerated dynamic queries. When an accelerated dynamic query was run, it went through a full PREPARE in Db2 for z/OS.

This feature improves PREPARE performance for repeated accelerated dynamic queries; long PREPAREs become short PREPAREs by caching the prepared statements in the statement pools. No user action is required to trigger this feature.

Controlled tests on PREPAREs of seven accelerated dynamic queries showed over 70% Db2 elapsed time improvement and 80% Db2 CPU improvement.

Tests on PREPAREs of four un-accelerated dynamic queries show no performance difference with and without this feature, as expected.

10.5 IDAA version 4 PTF-5

This section describes load improvement and a load and query performance comparison of N3001 (Mako full rack) and N2001 (Striper full rack), in-database transformation, parallel incremental updates, and auto keys for incremental update enabled tables.

10.5.1 Load improvement with PI26783

APAR PI26783 (HIGH CPU CONSUMPTION FOR IDAA LOAD TABLE PROCESS. IMPROVE IDAA LOAD TO SUPPORT UNKNOWN RECORD LENGTH IN DB2 UNLOAD PHASE) reduces CPU-consuming BSAM process during the DB2 UNLOAD step when IDAA loads data from Db2 to an accelerator. This APAR is addressed for Db2 10 and 11 for z/OS. The improvement is included in Db2 12 base code.

CPU savings are relative to the number of rows in the table. During controlled testing, loading a 29.5 billion row table with an average record length of 131 bytes showed 25% CPU time savings when loading data from a zEC12 LPAR to a full rack Striper with 12 streams, and 30% CPU time savings to a full rack Mako.

10.5.2 Load and query performance comparison on Mako versus Striper

The Mako and Striper machines share a basic architecture: I/O capacity, memory, and the Field Programmable Gate Array (FPGA) The differences are that Mako has twice the CPU capacity and build-in hardware support for encryption of data at rest. Mako also has enhancements on the host and snippet processing units (SPU) network.

On our IBM Retail Data Warehouse benchmark workloads, little performance difference was observed in IDAA load between Mako and Striper. However, accelerated queries run 30% faster in average with Mako compared to Striper.

10.5.3 In-database transformation

Many users choose not to run Extract, Load, and Transformation (ELT) processes and analytic applications on Db2 for z/OS because of the cost or need for the speed. They would need to move the transaction data from Db2 to somewhere else. Moving the large volumes of data can be expensive and time consuming. Because of the cost or operational overhead, the data moving can be done periodically, such as at the end of the day or at the end of the week. The infrequent update of transactional data prevents the analytic applications run on real-time data that is available in Db2 for z/OS. With the IDAA solution, Db2 data can be loaded or incrementally updated to the accelerator conveniently without much delay.

In addition, eligible long-running SQLs are offloaded automatically to the accelerator with great query performance on the data that can be near real time. Therefore, it is understandable that people want to keep their ELT processes and analytic SQLs inside Db2 and the accelerator combined solution.

An SQL statement such as INSERT from SUBSELECT is most commonly used as a part of ELT processes. Unfortunately, so far only the select part can be considered for acceleration, and the accelerator returns all the rows back to Db2 to insert into a Db2 table. This process might not be performance-efficient or cost-effective with a typically large volume of the data.

Moreover, some ETL processes or tools create intermediate tables (for example, DGTTs or PBGs) to store the intermediate data and query against them within the same unit of work. Db2 and IDAA cannot accelerate these queries.

To improve performance and cost, a special mechanism is needed so that the accelerator can directly insert data into or select data from within the accelerator without moving data to or from Db2.

To address this need, Db2 introduces a new type of table: Accelerator-only table (AOT). When the user creates an AOT, instead of specifying the database and table space, they indicate in which accelerator to create it. Although Db2 keeps the table and its column information in the catalog and directory, the data is only in the accelerator, not in Db2. ELT processes can use AOT tables as intermediate tables. After the source tables are loaded on the accelerator, all of the following ELT processes can run on the accelerator, which is known as *accelerated in-database transformation*.

Before this feature was available, Db2 offloaded only select statements to an accelerator. Now, Db2 can offload DDL statements and insert, update, and delete statements if they are for or against an AOT table.

Insert with select by using AOT

During our performance testing, we prepared different categories of insert with subselect statements, and used different numbers of rows and columns and different data types.

Performance numbers were collected and compared between AOT, DGTT, and PBG being the insert targeting tables.

A dedicated z13 LPAR with six CPs and a full rack Mako machine were used to evaluate the performance of AOT. All of the tables that the queries reference were accelerated. For the DGTT and PBG cases, the accelerator returned data to Db2 to insert. For the AOT case, the accelerator inserted returning data directly into the AOT on the accelerator side.

The results are shown in Figure 10-1.

			Elapsed time comparison (unit is second)					CPU time comparison (unit is second)		
Remarks	#Rows	CORR	AOT	DGTT		PBG		AOT	DGTT	PBG
			CL2EL	CL2EL	DGTT/AOT	CL2EL	PBG/AOT	CL2CPU	CL2CPU	CL2CPU
small # rows for insert	5	RI101	102.00	103.83	1.02	103.07	1.01	0.01	0.08	0.02
	2	RI102	90.31	90.13	1.00	90.33	1.00	0.01	0.01	0.01
	175	RI103	286.21	286.33	1.00	284.89	1.00	0.01	0.01	0.01
	4,200	RI104	3.63	4.38	1.21	3.21	0.88	0.01	0.03	0.03
large # rows for insert, RI210 source is AOT	340,000	RI201	1.83	2.29	1.26	3.57	1.95	0.00	1.18	1.93
	3,400,000	RI202	11.57	21.65	1.87	29.85	2.58	0.00	11.71	16.64
	1,200,000	RI203	3.58	7.96	2.22	9.04	2.52	0.00	4.52	6.23
	12,000,000	RI204	13.21	71.89	5.44	76.12	5.76	0.00	47.58	53.96
	1,000,000	RI205	1.58	5.04	3.19	9.55	6.05	0.00	4.38	5.87
	10,000,000	RI206	3.48	46.90	13.46	63.16	18.13	0.00	44.00	48.26
	100,000,000	RI207	27.16	462.35	17.02	673.99	24.82	0.00	437.70	567.32
	1,000,000,000	RI208	290.58	canceled		14863.80	51.15	0.00	canceled	13879.28
	180,000,000	RI209	8.94	851.74	95.29	1291.38	144.47	0.00	801.85	1191.08
	180,000,000	RI210	5.15	843.07	163.80	1377.73	267.68	0.00	794.80	1186.29
different # columns - 16, 32, 48	1,000,000	RI301	1.56	5.04	3.23	9.57	6.13	0.00	4.43	5.93
	1,000,000	RI302	2.34	7.65	3.27	12.90	5.52	0.00	5.65	7.18
	1,000,000	RI303	2.76	10.72	3.88	15.38	5.57	0.00	6.89	8.67
different data types - INT, CHAR, DATE	1,000,000	RI401	0.88	4.12	4.68	7.98	9.07	0.00	3.47	5.29
	1,000,000	RI402	1.22	4.54	3.72	9.24	7.57	0.00	3.91	5.41
	1,000,000	RI403	0.90	4.63	5.13	8.41	9.32	0.00	4.16	5.96
		Sum	858.90			18943.18	22.06	0.07		16995.36

Figure 10-1 Insert with subselect with AOT, DGTT, PBG as insert target table

Queries RI101 ~ RI104 are complex selects with a few returned rows for insert. Queries RI101 ~ RI104 contain different subselect statement. Little performance difference was observed among AOT, DGTT, and PBG as most time was spent at query in IDAA.

Queries RI201 ~ RI210 are simple selects with many rows returned for insert. The advantage of using AOT to insert directly is significant (up to 163X between AOT and DGTT, and 267X between AOT and PBG). We exhausted the work file space in Db2 subsystem in RI208 for DGTT scenario and did not continue the query.

Queries RI301 ~ RI303 are simple selects with 1 million returned rows of 16, 32, and 48 columns respectively. Again, AOT shows the benefit.

Queries RI401 ~ RI403 are simple selects with 1 million returned rows of the same number of columns of integer, char or varchar, and date columns, respectively.

When AOTs are used as the target of insert from select, the select and insert parts were run on the accelerator. Almost no CPU cost was left on Db2 side.

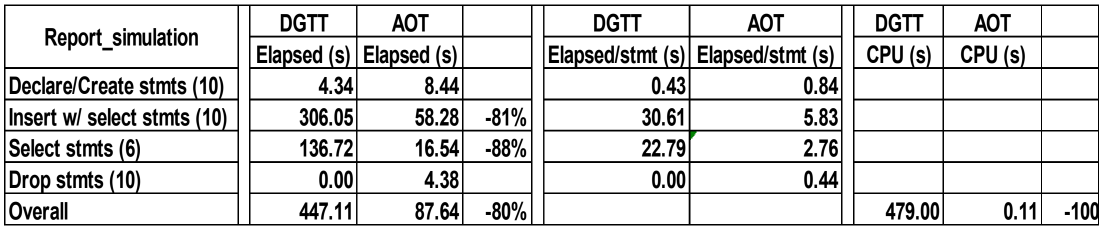
These experiments demonstrate the benefit of the use of AOT when more than 300,000 rows are processed.

AOT versus DGTT as intermediate tables

To simulate actual ELT process, we simplified a customer's case and evaluated the overall elapsed time and CPU time saving by using AOT versus DGTT as the intermediate tables. We completed the following steps to simulate report generation by JDBC type 4 application:

1. Create 10 intermediate tables.
2. Populate the intermediate tables by using 10 insert with select statements (all the tables in the select part are accelerated and loaded).
3. Run six queries that reference the 10 intermediate tables.
4. Drop the 10 intermediate tables.

The performance comparisons of the use AOT versus DGTT as the intermediate tables is shown in Figure 10-2.



Report_simulation	DGTT	AOT		DGTT	AOT	DGTT	AOT	
	Elapsed (s)	Elapsed (s)		Elapsed/stmt (s)	Elapsed/stmt (s)	CPU (s)	CPU (s)	
Declare/Create stmts (10)	4.34	8.44		0.43	0.84			
Insert w/ select stmts (10)	306.05	58.28	-81%	30.61	5.83			
Select stmts (6)	136.72	16.54	-88%	22.79	2.76			
Drop stmts (10)	0.00	4.38		0.00	0.44			
Overall	447.11	87.64	-80%			479.00	0.11	-100

Figure 10-2 Performance comparison for report simulation using DGTT and AOT as intermediate table

Creating AOTs takes longer than declaring DGTTs, and dropping AOTs also takes more time. However, the insert and select statements run much faster with AOTs than DGTTs and the overall report process finishes 80% faster with significantly less CPU time by using AOTs than DGTTs.

Concurrent insert from select

We also experimented with inserting 10 million rows into an AOT table from a select (per thread) with concurrent threads by using a JDBC type 4 application. The elapsed time per occurrence for insert with select under different concurrency for case A and B is shown in Figure 10-3.

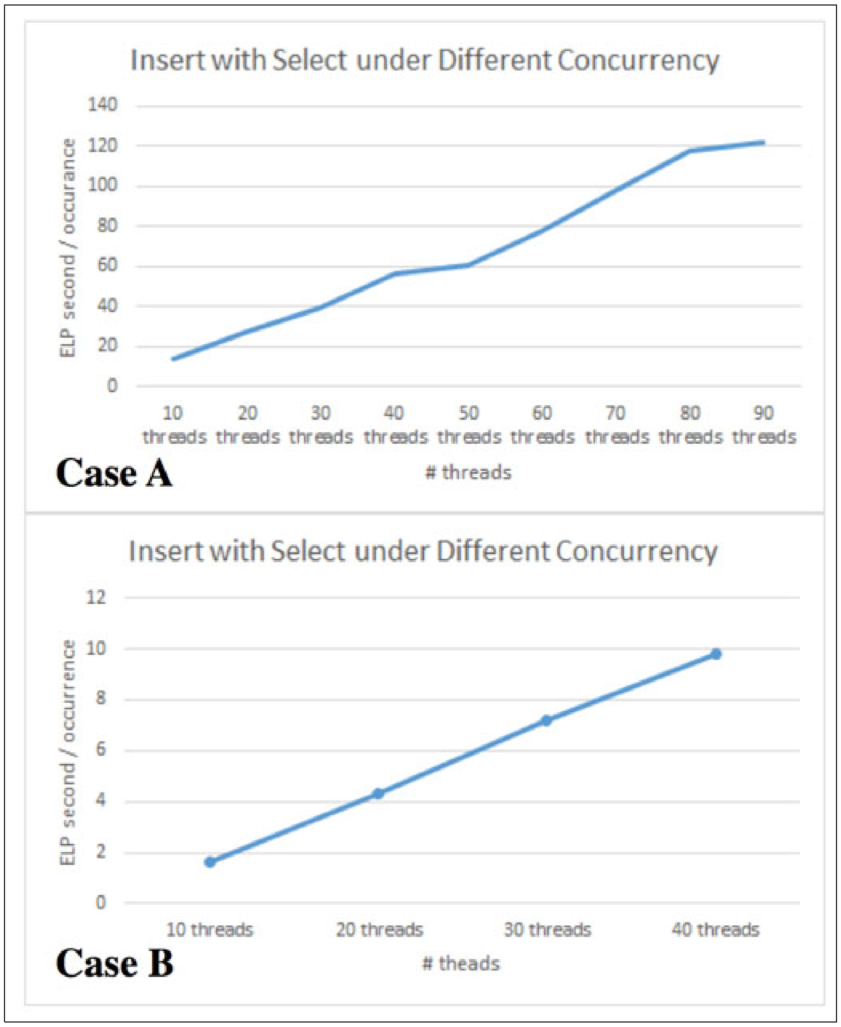


Figure 10-3 Insert with subselect under different concurrency

In the measurement, each thread has its own AOT target table but the same select statement (same source tables with joins). The result shows the elapsed time per insert lines up with the number of concurrent threads, which means our concurrent inserts do not scale up with the number of concurrent threads (marked as Case A in Figure 10-3).

To verify whether the contention is from the source table, we ran other test (marked as Case B Figure 10-3). In Case B separate source and target (AOT) tables are used per thread.

The liner increase in the elapsed time occurred as we increase the number of threads because of the fundamental design strategy of Netezza. The NPS design goal is to apply all the system resources to a single process to accelerate the performance. Therefore, each thread can run fast, yet the elapsed time with the concurrent process tends to suffer as a tradeoff.

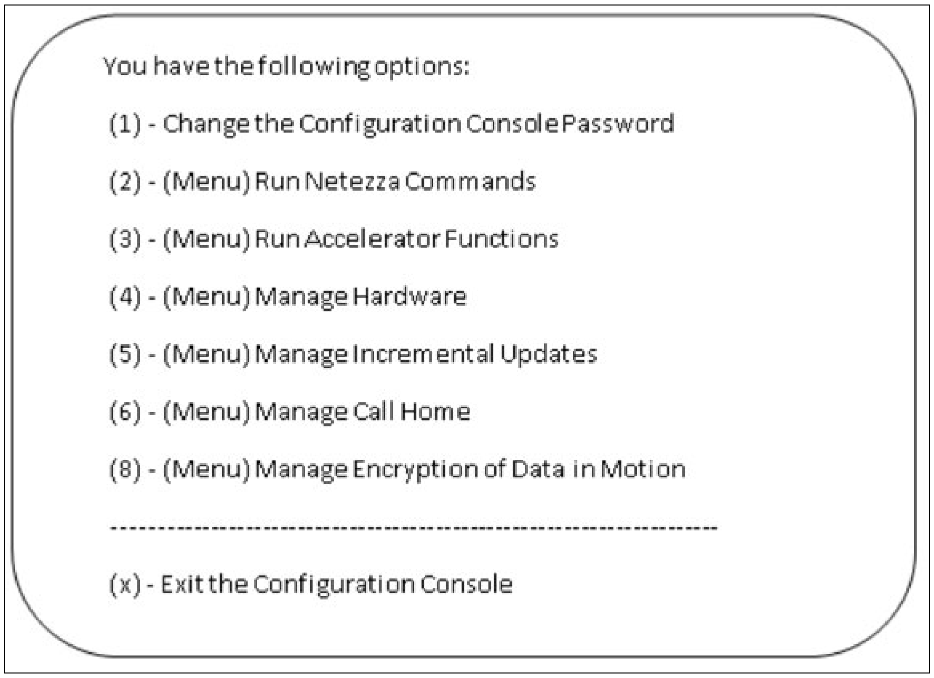
10.5.4 Apply incremental updates in parallel

Incremental updates capability can quickly replicate the changes in the Db2 to the accelerated tables at IDAA. However, when many tables are in the subscription, replication latency can suffer. In such a case, customers can use the parallel apply feature so that incremental update changes can be applied in parallel. This feature is not enabled as a default as some usage consideration are important, such as referential integrity usage. When enabled, the default number of parallel threads is four, but this limit can be changed with guidance from IBM.

Enabling parallel apply

Complete the following steps to enable parallel apply:

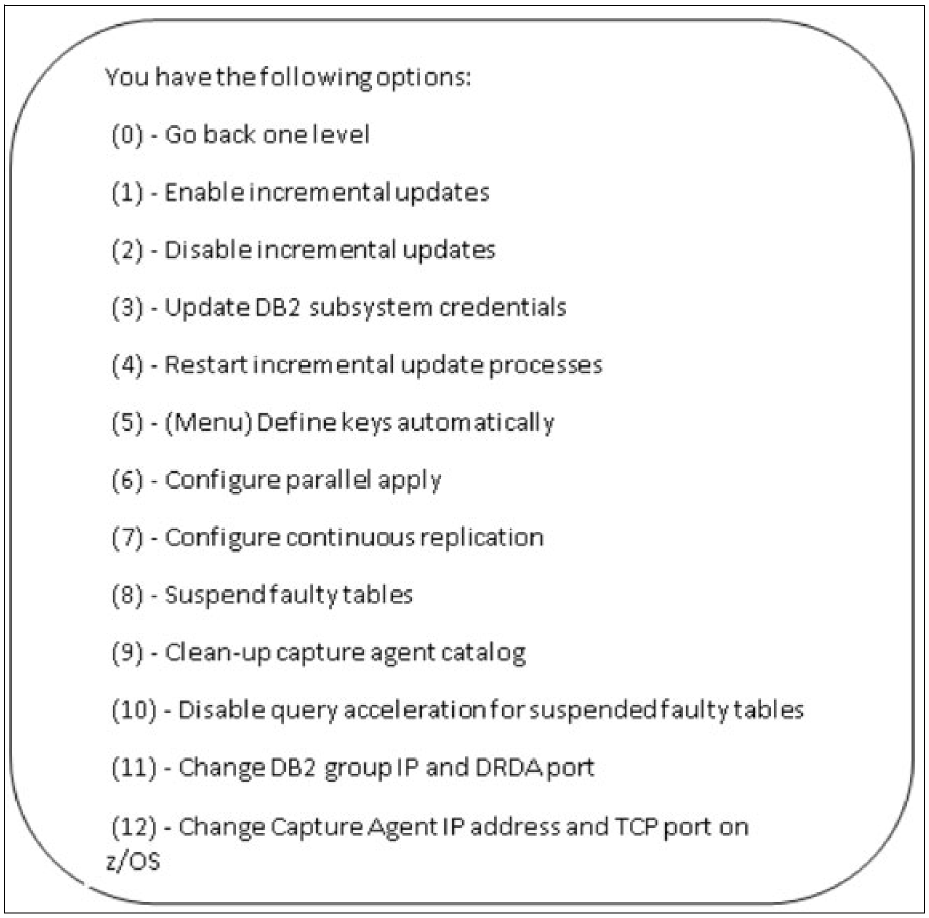
1. Log on to the IBM Db2 Analytics Accelerator Console.
2. Press Enter to display the Configuration Console window that is shown in Figure 10-4.



```
You have the following options:

(1) - Change the Configuration Console Password

(2) - (Menu) Run Netezza Commands

(3) - (Menu) Run Accelerator Functions

(4) - (Menu) Manage Hardware

(5) - (Menu) Manage Incremental Updates

(6) - (Menu) Manage Call Home

(8) - (Menu) Manage Encryption of Data in Motion

--------------------------------------------------------------------

(x) - Exit the Configuration Console
```

Figure 10-4 Configuration Console window

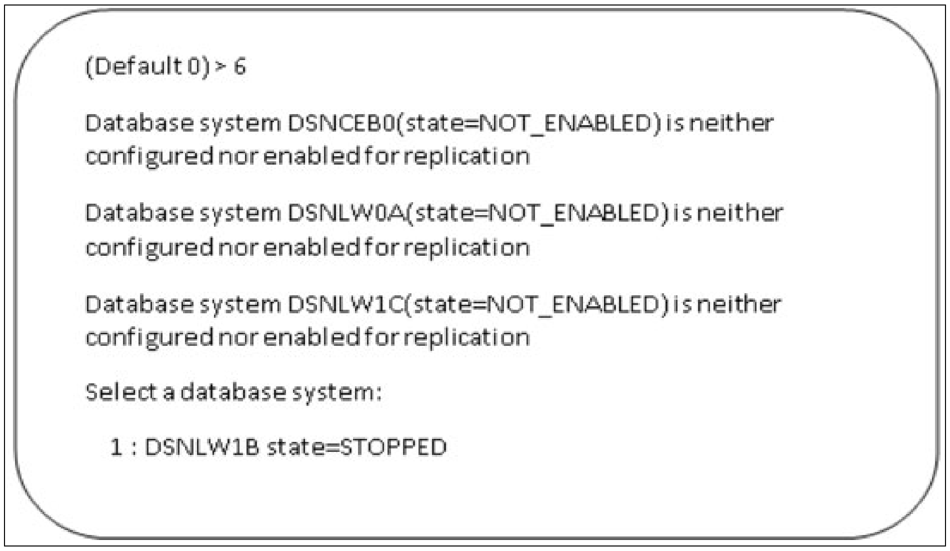
3. Enter 5 and press Enter to display the submenu that is shown in Figure 10-5.



```
You have the following options:
 (0) - Go back one level
 (1) - Enable incremental updates
 (2) - Disable incremental updates
 (3) - Update DB2 subsystem credentials
 (4) - Restart incremental update processes
 (5) - (Menu) Define keys automatically
 (6) - Configure parallel apply
 (7) - Configure continuous replication
 (8) - Suspend faulty tables
 (9) - Clean-up capture agent catalog
 (10) - Disable query acceleration for suspended faulty tables
 (11) - Change DB2 group IP and DRDA port
 (12) - Change Capture Agent IP address and TCP port on
z/OS
```

Figure 10-5 Configuration Console submenu

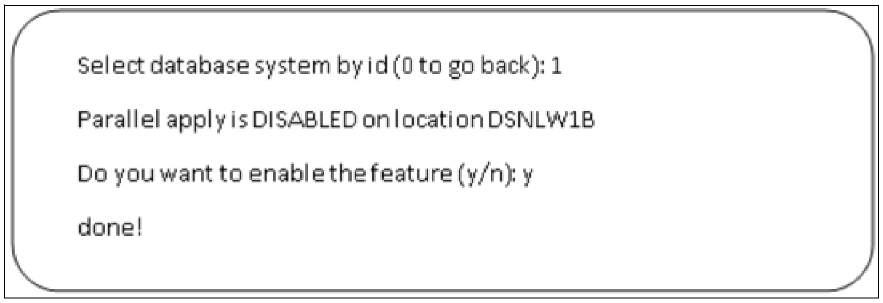
4. Enter 6 and press Enter. A window similar to the window that is shown in Figure 10-6 opens. Db2 subsystems that cannot be selected because they are not enabled for incremental updates are listed. Under “Select a database system”, you see the Db2 subsystems that you can select.



```
(Default 0) > 6

Database system DSNCEB0(state=NOT_ENABLED) is neither
configured nor enabled for replication

Database system DSNLW0A(state=NOT_ENABLED) is neither
configured nor enabled for replication

Database system DSNLW1C(state=NOT_ENABLED) is neither
configured nor enabled for replication

Select a database system:

  1 : DSNLW1B state=STOPPED
```

Figure 10-6 Available Db2 subsystems

5. Enter the number of the Db2 subsystem that you want to enable and press Enter. The message that is shown in Figure 10-7 is displayed.



```
Select database system by id (0 to go back): 1

Parallel apply is DISABLED on location DSNLW1B

Do you want to enable the feature (y/n): y

done!
```

Figure 10-7 Enabling the feature prompt

Performance measurement

The performance measurement used as many as 500 IBM Retail Data Warehouse LINEITEM-like tables to simulate the customer’s use case. Each table includes approximately 24 million rows and is in a partition by range table space. Concurrent insert and update transactions were run against these 500 tables. The measurement is to compare the replication throughput of a single apply to multiple apply with a default of four parallel apply threads.

Performance results

The measurements showed up to 98% replication throughput improvement comparing parallel apply by using four threads to apply by using a single thread.

Parallel Apply feature consideration

Our measurements demonstrate that the Parallel Apply feature is effective to reduce the replication latency when many tables must be replicated.

10.5.5 Automatic generation of distribution keys and organizing keys for incremental update enabled tables

The use of the appropriate distribution keys and organizing keys can improve the performance of incremental update processes considerably. IDAA V4 provides the automatic selections of distribution and organizing keys.

Distribution keys and organizing keys

Distribution and organizing keys are optional to improve the performance of accelerator.

Distribution keys

A user can choose between two methods to distribute table rows to the worker nodes of their accelerator: Random distribution (default) and distribution by using a distribution key. When a distribution key is specified, the accelerator uses a hash function on the key columns to determine the data slice or worker node that receives a table row. Rows with equal hash values are distributed to the same worker node.

By default, random distribution is used to distribute table rows to worker nodes for query processing. That is, all tables are evenly distributed among the worker nodes.

Organizing keys

Organizing key also affect the query response time. When choosing an organizing key, columns are selected by grouping the rows of an accelerator-shadow table within the data slices on the worker nodes. This grouping creates grouped segments or blocks of rows with equal or nearby values in the columns that are selected as organizing keys.

If an incoming SQL query references one of the organizing key columns in a range or equality predicate, the query can run much faster because entire blocks can be skipped. It is not necessary to scan the entire accelerator-shadow table. Therefore, the time that is needed for disk output operations that are related to the query is drastically reduced.

Organizing keys work with zone maps, which allow the query engine to identify the disk extents (addressable storage blocks on the disk) that contain the clusters of rows that are relevant to a query.

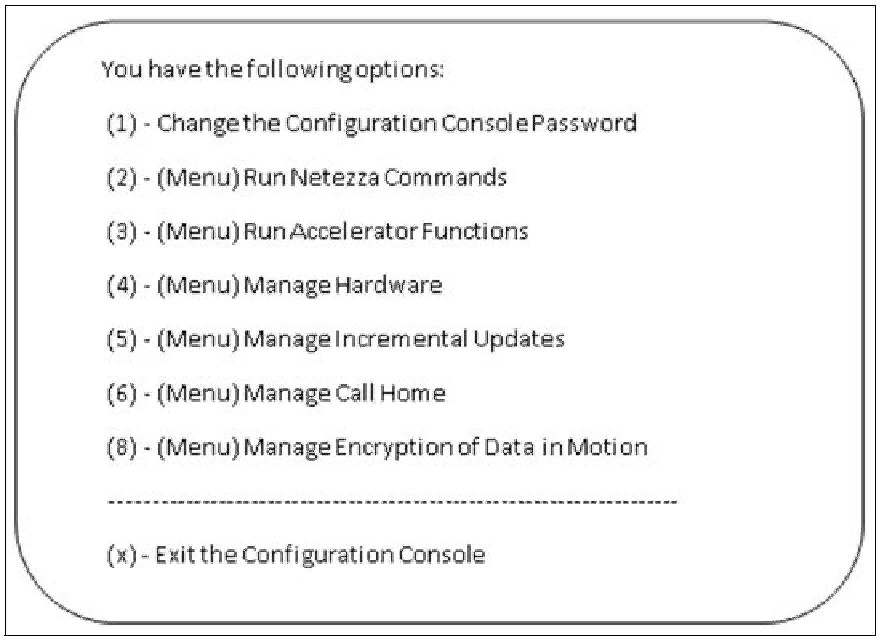
Defining distribution keys and organizing keys for incremental update enabled tables

Customers can configure the product so that a distribution key, organizing key, or both are defined automatically for tables that are enabled for replication if there is a unique index that is defined on the tables that are on Db2 for z/OS. Keys are automatically defined only when the table is being enabled for replication. A replication enabled table with keys is not affected by the setting. The IDAA server needs the most up-to-date Db2 catalog statistics to determine which columns to define as the keys, so users must run Db2 RUNSTATS on Db2 tables that are to be added into the accelerator and enabled for incremental updates.

Enabling defining auto keys

Complete the following steps to enable defining auto keys:

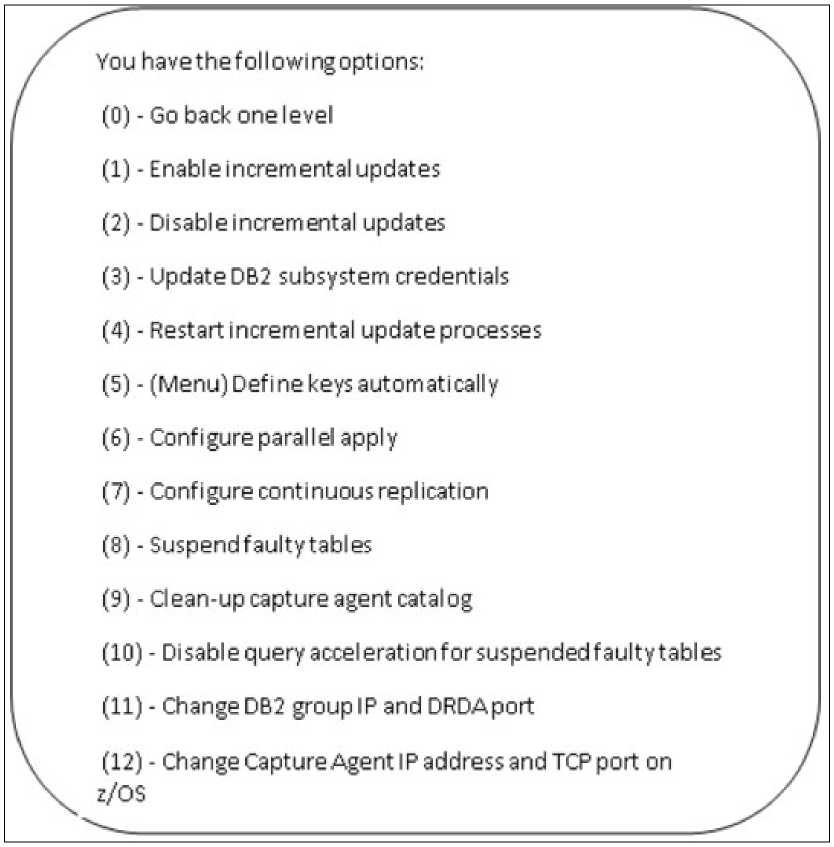
1. Log on to IBM Db2 Analytics.
2. Press Enter to display the Configuration Console window, as shown in Figure 10-8.



```
You have the following options:
 (1) - Change the Configuration Console Password
 (2) - (Menu) Run Netezza Commands
 (3) - (Menu) Run Accelerator Functions
 (4) - (Menu) Manage Hardware
 (5) - (Menu) Manage Incremental Updates
 (6) - (Menu) Manage Call Home
 (8) - (Menu) Manage Encryption of Data in Motion
 -------------------------------------------------------------------
 (x) - Exit the Configuration Console
```

Figure 10-8 Configuration Console window

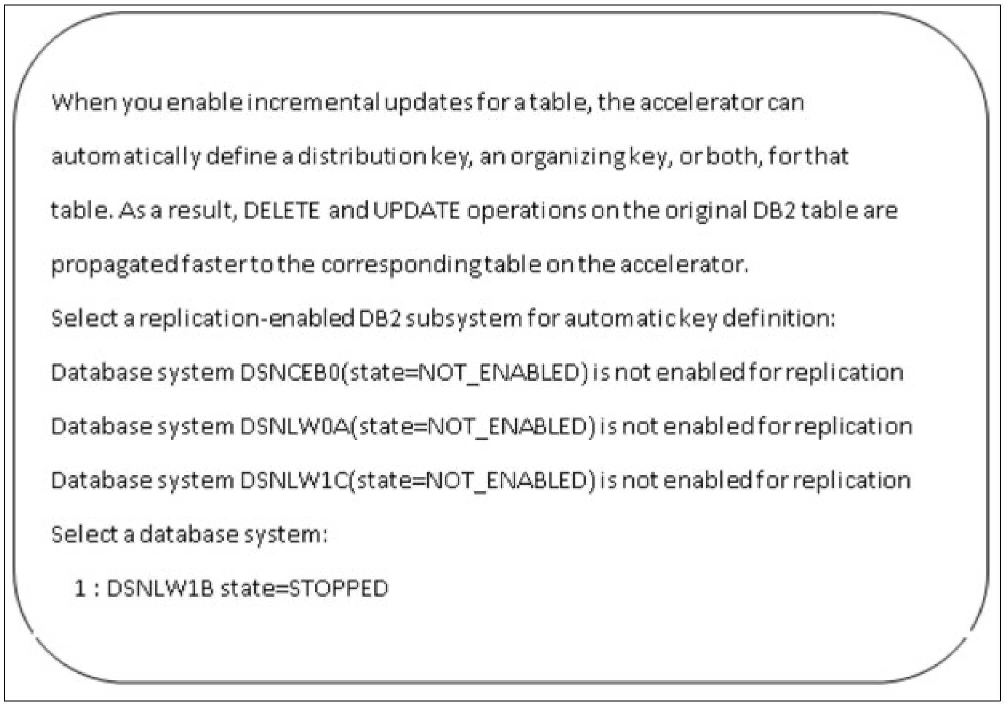
3. Enter 5 and press Enter to display the submenu that is shown in Figure 10-9 on page 312.



```
You have the following options:
 (0) - Go back one level
 (1) - Enable incremental updates
 (2) - Disable incremental updates
 (3) - Update DB2 subsystem credentials
 (4) - Restart incremental update processes
 (5) - (Menu) Define keys automatically
 (6) - Configure parallel apply
 (7) - Configure continuous replication
 (8) - Suspend faulty tables
 (9) - Clean-up capture agent catalog
 (10) - Disable query acceleration for suspended faulty tables
 (11) - Change DB2 group IP and DRDA port
 (12) - Change Capture Agent IP address and TCP port on
z/OS
```

Figure 10-9 Configuration Console submenu

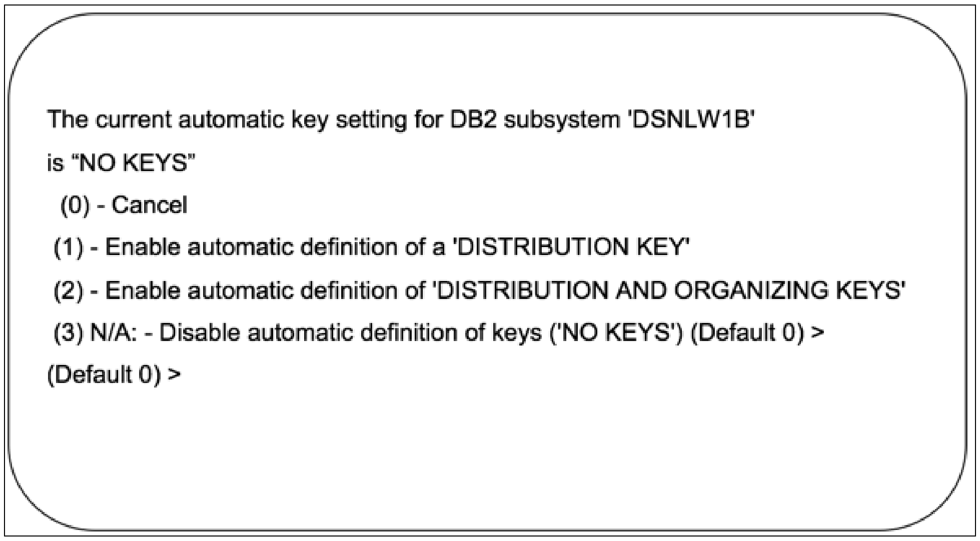
4. Enter 5 to display the menu that is shown in Figure 10-10.



```
When you enable incremental updates for a table, the accelerator can
automatically define a distribution key, an organizing key, or both, for that
table. As a result, DELETE and UPDATE operations on the original DB2 table are
propagated faster to the corresponding table on the accelerator.
Select a replication-enabled DB2 subsystem for automatic key definition:
Database system DSNCEB0(state=NOT_ENABLED) is not enabled for replication
Database system DSNLW0A(state=NOT_ENABLED) is not enabled for replication
Database system DSNLW1C(state=NOT_ENABLED) is not enabled for replication
Select a database system:
  1 : DSNLW1B state=STOPPED
```

Figure 10-10 Selecting a database system

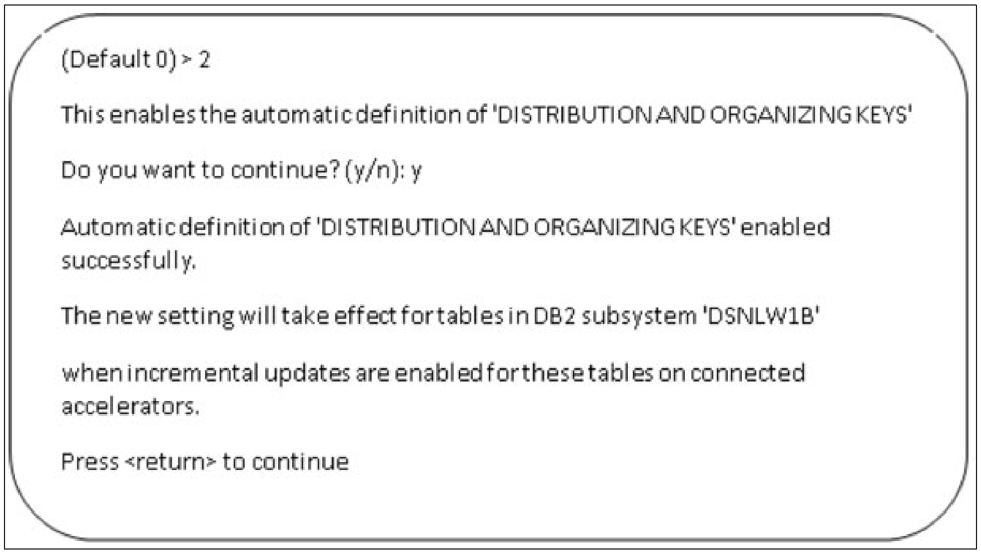
5. Enter the number of the Db2 subsystem that you want to enable and press Enter. The menu that is shown in Figure 10-11 is displayed.



```
The current automatic key setting for DB2 subsystem 'DSNLW1B'
is "NO KEYS"
 (0) - Cancel
 (1) - Enable automatic definition of a 'DISTRIBUTION KEY'
 (2) - Enable automatic definition of 'DISTRIBUTION AND ORGANIZING KEYS'
 (3) N/A: - Disable automatic definition of keys ('NO KEYS') (Default 0) >
(Default 0) >
```

Figure 10-11 Automatic settings submenu

6. Choose Option 2 to enable the distribution and organizing keys; then, enter y to confirm, as shown in Figure 10-12.



```
(Default 0) > 2
This enables the automatic definition of 'DISTRIBUTION AND ORGANIZING KEYS'
Do you want to continue? (y/n): y
Automatic definition of 'DISTRIBUTION AND ORGANIZING KEYS' enabled
successfully.
The new setting will take effect for tables in DB2 subsystem 'DSNLW1B'
when incremental updates are enabled for these tables on connected
accelerators.
Press <return> to continue
```

Figure 10-12 Enabling distribution and organizing keys

Performance measurement

By using insert and update transactions on 500 23 GB and 10 1 TB partitioned tables, we compared the replication throughput and replication latency of enabling the auto key for a distribution or organizing key, or both types of keys, versus no keys. All of the measurements were done with parallel apply enabled.

Performance results

The measurements showed that defining distribution and organizing keys automatically improved replication latency and throughput.

The positive performance effect on delete operations during incremental updates when defining both types of keys is listed in Table 10-1. This improvement is achieved because of scanning fewer data pages.

Table 10-1 Netezza delete performance result for 10 1-TB tables

Testing features	Execution time (sec)	Average pages per data slice	Number of slices processed	Number of pages scanned/slice
No keys	9.023	25,5	240	947
Distribution keys	8.443	28,6		947
Both keys	0.090		2	2

An approximate 12% replication throughput improvement was observed when the distribution and organizing key were defined compared to having no keys defined, as listed in Table 10-2.

Table 10-2 Replication throughput and latency result

Testing features	Maximum latency (min:sec)	Average throughput (1K rows/second)	Throughput improvement (%)	Average apply time (min:sec)
No Keys	06:10	25,5		04:28
Auto Keys (both)	01:57	28,6	12%	01:41

Considerations for optimal key definitions

RUNSTATS information must be up-to-date for IDAA to detect the correct candidate for distribution keys and organizing keys.

10.6 IDAA version 4 PTF-6

In this section, we describe IDAA load and query performance with z13 and continuous operation of incremental updates.

10.6.1 IDAA load and query performance with z13 versus zEC12

By upgrading zEC12 to z13, we observed an approximately 38% CPU time reduction loading the data into IDAA because of the general processor improvement with z13 against compressed tables. In addition, 26% Db2 CPU time reduction in overall accelerated queries was observed when our IBM Retail Data Warehouse benchmark workloads were used.

10.6.2 Continuous operation of incremental updates

Before IDAA version 4 PTF-6, replication for all tables in the subscription is stopped for a short period before loading any of the tables in the subscription starts, and after the table load stops. Although this period is short, the effect on replication throughput and latency becomes significant when many tables are defined in the subscription and frequent concurrent load or reload activities occur.

Continuous operation

With the continuous operation feature, incremental updates can continue even when there is concurrent load or reload activity without stopping. To make the most of this feature, IDAA load table applications must make the following changes when calling the SYSPROC.ACCEL_LOAD_TABLES stored procedure:

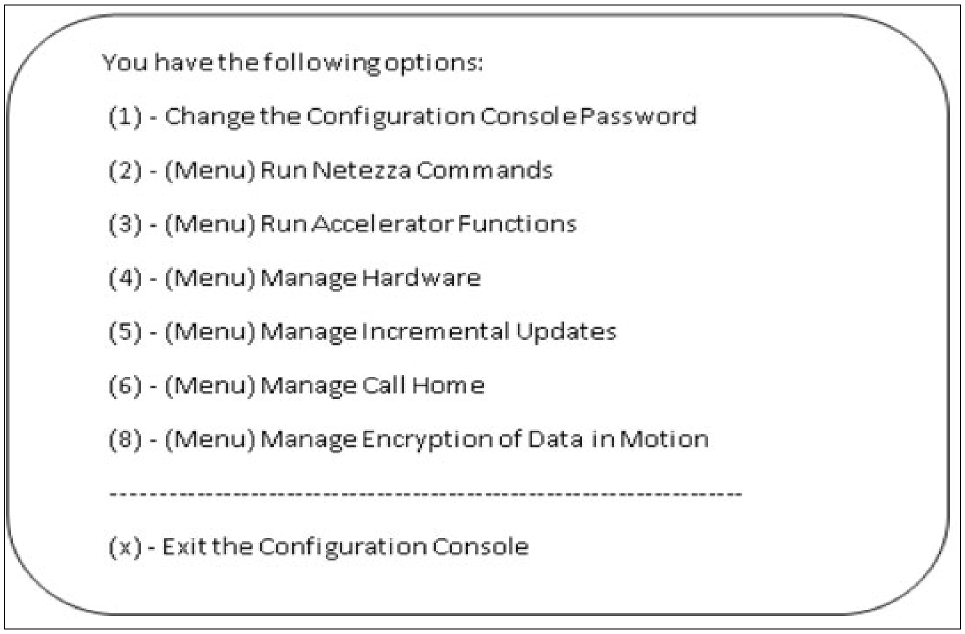
- Use Lock-mode "ROW" for reload/load.
- Use forceFullReload = true to avoid incorrect result for reload.

This performance enhancement feature is not enabled by default. The procedure that is used to enable it is described next.

Enabling continuous options

Complete the following steps to enable continuous options:

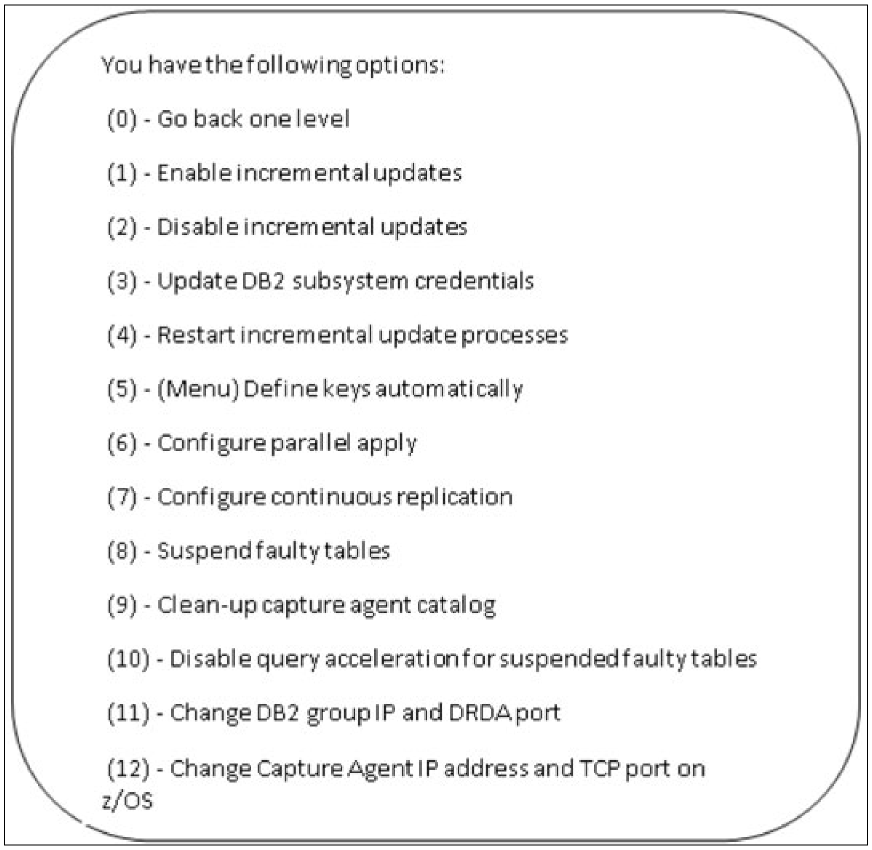
1. Log on to the IBM Db2 Analytics Accelerator Console.
2. Press Enter to display the Configuration Console window that is shown in Figure 10-13.



```
You have the following options:
 (1) - Change the Configuration Console Password
 (2) - (Menu) Run Netezza Commands
 (3) - (Menu) Run Accelerator Functions
 (4) - (Menu) Manage Hardware
 (5) - (Menu) Manage Incremental Updates
 (6) - (Menu) Manage Call Home
 (8) - (Menu) Manage Encryption of Data in Motion
 ------------------------------------------------------------------
 (x) - Exit the Configuration Console
```

Figure 10-13 Configuration Console window

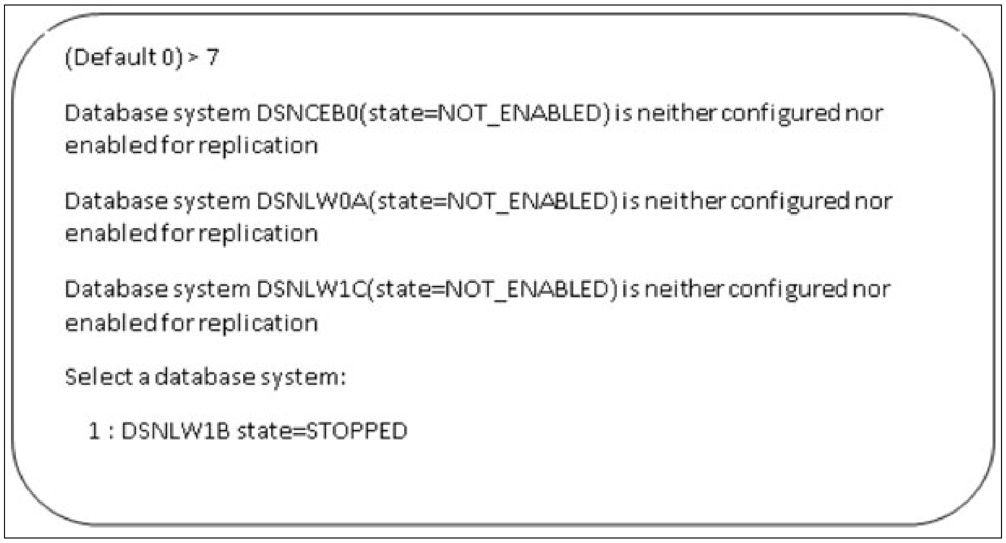
3. Enter 5 and press Enter to display the submenu that is shown in Figure 10-14 on page 316.



```
You have the following options:
 (0) - Go back one level
 (1) - Enable incremental updates
 (2) - Disable incremental updates
 (3) - Update DB2 subsystem credentials
 (4) - Restart incremental update processes
 (5) - (Menu) Define keys automatically
 (6) - Configure parallel apply
 (7) - Configure continuous replication
 (8) - Suspend faulty tables
 (9) - Clean-up capture agent catalog
 (10) - Disable query acceleration for suspended faulty tables
 (11) - Change DB2 group IP and DRDA port
 (12) - Change Capture Agent IP address and TCP port on
z/OS
```

Figure 10-14 Options window

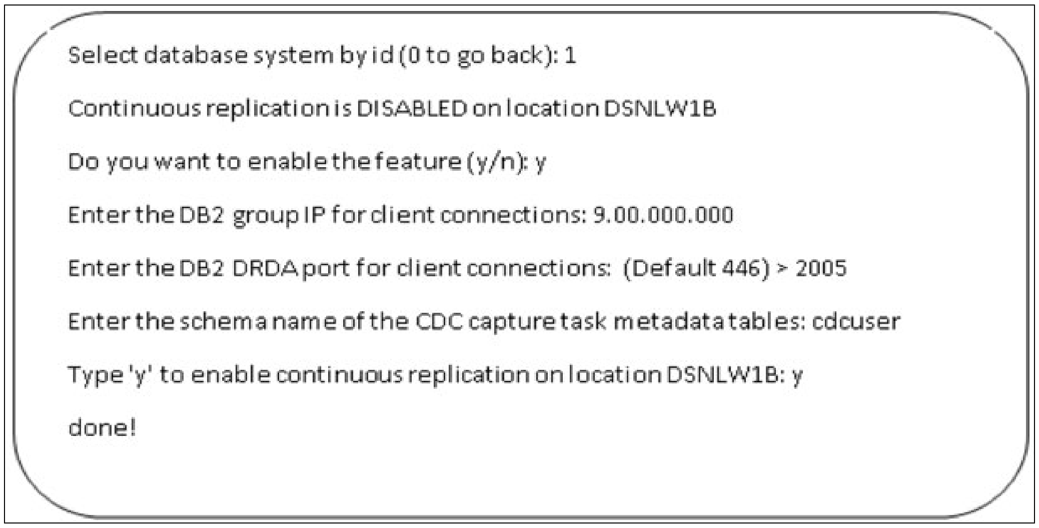
4. Enter 7 and press Enter. A window that is similar to the window that is shown in Figure 10-15 is displayed. Db2 subsystems that cannot be selected because they were not enabled for incremental updates are listed. The Db2 subsystems that you can select are listed in the Select a database system section.



```
(Default 0) > 7
Database system DSNCEB0(state=NOT_ENABLED) is neither configured nor
enabled for replication
Database system DSNLW0A(state=NOT_ENABLED) is neither configured nor
enabled for replication
Database system DSNLW1C(state=NOT_ENABLED) is neither configured nor
enabled for replication
Select a database system:
  1 : DSNLW1B state=STOPPED
```

Figure 10-15 Selecting a database system

5. Enter the number of the Db2 subsystem that you want to enable and press Enter. Then, follow the steps that are listed in the menu that is shown in Figure 10-16 to enable the feature.



```
Select database system by id (0 to go back): 1
Continuous replication is DISABLED on location DSNLW1B
Do you want to enable the feature (y/n): y
Enter the DB2 group IP for client connections: 9.00.000.000
Enter the DB2 DRDA port for client connections:  (Default 446) > 2005
Enter the schema name of the CDC capture task metadata tables: cdcuser
Type 'y' to enable continuous replication on location DSNLW1B: y
done!
```

Figure 10-16 Enabling the feature

Performance measurement

For the performance evaluation of this feature, we used 200 of tables from IBM Retail Data Warehouse workload each containing 24 million rows. Concurrent insert and update transactions going against the 200 tables were available while half of those 200 tables were being reloaded. The following measurements were conducted:

- Continuous operation disabled, reload with LOCK-MODE tableset as the default behavior
- Continuous operation enabled, reload with LOCK-MODE row

Performance results

The measurements showed up to 37% replication throughput improvement compared to without continuous operation enabled. The Continuous Operation feature is effective when many tables must be loaded or reloaded constantly while replication is ongoing.

10.7 IDAA version 5 PTF-1

In this section, we describe the IDAA enhancements in IDAA version 5 PTF-1 to increase load throughput and the support for encryption of data in motion.

10.7.1 Load throughput increase from lifting TCP send buffer size restriction

When loading a partitioned table from Db2 to the accelerator, the load might take longer to complete if AQT_MAX_UNLOAD_IN_PARALLEL environment variable is set to a higher number (for example, more than 30). This issue occurs because of a restriction in an algorithm to calculate the TCP send buffer size. After lifting this restriction, the throughput of an initial load of a table (250 partitions and 3.88 TB data size) increased from 2.63 TB/hour to 3.94 TB/hour (almost 50%) from Db2 running on a z13 LPAR to IDAA by using a Mako with AQT_MAX_UNLOAD_IN_PARALLEL=12.

The effect of lifting this restriction is that the user can receive TCP error number 1122 when AQT_MAX_UNLOAD_IN_PARALLEL is set to a number that is too high (50 in our case). Reduce the number of AQT_MAX_UNLOAD_IN_PARALLEL or reduce the number of concurrent threads for load when TCP error number 1122 is encountered.

10.7.2 Encrypting data in motion by using IPSec

Earlier versions of IBM Db2 Analytics Accelerator for z/OS were delivered without a network encryption function. Although this issue did not pose a security risk when the setup recommendations were followed (dedicated private network between the mainframe computer or LPAR and the accelerator), it soon became obvious that in many situations, this type of setup did not align well with network infrastructures. Therefore, a secure way of data traffic had to be provided for customers who want to route sensitive data, such as patient data, credit-card transactions, or social security numbers through their corporate intranet.

Furthermore, the security standards of many organizations demand that any sensitive data that is sent across a network must be encrypted. Finally, encryption capabilities were added to the product.

To protect data security, IDAA provides functionality to encrypt data in motion by using IPSec in the network layer. However, performance is affected when encryption of data in motion is enabled.

Setting up encryption of data in motion

To set up IPSec, we consulted *IBM z/OS V2R1 Communications Server TCP/IP Implementation Volume 2*, SG24-8097. Because it uses the IKEV2 protocol in the authentication phase, the z/OS communication server must be configured with Network Security Services (NSS). Users must set up and configure the TCP/IP stack in Communication Server with Internet Key Exchange Daemon (IKED), Network Security Services Daemon (NSSD), policy agent, and Traffic Regulation Management Daemon (TRMD).

The accelerator side uses the IPSec “netkey” implementation that is provided by the Redhat Enterprise Linux 6 kernel with the IKE daemon “pluto” that is provided by libreswan 3.13. The standard PDA distribution does not include the libreswan IPSec implementation. Therefore, users require IBM service to deploy RHEL 6.6 and Host management version 5.1.4.0 or higher beforehand.

In our testing environment, as shown in Figure 10-17 on page 319, the following set-up was done on the z/OS side (a z13 LPAR):

1. Enable PAGENT policies for TTLS and IPSec for all the IP addresses between z/OS and accelerator.
2. Install the SYSLOGD daemon and set the logging level to the minimum (logging for errors only) to reduce CPU usage.
3. Install the TRMD daemon.
4. Install the NSSD and IKED daemons.
5. Create a TLS certificate for the communication between NSSD and IKED.
6. Create an NSS certificate for the communication between z/OS NSSD and accelerator.
7. Create a certificate for the accelerator and transfer it to the accelerator.

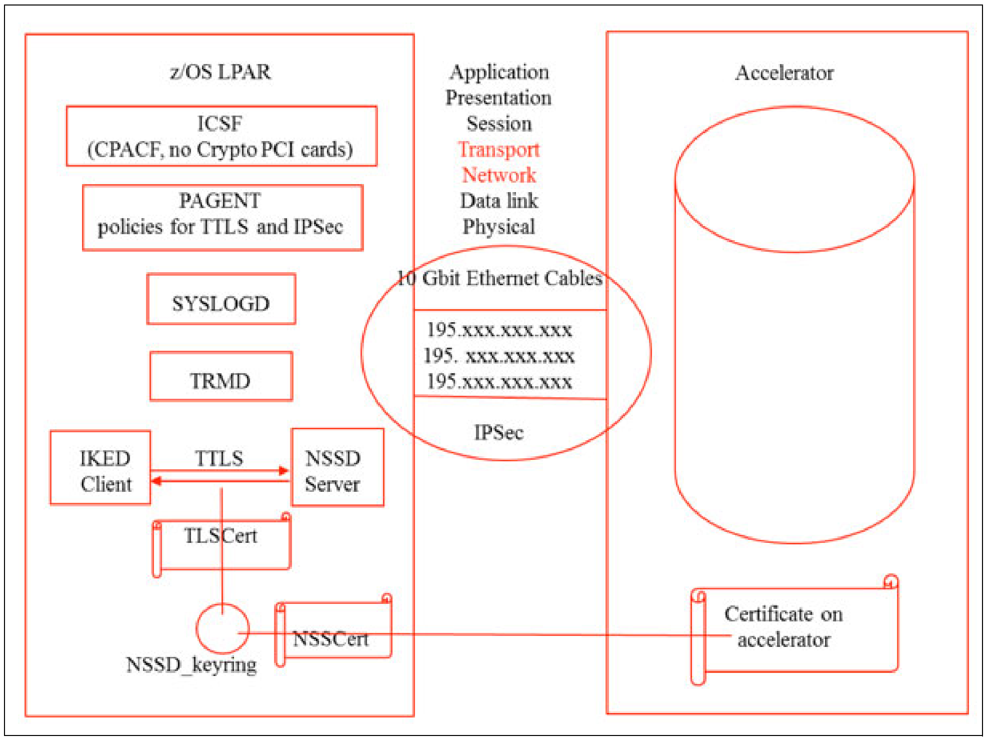


Figure 10-17 IPSec setup in our test environment

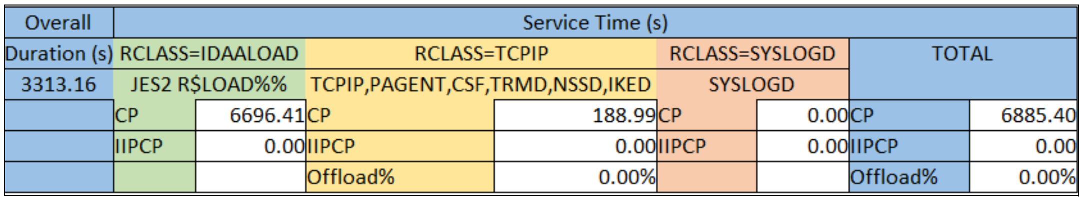
For z/OS Cryptographic Services Integrated Cryptographic Service Facility System (ICSF), our LPAR has no crypto CPI cards. Instead, we use CP Assist for Cryptographic Functions (CPACF). Although CPACF instructions provide improved performance, it has a CPU cost. However, most of the CPU is zIIP eligible (our LPAR does not have a zIIP engine installed).

8. For our convenience when using the RMF reports, we created the following report classes:

   – (RCLASS) - IDAALOAD for the load process
   – IDAAQRY for queries
   – TCP/IP for TCP/IP (including cryptographic service)
   – PAGENT, CSF, TRMD, NSSD, and IKED
   – SYSLOGD for the SYSLOGD daemon

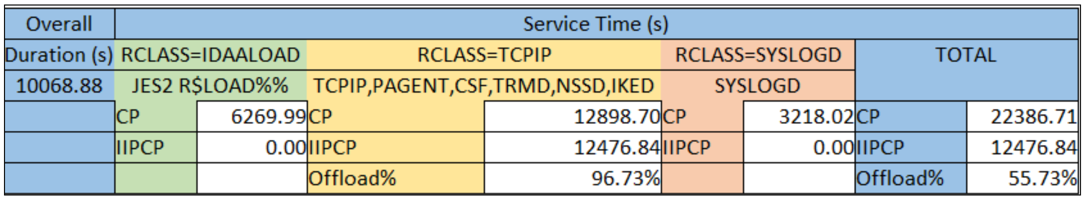
Load performance with and without encryption

The load performance comparison with no encryption of data in motion, compared to the use of encryption is shown in Figure 10-18 and Figure 10-19 on page 320. The load data size is 3.88 TB for 29.5 billion rows.



Overall	Service Time (s)							
Duration (s)	RCLASS=IDAALOAD		RCLASS=TCPIP		RCLASS=SYSLOGD		TOTAL	
3313.16	JES2 R$LOAD%%		TCPIP,PAGENT,CSF,TRMD,NSSD,IKED		SYSLOGD			
	CP	6696.41	CP	188.99	CP	0.00	CP	6885.40
	IIPCP	0.00	IIPCP	0.00	IIPCP	0.00	IIPCP	0.00
			Offload%	0.00%			Offload%	0.00%

Figure 10-18 Service time of load with no encryption of data in motion



Overall	Service Time (s)							
Duration (s)	RCLASS=IDAALOAD		RCLASS=TCPIP		RCLASS=SYSLOGD		TOTAL	
10068.88	JES2 R$LOAD%%		TCPIP,PAGENT,CSF,TRMD,NSSD,IKED		SYSLOGD			
	CP	6269.99	CP	12898.70	CP	3218.02	CP	22386.71
	IIPCP	0.00	IIPCP	12476.84	IIPCP	0.00	IIPCP	12476.84
			Offload%	96.73%			Offload%	55.73%

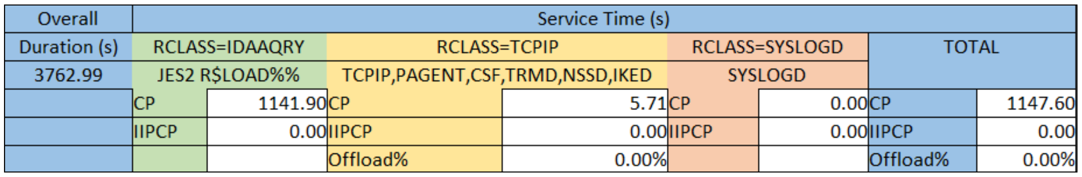
Figure 10-19 Service time of load with encryption of data in motion

Regarding elapsed time and the use of encryption of data in motion, the load process runs three times as long than without (3313 seconds versus 10068 seconds).

Regarding CPU time with encryption of data in motion, the load process uses 3.25 times more CPU than without, with 15501 seconds extra CPU from SYSLOGD, cryptographic service, and other daemons. Approximately 80% of these 15501 extra CPU seconds is zIIP-eligible.

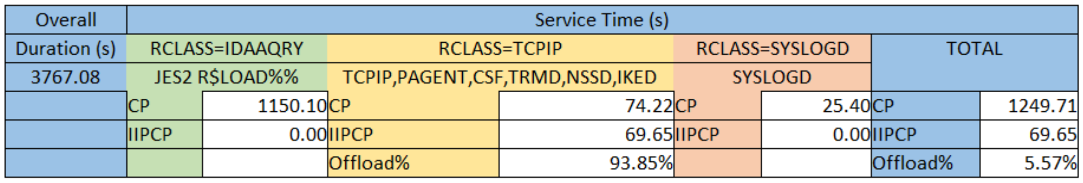
Queries performance with and without encryption

Figure 10-20 and Figure 10-21 show the query performance comparison between no encryption and encryption of data in motion. A total of 28 queries (21 IBM Retail Data Warehouse benchmark queries and 7 extra queries) fetched 170 million returned rows in this evaluation.



Overall	Service Time (s)							
Duration (s)	RCLASS=IDAAQRY		RCLASS=TCPIP		RCLASS=SYSLOGD		TOTAL	
3762.99	JES2 R$LOAD%%		TCPIP,PAGENT,CSF,TRMD,NSSD,IKED		SYSLOGD			
	CP	1141.90	CP	5.71	CP	0.00	CP	1147.60
	IIPCP	0.00	IIPCP	0.00	IIPCP	0.00	IIPCP	0.00
			Offload%	0.00%			Offload%	0.00%

Figure 10-20 Service time of queries with no encryption of data in motion



Overall	Service Time (s)							
Duration (s)	RCLASS=IDAAQRY		RCLASS=TCPIP		RCLASS=SYSLOGD		TOTAL	
3767.08	JES2 R$LOAD%%		TCPIP,PAGENT,CSF,TRMD,NSSD,IKED		SYSLOGD			
	CP	1150.10	CP	74.22	CP	25.40	CP	1249.71
	IIPCP	0.00	IIPCP	69.65	IIPCP	0.00	IIPCP	69.65
			Offload%	93.85%			Offload%	5.57%

Figure 10-21 Service time of queries with encryption of data in motion

The performance effect of encryption of data in motion on queries is not as dramatic as on load. This result occurs because the data amount transferred in between z/OS and the accelerator is much smaller with the query workload run than the load run.

Incremental updates tests with parallel apply enabled were also conducted, comparing with and without encryption of data in motion. The result showed up to 16% replication throughput regression after enabling the encryption of data in motion, compared to no encryption of data in motion.

10.8 IDAA version 5 PTF-2

IDAA version 5 PTF2 is the minimum requirement to support the IDAA features with Db2 12.

10.8.1 Db2 12 support for IDAA

Db2 12 support for IDAA is available with Db2 12 GA code at Function Level 100. The performance of load, reload, query, AOT, and encryption of data in motion enhancements were evaluated and results show equivalent performance with Db2 11.

10.9 IDAA version 5 PTF-3

In this section, we describe performance enhancements for loading many small tables that are made available in V5 PTF-3.

10.9.1 Loading many small tables

In IDAA v5 PTF-3, new features improve the performance of loading small tables. The enhancements include better concurrency during loads and internal code optimizations in Netezza.

Performance of loading many small tables

Performance tests were conducted by using a Striper with full rack (N2001). One stored procedure call was made per table that was loaded.

Concurrent load with 40 threads

We ran a concurrent load scenario that used 40 threads in which each thread loaded 100 tables that contained only one to a few rows. These parameters were used to demonstrate the best benefit from the enhancement. In total, 4,000 tables (40 threads * 100 tables) were processed.

When comparing with and without this enhancement (IDAA v5 PTF-3 to IDAA v5 PTF-2), we made the following observations:

- Approximately 30% elapsed time saving from the load was realized.
- CPU time can increase slightly (a few percent) on the Db2 side if there is increased contention because of increased concurrency. Otherwise, the CPU time remained the same.

Single stream load

We also ran single stream load in which one thread loads 100 1-row tables.

When comparing with and without the enhancement (IDAA v5 PTF-3 to IDAA v5 PTF-2), we made the following observations:

- Approximately 70% elapsed time saving because of internal optimization was realized.
- The CPU time remained same.

11

Installation and migration

In this chapter, we describe IBM Db2 12 installation and migration changes, focusing on performance. In contrast to IBM Db2 11's multiple migration modes, Db2 12 provides a single migration job, and a new ACTIVATE command is introduced to activate new function levels.

Db2 12 also modified how access paths and cache information for dynamic queries are stored in the catalog tables in support of implementing dynamic plan stability. Furthermore, some performance-related system parameters are modified that correspond to the various performance-related enhancements that were introduced in Db2 12.

This chapter includes the following topics:

11.1 Migration performance

Db2 12 eliminates the various migration modes that existed in Db2 11, such as CM, CM*, ENFM, ENFM*, and New Function Mode (NFM). Db2 12 function level V12R1M100 is equivalent to CM, or conversion mode in Db2 11, while function level V12R1M500 is equivalent to Db2 11's NFM. CM* mode is now called a "star" function level. For more information about function levels, see *DB2 12 for z/OS Technical Overview*, SG24-8383.

When migrating to Db2 12, the single migration job DSNTIJTC (CATMAINT) performs all changes necessary to tailor the Db2 11 catalog and directory objects. After running DSNTIJTC, the Db2 catalog reaches catalog level of V12R1M500.

Catalog and directory objects at V12R1M500 level can be used by Db2 12 (regardless of the function level) and by Db2 11 with the fallback SPE applied. As a result, if the fall back SPE is in place, fallback to Db2 11 or coexisting with Db2 11 members (in data sharing configuration) is possible after the catalog is changed to the Db2 12 format.

In Db2 12, no job is available to enable NFM. Instead, the new Db2 **ACTIVATE** command is used to activate new functions. Because only one step is needed to update the catalog during migration, catalog availability is greatly improved. As a result, applications that access the catalog expect less interruption during the migration.

The important migration steps for Db2 12 are shown in Figure 11-1. The crossed-out squares represent the jobs from Db2 11's migration process that are no longer required.

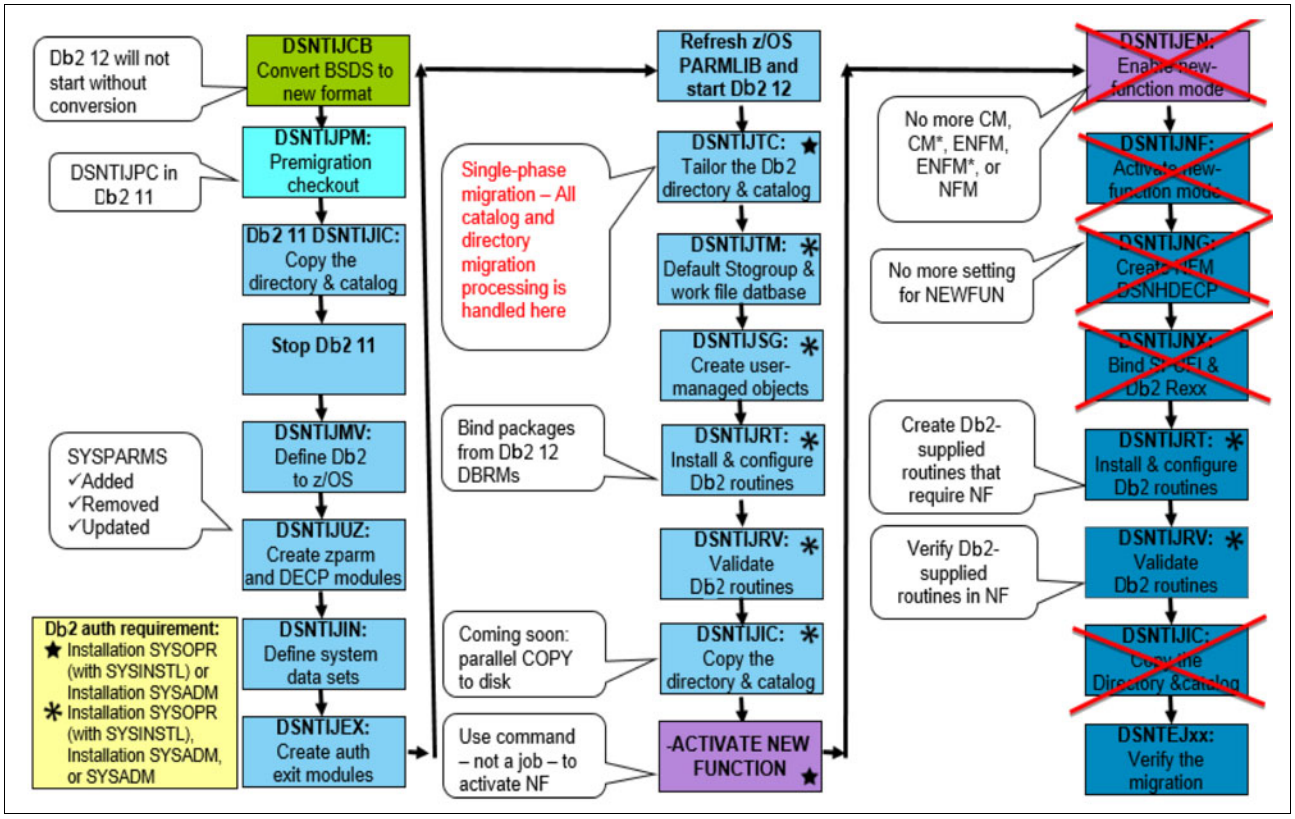

Db2 12 will not start without conversion
DSNTIJCB Convert BSDS to new format
DSNTIJPC in Db2 11
DSNTIJPM: Premigration checkout
Db2 11 DSNTIJIC: Copy the directory & catalog
Stop Db2 11
SYSPARMS ✓Added ✓Removed ✓Updated
DSNTIJMV: Define Db2 to z/OS
DSNTIJUZ: Create zparm and DECP modules
Db2 auth requirement: ★ Installation SYSOPR (with SYSINSTL) or Installation SYSADM ✱ Installation SYSOPR (with SYSINSTL), Installation SYSADM, or SYSADM
DSNTIJIN: Define system data sets
DSNTIJEX: Create auth exit modules
Single-phase migration – All catalog and directory migration processing is handled here
Bind packages from Db2 12 DBRMs
Coming soon: parallel COPY to disk
Use command – not a job – to activate NF
Refresh z/OS PARMLIB and start Db2 12
DSNTIJTC: ★ Tailor the Db2 directory & catalog
DSNTIJTM: ✱ Default Stogroup & work file datbase
DSNTIJSG: ✱ Create user-managed objects
DSNTIJRT: ✱ Install & configure Db2 routines
DSNTIJRV: ✱ Validate Db2 routines
DSNTIJIC: ✱ Copy the directory & catalog
-ACTIVATE NEW FUNCTION ★
No more CM, CM*, ENFM, ENFM*, or NFM
No more setting for NEWFUN
Create Db2-supplied routines that require NF
Verify Db2-supplied routines in NF
~~DSNTIJEN: Enable new-function mode~~
~~DSNTIJNF: Activate new-function mode~~
~~DSNTIJNG: Create NFM DSNHDECP~~
~~DSNTIJNX: Bind SPUFI & Db2 Rexx~~
DSNTIJRT: ✱ Install & configure Db2 routines
DSNTIJRV: ✱ Validate Db2 routines
~~DSNTIJIC: Copy the Directory & catalog~~
DSNTEJxx: Verify the migration

Figure 11-1 Db2 12 migration process

Note: For more information about the migration process, see *DB2 12 for z/OS Technical Overview*, SG24-8383.

11.1.1 Migration performance measurement tests

To assess Db2 12's migration performance, two series of migration tests were run in laboratory-controlled environments. In this section, the performance results of these migration tests are described to provide a general reference for users to estimate the expected CPU and elapsed time consumptions when migrating to Db2 12.

The two series of migration tests used two different-sized Db2 catalogs. The following test subjects were used:

- A 28.3 GB catalog in Db2 10
- A 5.92 GB catalog in Db2 10

Both tests scenarios consisted of the following tasks:

- Migrate the catalog from Db2 10 NFM to Db2 11 NFM.
- Migrate the catalog from Db2 11 NFM to Db2 12 Function Level 500.

The system configuration details of the measuring environment are listed in Table 11-1.

Table 11-1 Measurement environment system configuration

System configuration	
Hardware	z13 (model 2964)
Processors	4 general CPs
Operating system	z/OS 2.1

The Db2 subsystem configuration details of the measuring environment are listed in Table 11-2.

Table 11-2 Measurement environment system configuration

Db2 system configuration	
Configuration	Non-Data Sharing configuration
Db2 version	Db2 10, Db2 11, and Db2 12
Buffer pool assignments (VPSIZE)	
BP0	50 K
BP4 (work file)	10K
BP8K0	50 K
BP16K0	50 K
BP32K	50 K

The performance results were captured at the following migration steps:

- Migration from V10 to V11 CM: Measuring CATMAINT UPDATE
- Migration from V11 CM to V11 ENFM: Measuring DSNTIJEN performance, which performs online REORG of the following table spaces:
  - SYSLGRNX (SHRLEVEL CHANGE)
  - SYSCOPY
  - SYSRTSTS
  - SYSTSIXS (SHRLEVEL CHANGE)
  - SYSSTR
  - SYSTSTAB (SHRLEVEL CHANGE)
- Migration from V11 to V12:
  - Measuring CATMAINT UPDATE
  - Measuring the elapsed time and Db2 address space CPU time for the completion of the new ACTIVATE command

The migration performance results for the 28.3 GB Db2 10 catalog are listed in Table 11-3 on page 326. The CPU and elapsed time indicates that migration to Db2 12 requires 57% less CPU to complete compared with migration to Db2 11 and the migration to Db2 12 can complete 23% quicker than the migration from Db2 10 to Db2 11.

Table 11-3 Catalog 1 (28.3 GB) migration performance observations

	Db2 10 to Db2 11		**Db2 11 to Db2 12**		**DELTA (%)**	
	CPU	ELAPSED	CPU	ELAPSED	CPU	ELAPSED
CATMAINT UPDATE (sec.)	0.060	4.69	10.53	52.78		
Migration to NFM or FL activation (sec.)	24.5	64.5	0.01	0.28		
Total (sec.)	24.5	69.2	10.54	53.1	-57%	-23%

The migration performance results for the 5.92 GB Db2 10 catalog are listed in Table 11-4.

Table 11-4 Catalog 2 (5.92 GB) migration performance observations

	Db2 10 to Db2 11		**Db2 11 to Db2 12**		**DELTA (%)**	
	CPU	ELAPSED	CPU	ELAPSED	CPU	ELAPSED
CATMAINT UPDATE (sec.)	0.059	5.42	8.75	33.05		
Migration to NFM or FL activation (sec.)	39.0	76.9	0.00	0.20		
Total (sec.)	39.1	82.3	8.75	33.3	-78%	-60%

These results show that the migration from Db2 11 to Db2 12 requires 78% less CPU time to complete compared to migrating from Db2 10 to Db2 11. The migration to Db2 12 also shows a 60% improvement in elapsed time compared to the migration to Db2 11.

As can be observed from these test results, the major cost saving of Db2 12 migration comes from the Migration to NFM or FL activation phase. The migration from Db2 11 CM to Db2 11 NFM is done through online REORG in Db2 11, which took over 1 minute to complete in the tests. The migration from Db2 12 V12R1M100 to V12R1M500 is done by using the `ACTIVATE` command, which took sub seconds to complete in the tests.

Migration from Db2 11 CM to Db2 11 NFM for Catalog 2, which is smaller than Catalog 1, took longer to complete and used more CPU. The reason for this difference is Catalog 1's bigger size is a result of a much bigger SPT01 table space. In fact, Catalog 1 has a smaller combined size for table spaces SYSLGRNX, SYSCOPY, SYSRTSTS, SYSTSIXS, SYSSTR, and SYSTSTAB compared to Catalog 2. Because migration from Db2 11 CM to NFM starts online REORG of these six table spaces, Catalog 2's bigger combined size of them caused job DSNTIJEN to run longer and cost more CPU.

11.2 Dynamic plan stability

Unlike static SQL, whose access path and runtime structures are secured at BIND/REBIND time, repeating dynamic SQL query performance can suffer from instability because of factors, such as cache misses and access path changes. For example, if Db2 is recycled, or a cached dynamic statement is flushed out of the dynamic statement cache, a cache miss occurs and a full prepare for the statement is required when a dynamic statement that was cached earlier must be run again.

The access path and runtime structures of full prepares might change if the inputs to the optimizer change because of situations, such as a RUNSTATS after data changes, system parameter changes, system maintenance, and release migration. If the changed access path is less optimal, this change results in a query regression.

Starting with Db2 12, Db2 provides a new mechanism to stabilize selected dynamic SQL statements by allowing their access paths and cached runtime structures to be saved into a new catalog table. These saved access paths and cache structures are located and loaded when cache misses for the corresponding dynamic SQL statements occur. With this new mechanism, applications with repeated dynamic SQL can get stable and reliable performance similar to static SQL. Also, by replacing expensive full prepares with cache loads from Db2 catalog tables, these applications can experience significant performance improvement.

11.2.1 Stabilizing dynamic queries

After the dynamic SQL statements are cached in the dynamic statement cache, users can issue the Db2 `START DYNQUERYCAPTURE` command to capture a single dynamic statement or a group of statements that meet the criteria that is specified in the command, such as THRESHOLD and CURSQLID. For the queries with an execution count that is less than the THRESHOLD criteria at the time of the capture, Db2 continues to monitor the queries and capture the statements automatically after their execution counts meet the THRESHOLD criteria if MONITOR YES was also specified in the capture command.

Cost of capturing dynamic queries

Several performance measurements were conducted by using a z13 LPAR that is running z/OS 2.1 with six general CPs to evaluate the capture cost of the dynamic plan stability feature of Db2 12.

System ZPARMs that are related to the tests featured the following configuration:

- STATSINT=1 minute
- CACHEDYN_STABILIZATION=BOTH
- CACHEDYN=YES

Also, IFCID 316, 317, and 318 are turned on.

When EDMSTMTC is 1 GB, the dynamic statement cache holds up to 950 complex statements or 80,000 simple statements in the measurements. When EDMSTMTC is 4 GB, the dynamic statement cache holds up to 3,600 complex statements or 320,000 simple statements in the measurements.

No stabilized statements existed before the performance data was captured.

Elapsed time is measured as the duration between the DSNX221I and DSNX222I messages, which marks the start of the dynamic query capture command and the completion of the capture process.

CPU time is taken from CP CPU TIME for DATABASE SERVICES ADDRESS SPACE (DBM1) in the statistics report, which consists of the CPU cost of the capture process and that of the capture command. The CPU cost is zIIP eligible.

Capture performance results

As expected, the dynamic query capture overhead varies depending on the sizes of the statements and their cache structures, and the number of statements that are captured. The test results that are shown in Figure 11-2 show that larger, more complex SQL statements require more CPU per statement to capture.

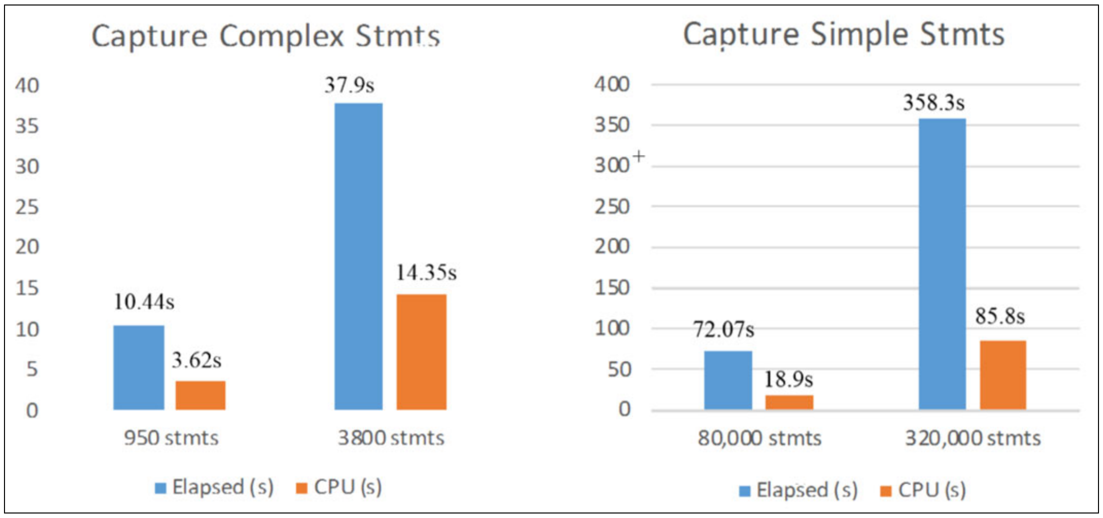


Figure 11-2 Capture performance results

MONITOR YES overhead

MONITOR YES relieves users from the hassle of periodically performing manual snapshot captures. By using this option, Db2 periodically monitors the cache candidate statements' execution count and captures the candidate statements automatically whenever the execution count reaches the THRESHOLD.

The system parameter STATSINT controls interval of materialization of real-time statistics. Starting in Db2 12, the same parameter also controls how often Db2 checks statement execution counts.

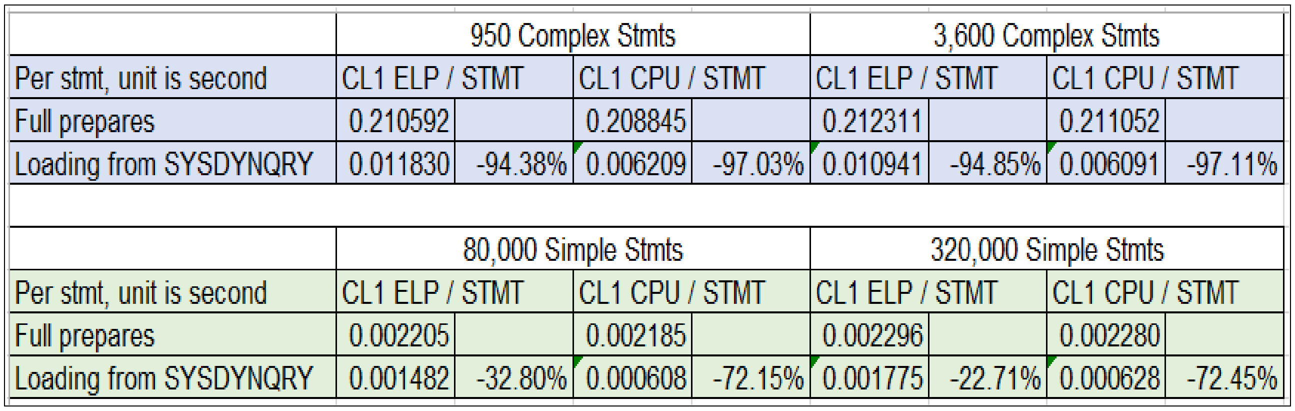
In our test environment, STATSINT is set to 1 minute, which means that Db2 checks the statement execution count every minute. In our test with 320,000 simple statements in the dynamic statement cache with no other user activities, the statistics trace shows approximately 2 seconds of CP CPU TIME in DATABASE SERVICES ADDRESS SPACE for a 30-minute period. Therefore, the overhead that was incurred by MONITOR YES is small.

11.2.2 Locating and loading stabilized statements

Without dynamic plan stability, a dynamic SQL statement that is not found in the dynamic statement cache goes through a full prepare. With dynamic plan stability, Db2 checks for a match in stabilized statements that are stored in the Db2 catalog after a cache miss. If a match is located, this stabilized statement and its cache structures are loaded from the catalog back into the dynamic statement cache and the full prepare is avoided. In this way, the query can be run with the stabilized access path and reliable query performance can be achieved. If a match is not located among the stabilized queries in the catalog, the query goes through a full prepare.

Locating and loading performance results

The performance measurements in this section compare the cost of locating and loading the stabilized statements and their cache structures from the catalog against the cost of full prepares. As shown in Figure 11-3, for our 950 and 3,600 complex queries cases, we see over 90% elapsed time and CPU time savings with locating and loading the stabilized statements from the catalog compared with doing full prepares. For our 80,000 and 320,000 simple queries, we see over 20% elapsed time and 70% CPU time savings with matching stabilized statements over full prepares.



	950 Complex Stmts				3,600 Complex Stmts			
Per stmt, unit is second	CL1 ELP / STMT		CL1 CPU / STMT		CL1 ELP / STMT		CL1 CPU / STMT	
Full prepares	0.210592		0.208845		0.212311		0.211052	
Loading from SYSDYNQRY	0.011830	-94.38%	0.006209	-97.03%	0.010941	-94.85%	0.006091	-97.11%

	80,000 Simple Stmts				320,000 Simple Stmts			
Per stmt, unit is second	CL1 ELP / STMT		CL1 CPU / STMT		CL1 ELP / STMT		CL1 CPU / STMT	
Full prepares	0.002205		0.002185		0.002296		0.002280	
Loading from SYSDYNQRY	0.001482	-32.80%	0.000608	-72.15%	0.001775	-22.71%	0.000628	-72.45%

Figure 11-3 Locate and load performance results

11.2.3 Overhead of stabilized statement misses

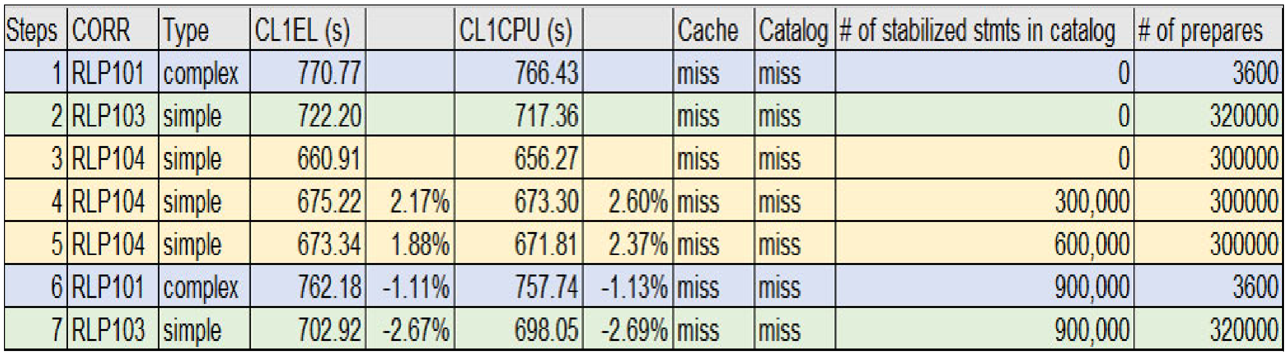
With dynamic plan stability, the query is forced to go through a full prepare if locating the matching stabilized statements results in a miss. For these situations, the process of finding the matching stabilized statements is pure overhead compared to not using dynamic plan stability.

To determine the amount of overhead of stabilized statement misses, we performed a series of tests. Before the tests start, no stabilized dynamic statements are available. Db2 is recycled and the dynamic statement cache is empty. The test is done by using the following process:

1. Full prepare the 3,600 RLP101 complex queries when no stabilized queries are present. This result is the baseline to compare to for Step 6, in which the same 3,600 RLP101 complex queries go through a full prepare when 900,000 unmatched stabilized queries are in the catalogs.

2. Recycle Db2 to clear out the dynamic statement cache. Full prepare the 320,000 RLP103 simple queries when no stabilized queries are present. This result is the baseline to compare to for Step 7, where the same 320,000 RLP103 simple queries go through a full prepare when 900,000 unmatched stabilized queries are in the catalogs.
3. Recycle Db2 to clear out the dynamic statement cache. Full prepare the 300,000 RLP104 simple queries when no stabilized queries are present. This result is the baseline to compare to for steps 4 and 5, where the similar 300,000 RLP104 simple queries go through a full prepare when unmatched stabilized queries are in the catalogs.
4. Capture all of the Step 3 queries into catalog to have 300,000 stabilized queries in catalogs. Recycle Db2 to clear out the dynamic statement cache. Modify the 300,000 queries in Step 3 to use different set of values in the local predicates so that they are not matched from the catalogs. Full prepare these 300,000 RLP104 simple queries with 300,000 unmatched stabilized queries in the catalogs. Compare the full prepare performance with Step 3.
5. Capture all of the Step 4 queries into catalog to have 600,000 stabilized queries in catalogs. Recycle Db2 to clear out the dynamic statement cache. Modify the 300,000 queries in Step 3 to use a different set of values in the local predicates so that they do not get matched from the catalogs. Full prepare these 300,000 RLP104 simple queries with 600,000 unmatched stabilized queries in the catalogs. Compare the full prepare performance with Step 3.
6. Capture all of the Step 5 queries into catalog to have 900,000 stabilized queries in catalogs. Recycle Db2 to clear out the dynamic statement cache. Full prepare 3,600 RLP101 complex queries with 900,000 unmatched stabilized queries in the catalogs. Compare the full prepare performance with Step 1.
7. Recycle Db2 to clear out the dynamic statement cache. Full prepare 320,000 RLP103 simple queries with 900,000 unmatched stabilized queries in the catalogs. Compare the full prepare performance with Step 2.

The test results are shown in Figure 11-4.



Steps	CORR	Type	CL1EL (s)		CL1CPU (s)		Cache	Catalog	# of stabilized stmts in catalog	# of prepares
1	RLP101	complex	770.77		766.43		miss	miss	0	3600
2	RLP103	simple	722.20		717.36		miss	miss	0	320000
3	RLP104	simple	660.91		656.27		miss	miss	0	300000
4	RLP104	simple	675.22	2.17%	673.30	2.60%	miss	miss	300,000	300000
5	RLP104	simple	673.34	1.88%	671.81	2.37%	miss	miss	600,000	300000
6	RLP101	complex	762.18	-1.11%	757.74	-1.13%	miss	miss	900,000	3600
7	RLP103	simple	702.92	-2.67%	698.05	-2.69%	miss	miss	900,000	320000

Figure 11-4 Test results

The results show that no significant overhead results from locating the stabilized statements in cache miss situations when up to 900,000 stabilized statements are in Db2 catalog. Periodic cleanup of stabilized statements ensures that the overhead stays low. Consider the following points:

- Comparing to Step 3, Step 4 has less than 3% CPU overhead.
- Comparing to Step 3, Step 5 has less than 3% CPU overhead.
- Comparing to Step 1, Step 6 has no overhead.
- Comparing to Step 2, Step 7 has no overhead.

11.2.4 Free stabilized statements

When the user wants to explore new access paths or when the stabilized queries are no longer needed, the user can issue the **FREE STABILIZED DYNAMIC QUERY** command to free the stabilized queries from the catalog. The user can experience many synchronous I/Os against the catalog table when stabilized statements are freed. The synchronous I/O numbers that are shown in Figure 11-5 are from a test that freed 320,000 simple stabilized queries.

```
CLASS 3 SUSPENSIONS    ELAPSED TIME    EVENTS
--------------------   ------------  --------
SYNCHRON. I/O            6:20.616619   2572565
 DATABASE I/O            6:20.336360   2571937
SER.TASK SWTCH           2:39.182994    319422
 UPDATE COMMIT           2:39.182994    319422
TOTAL CLASS 3            9:00.274444   2893220
```

Figure 11-5 Synchronous I/O data

Freeing stabilized statements can also cause DATABASE SERVICES ADDRESS SPACE (DBM1) CPU time to increase. The more statements to free and the bigger their size, the more CPU cost is to be expected. Figure 11-6 on page 331 shows the elapsed and CPU time (including the cost of the **FREE** command and FREE process) for freeing stabilized statements.

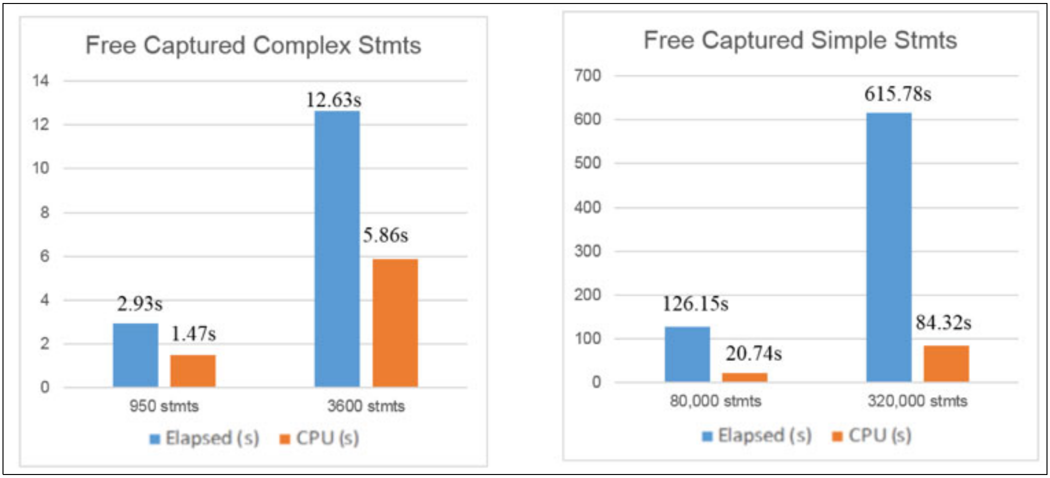


Figure 11-6 Elapsed and CPU time for freeing stabilized statements

11.3 Temporal real-time statistics tables

Temporal capabilities were introduced in Db2 10 and enhanced in Db2 11. One of the functionalities temporal tables can provide is to track information for data change auditing in a transparent manner. Before Db2 12, this feature was limited to application tables and was not available for Db2 catalog tables. In Db2 12, system-period data versioning can be enabled for real-time statistics (RTS) catalog tables.

RTS tables are valuable for tracking the size, activity, and organization changes of table spaces and index spaces. By using Db2 12, customers can enable system-period data versioning for real-time statistics catalog tables, SYSIBM.SYSINDEXSPACESTATS, and SYSIBM.SYSTABLESPACESTATS.

By using a temporal relationship, the previous version of the row's data (before an update) is inserted into the history table when a base table row is updated. The temporal relationship for real-time statistics table is "soft", meaning that a failure of the insert into the history table does not undo the changes that are made to the base table.

The ALTER TABLE ADD VERSIONING statement is used to enable system-period versioning for the real-time statistics tables, as shown in the following example:

```
ALTER TABLE SYSIBM.SYSTABLESPACESTATS ADD VERSIONING USE HISTORY TABLE
SYSIBM.SYSTABSPACESTATS_H;
ALTER TABLE SYSIBM.SYSINDEXSPACESTATS ADD VERSIONING USE HISTORY TABLE
SYSIBM.SYSIXSPACESTATS_H;
```

The ALTER TABLE DROP VERSIONING statement is used to disable system-period versioning of real-time statistics tables, as shown in the following example:

```
ALTER TABLE SYSIBM.SYSTABLESPACESTATS DROP VERSIONING;
ALTER TABLE SYSIBM.SYSINDEXSPACESTATS DROP VERSIONING.
```

INSERT, UPDATE, and DELETE statements are allowed for the history tables with proper authorization.

The history tables SYSTABSPACESTATS_H and SYSIXSPACESTATS_H are created as part of the CATMAINT process during installation and migration of Db2 12.

Temporal functionality for real-time statistics tables is optional and is available only after Db2 12 is operating at function level V12R1M500.

11.3.1 Performance measurements

To verify the performance effect of enabling system-period data versioning for the RTS tables, the IBM Brokerage workload was used and run twice with Db2 12 at function level V12R1M500 (once with RTS system-period data versioning enabled and once with the functionality disabled).

To analyze the performance effect on the workload from versioning the RTS tables, the RTS statistics were externalized every minute by setting STATINT to 1, instead of the default interval of 30 minutes. The workload was run for approximately 6 hours before measuring CPU and elapsed time for the measurement period of 1 hour.

During the 1 hour measurement with RTS system-period data versioning enabled, the size of the RTS history tables grew as is to be expected. In all, 298,192 rows were added to SYSIBM.SYSTABSPACESTATS_H and 491,996 rows were inserted into SYSIBM.SYSIXSPACESTATS_H. More inserts are expected to introduce more workload cost. However, as the data that is shown in Figure 11-7 on page 333 and Figure 11-8 on page 334 indicates, no significant CPU or elapsed time affected the measured workload when RTS versioning was enabled.

Figure 11-7 shows the Db2 class 2 elapsed time per commit for the Brokerage workload during the 1 hour measurement period with and without RTS system-period versioning enabled.

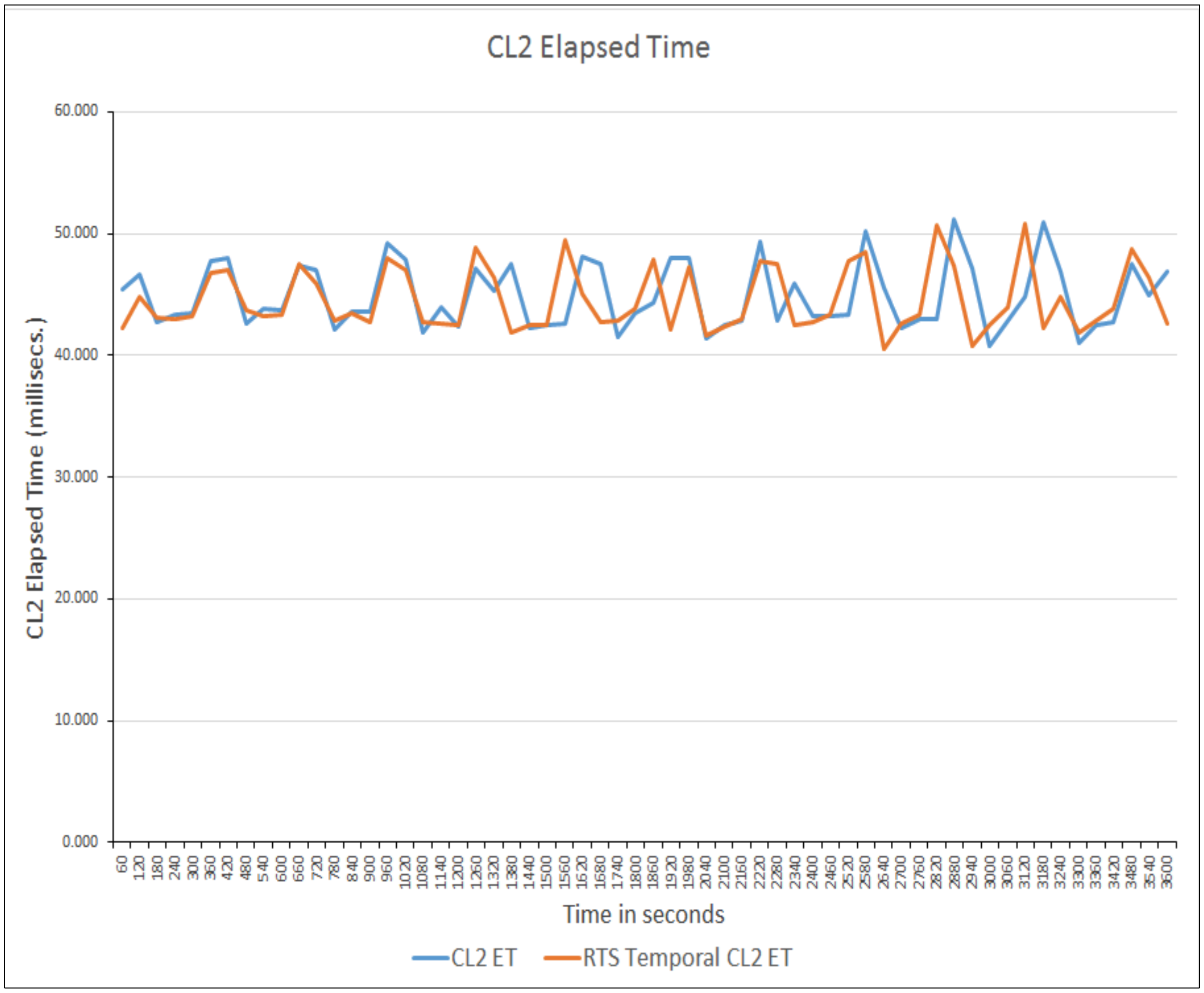


Figure 11-7 Class 2 elapsed time with RTS temporal not enabled versus enabled

Figure 11-8 shows the Db2 class 2 CPU time per commit for the Brokerage workload during the 1 hour measurement period with and without RTS system-period versioning. The daemon process that externalizes the RTS statistics runs in the DBM1 address space. No significant increase in DBM1 CPU time was observed when comparing the measurements with and without RTS system-period versioning enabled.

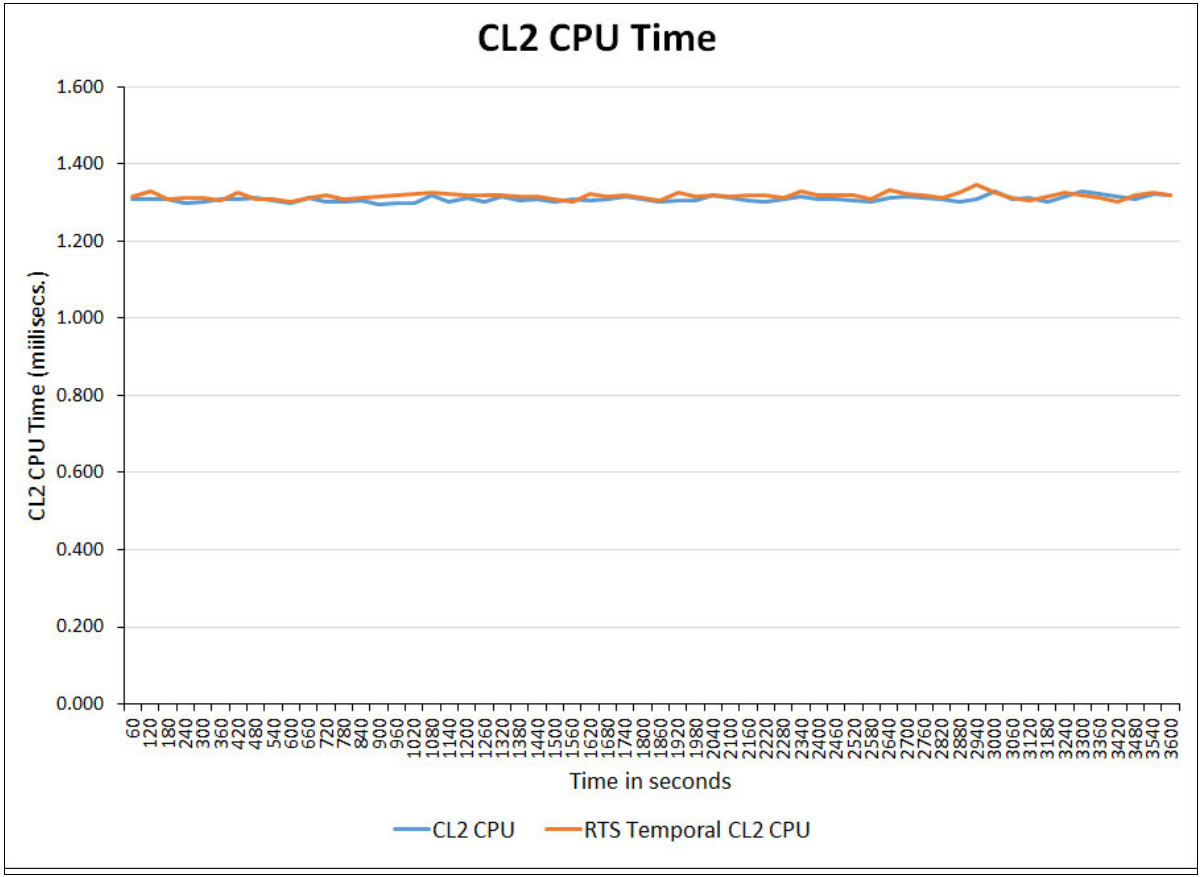


Figure 11-8 Class 2 CPU time with RTS temporal not enabled versus enabled

11.4 System parameters' performance-related changes

Db2 12 introduces many new features and enhancements. To support these changes, some new Db2 system parameters were added, some old parameters' default values were changed, and several parameters were made obsolete. The ZPARM changes that are related to Db2 performance are described in this section.

11.4.1 New system parameters related to performance

Several new system parameters can affect Db2 performance.

CACHEDYN_STABILIZATION

The CACHEDYN_STABILIZATION subsystem parameter specifies whether dynamic SQL statements can be captured for stabilization, and whether Db2 uses the captured runtime structures to process stabilized dynamic SQL statements that are not present in the dynamic statement cache.

Db2 avoids processing full prepares when it uses the captured runtime structures. When a statement is stabilized, the current SQLID, statement text, and runtime structures are written to the SYSIBM.SYSDYNQUERY catalog table.

The following values for CACHEDYN_STABILIZATION are acceptable:

- CAPTURE: Dynamic SQL statements can be stabilized through the **`-START DYNQUERY`** command by using **`MONITOR(NO)`** and **`MONITOR(YES)`**. However, Db2 does not load stabilized statements from SYSDYNQUERY.
- LOAD: Dynamic SQL statements cannot be stabilized by any means. The **`-START DYNQUERY`** command fails, and any **`MONITOR(YES)`** commands in progress do not stabilize statements (even if stabilization criteria are matched). During a full prepare, Db2 attempts to load stabilized statements from SYSDYNQUERY with which to run.
- BOTH: Dynamic SQL statements can be stabilized through the **`-START DYNQUERY`** command by using **`MONITOR(NO)`** and **`MONITOR(YES)`**. During a full prepare, Db2 attempts to load stabilized statements from SYSDYNQUERY.
- NONE: Statements cannot be stabilized by using any means. The **`-START DYNQUERY`** command fails, and any **`MONITOR(YES)`** commands in progress do not stabilize statements, even if stabilization criteria are matched. Db2 does not load stabilized statements from SYSDYNQUERY.

BOTH is the default value of CACHEDYN_STABILIZATION. This parameter is online changeable.

For more information about the dynamic SQL statement stabilization feature, see 11.2, "Dynamic plan stability" on page 327.

COMPRESS_DIRLOB

COMPRESS_DIRLIB specifies whether LOB table spaces in the Db2 directory are to be compressed. In Db2 data sharing, all members should use the same setting.

The following values for COMPRESS_DIRLOB are acceptable:

- NO: LOB table spaces in the Db2 directory are not compressed.
- YES: LOB table spaces in the Db2 directory are compressed the next time that they are reorganized.

NO is the default value of COMPRESS_DIRLOB. This parameter is online changeable.

DEFAULT_INSERT_ALGORITHM

DEFAULT_INSERT_ALGORITHM is introduced for the new Insert Algorithm 2 feature of Db2 12. It specifies the default algorithm for inserting data into table spaces, which affects the insert algorithm for MEMBER CLUSTER universal table spaces for which INSERT ALGORITHM 0 was specified. This system parameter takes effect only when the activated function level is function level 500 or higher.

The following values for DEFAULT_INSERT_ALGORITHM are acceptable:

- 1: Indicates that the basic insert algorithm is used.
- 2: Indicates that the new insert algorithm is used.

The 2 value is the default value for DEFAULT_INSERT_ALGORITHM. This parameter is online changeable.

For more information about the new insert algorithm, see 2.3, “Insert Algorithm 2” on page 50.

INDEX_MEMORY_CONTROL

INDEX_MEMORY_CONTROL specifies the amount of memory that Db2 can be allocated for fast traversal of Db2 indexes. It is introduced in Db2 12 in support of new fast index traversal feature.

The following values for INDEX_MEMORY_CONTROL are acceptable:

- AUTO: Db2 sets the upper limit of the storage that is used for the FTBs to 20% of the currently allocated buffer pools or 10 MB, whichever value is larger.
- DISABLE: Db2 returns any storage that is allocated for fast index traversal and allocates no other storage for this purpose. This process results in the fast index traversal effectively being disabled.
- 10 - 200000: Indicates the storage limit in MBs for fast index traversal.

The default value for INDEX_MEMORY_CONTROL is AUTO. This parameter is online changeable.

For more information about the fast index traversal feature, see 2.1, “Fast index traversal” on page 24.

RLFENABLE

The RLFENABLE subsystem parameter specifies whether the resource limit facility limits resources that are used by dynamic SQL statements only, static SQL statement only, or all SQL statements.

The following values for INDEX_MEMORY_CONTROL are acceptable:

- DYNAMIC: Resource limits apply to dynamic SQL statements only.
- STATIC: Resource limits apply to static SQL statements only.
- ALL: Indicates that resource limits apply to dynamic and static SQL statements.

The default value for RLFENABLE is DYNAMIC.

PAGESET_PAGENUM

PAGESET_PAGENUM specifies whether range-partitioned table spaces and the associated partitioned indexes that are created are to use absolute page numbers across partitions, or relative page numbers by default.

The following values for PAGESET_PAGENUM are acceptable:

- ABSOLUTE: Range-partitioned table spaces and associated indexes are created with absolute page numbers, as is the case in Db2 11.
- RELATIVE: Range-partitioned table spaces are created so that PGNUM in the page header has no partition number, and the partition number is contained only in the header page for the partition.

The default value of PAGESET_PAGENUM is ABSOLUTE.

Note: It is recommended that the same value is used on all data sharing members to maintain consistency. The use of this value avoid situations, such as there being different page numbering schemes for a table space. This kind of situation can occur by running the same CREATE TABBLESPACE DML without specifying the PAGENUM option on different Db2 members.

For more information about how relative page numbering works and related performance test results, see 2.5, “Range-partitioned table spaces with relative page numbering ” on page 63.

PEER_RECOVERY

PEER_RECOVERY specifies whether a data sharing member is to participate in data sharing peer recovery.

The following values for PEER_RECOVERY are acceptable:

- NONE: This member is not to participate in peer recovery. Use this option if you configured the z/OS Automatic Recovery Manager (ARM) to restart failed Db2 members.
- RECOVER: This member should be recovered by a peer member if it fails.
- ASSIST: This member should attempt to start peer recovery for other failed members. When this member detects a failure, it attempts to start a LIGHT(YES) restart for the failed member if it does not start to recover the retained locks.
- BOTH: RECOVER and ASSIST options are to be activated for this member.

The default value for PEER_RECOVERY is NONE.

For more information about the performance-related test results on peer recovery, see 4.7, “Db2 peer recovery” on page 149.

REMOTE_COPY_SW_ACCEL

REMOTE_COPY_SW_ACCEL specifies whether Db2 uses software to control the remote copy process for active log writes in Peer-to-Peer Remote Copy (PPRC) environments.

The following values for REMOTE_COPY_SW_ACCEL are acceptable:

- DISABLE: The hardware PPRC function is used if available. This setting is appropriate for non-PPRC environments.
- ENABLE: The host software is used. This option can improve overall Db2 performance in PPRC environments at the expense of some additional SRB CPU time in the xxxxMSTR address space. z/OS Version 2 Release 1 or later is required to support this function.

The default value for REMOTE_COPY_SW_ACCEL is DISABLE. This parameter is online changeable.

For more information about zHyperWrite technology and the performance-related test results when the REMOTE_COPY_SW_ACCEL option is used, see 3.8, “zHyperWrite ” on page 122.

11.4.2 System parameters updated related to performance

Some system parameters were updated that affect performance.

EDM_SKELETON_POOL

The default value of EDM_SKELETON_POOL is now 51,200 KB instead of 10,240 KB. This change was introduced for Db2 EDM pool management to use Db2 12 storage management changes to improve performance.

The suggestion is to keep the value of EDM_SKELETON_POOL as you did before migration, but to continue to monitor EDM statistics report, paying particular attention to the PAGES IN SKEL POOL section, and consider increasing EDM_SKELETON_POOL setting if the sum of HELD BY SKCT (field name QISESKCT) and HELD BY SKPT (field name QISESKPT) values are close to the current EDM_SKELETON_POOL value during peak periods. You can change this value by using the **`SET SYSPARM`** command.

For more information, see 2.9, “Db2 Storage Manager enhancements” on page 76.

SMFACCT

SMFACCT is used to specify which Db2 accounting trace classes are to be started automatically and sent to SMF.

In Db2 12, its default value changed to “1,2,3,7,8” from its original default of “1”.

This change is introduced because most customers enable accounting traces for classes 1, 2, 3, 7, and 8. With the SMFACCT default now set to “1,2,3,7,8”, accounting traces for classes 2, 3, 7, and 8 are activated after a Db2 recycle. This setting avoids the need to run a **`-START TRACE`** command for these classes after recycling Db2.

11.4.3 Obsolete system parameters related to performance

Some system parameters that are related to performance were made obsolete in Db2 12.

ALCUNIT

Before Db2 12, ALCUNIT was used to specify the primary and secondary space allocation units for archive log data sets in previous Db2 releases. Because modern Db2 log archiving demands are not well-suited for allocation in tracks or blocks, Db2 12 always allocates archive log data sets in cylinders.

CONTSTOR

Before Db2 12, CONTSTOR was used to specify whether Db2 is to periodically contract each thread’s working storage area. The Db2 11 default value for CONTSTOR is ENABLE. Db2 12 operates as though CONTSTOR is set to NO to improve performance.

DB2SORT

Before Db2 12, DB2SORT was used to specify whether Db2 utilities are to use Db2 Sort instead of DFSORT for utility sort processing when Db2 Sort is installed. Db2 11 features a default of ENABLE for DB2SORT, which is how Db2 12 always operates.

INDEX_IO_PARALLELISM

Before Db2 12, INDEX_IO_PARALLELISM is used to specify whether I/O parallelism is enabled for index insertion. However, if enough prefetch engines are available, no reason exists not to enable I/O parallelism for index insertion. Db2 12 always operates as though INDEX_IO_PARALLELISM=YES, which is the default in Db2 11.

LOBVALA, LOBVALS, XMLVALA, and XMLVALS

Before Db2 12, these values were used to determine how much storage Db2 can be used to store LOB/XML values in memory. In Db2 12, the Db2 database manager automatically determines the amount of storage that is needed for LOB/XML values in memory.

For more information, see 2.9.2, “Storage pool management improvement for XML and LOBs” on page 81.

MINSTOR

Before Db2 12, the MINSTOR value was used to specify whether Db2 is to use storage management algorithms that minimize the amount of working storage that is used by individual threads. In Db2 12, the Db2 database manager automatically determines the amount of storage that is needed and MINSTOR is removed.

12 Monitoring and instrumentation

Db2 12 introduces many changes to monitoring and instrumentation. These changes include adding Instrumentation Facility Component Identifiers (IFCIDs) and modifying some existing IFCIDs, updates to Db2 accounting and statistics information, and the support by IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS.

This chapter includes the following topics:

12.1 Db2 12 instrumentation changes

This section describes the changes and additions to the IFCID fields to support the new enhancements and functionality of Db2 12. It contains the known changes and enhancements at the time of this writing. Because Db2 is moving to a continuous delivery model, further changes in the instrumentation facility are likely to occur to support the extra features and functionality that will be available.

For more information about IFCIDs, see *DB2 12 for z/OS What's New?*, GC27-8861.

For more information about IFCIDs mappings, see *.SDSNMACS and *.SDSNIVPD(DSNWMSGS) data sets that are a part of the Db2 product delivery.

12.1.1 IFCID header changes

The standard header (QWHS) that is used by all trace records is modified to include the following new fields:

- QWHS_MOD_LVL: Function level of Db2 in the form VvvRrrMmmm, where *vv* is the version, *rr* is the release, and *mmm* is the modification level.
- QWHS_REC_INCOMPAT: Incompatible change counter.
- QWHS_REC_COMPAT: Compatible change counter.

The updated QWHS header is shown in Example 12-1.

Example 12-1 Updated QWHS

```
QWHSFLAG DS    X                 Flags (new possible value added)
QWHS_REC_Validity EQU X'40'      QWHS_REC_INCOMPAT and
*                              QWHS_REC_COMMAT check needed

QWHS_MOD_LVL DS    CL10          MODIFICATION LEVEL FOR
*                                CONTINUOUS DELIVERY
QWHS_REC_INCOMPAT DS XL2         Incompatible change value
*                                incremented each time an
*                                incompatible change occurs,
*                                such as changing the size of
*                                existing fields in a record
*                                or removing fields no longer
*                                being set which causes the
*                                offset to other fields to
*                                change is made
QWHS_REC_COMPAT DS XL2           Compatible change value.
*                                Incremented each time a
*                                compatible change occurs,
*                                such as adding a new field in a
*                                reserved area no longer setting
*                                an existing field or increasing
*                                the size of a record to add a
*                               new field is made
```

12.1.2 New IFCIDs

Db2 12 introduces the following new IFCIDs:

- IFCID 389: Records information about indexes that are allocated for fast index traversal.
- IFCID 404: Serviceability trace record for new subsystem parameter AUTH_COMPATIBILITY to identify the use of the SELECT privilege for UNLOAD utility access.
- IFCID 413: The beginning of pipe wait for Insert Algorithm 2.
- IFCID 414: The end of pipe wait for Insert Algorithm 2.
- IFCID 477: Records allocation and deallocation for fast index traversal.

IFCID 389: Information about indexes that use fast index traversal

The new IFCID 389 record records information about indexes that have Fast Traversal Block (FTB) structures that are allocated in support of the fast index traversal functionality in Db2 12.

The new IFCID 389 that indicates which indexes have FTBs that are associated with them is shown in Example 12-2.

Example 12-2 New IFCID 389 for indexes with FTBs

```
*************************************************************   IFCID 0389 to
record all indexes with FTBs. Each trace *   record is limited to 5000 bytes. If
this maximum is   *   reached then multiple IFCID records are generated for   *
one snapshot.                                                *
************************************************************
QW0389   DSECT             IFCID(QWHS0389)
QW0389H  DS    0CL8        HEADER RECORD
QW0389NU DS    H           NUMBER OF INDEXES WITH FTBs
QW0389FL DS    X           IFCID389 RECORD FLAGS
QW0389CO EQU   X'80'       ANOTHER IFCID389 RECORD FOLLOWS
         DS    CL5         RESERVED
QW0389AR DS    0CL16       (S)
*
QW0389DB DS    CL2         DATA BASE ID
QW0389OB DS    CL2         INDEX PAGE SET ID
QW0389PT DS    CL2         PARTITION NUMBER
QW0389LV DS    CL2         NUMBER OF INDEX LEVELS IN FTB
QW0389SZ DS    XL4         SIZE OF FTB IN BYTES
         DS    CL4         RESERVED
*
```

IFCID 404: Serviceability trace record for new subsystem parameter AUTH_COMPATIBILITY

A new UNLOAD authorization was added to the UNLOAD utility in Db2 12. This enhancement includes a new subsystem parameter, AUTH_COMPATIBILITY. The IFCID 404 provides information about when the SELECT authority is used to allow the UNLOAD utility to unload the rows from the table.

The new IFCID 404 trace record is shown in Example 12-3.

Example 12-3 New IFCID 404

```
******************************************************
* IFCID 0404 to service authorization compatibility  *
* settings                                           *
******************************************************
QW0404        DSECT
QW0404TO      DS CL1        (S)
              DS CL3        (S)
QW0404NM      DS CL16       (S)
QW0404PR      DS H      Privilege checked
QW0404OT      DS CL1    Object type
QW0404AT      DS CL1    Authid type
*                       ' ' - Authorization ID
*                       'L' - Role
QW0404F1_Off  DS H      Offset from QW0404 to authid or a role
QW0404F2_Off  DS H      Offset from QW0404 to schema name
QW0404F3_Off  DS H      Offset from QW0404 to object name
              DS CL2        (S)
*
QW0404F1_D    DSECT
QW0404F1_Len  DS H      Length of the following field
QW0404F1_Var  DS 0CL128 %U Authid or role
*
QW0404F2_D    DSECT
QW0404F2_Len  DS H      Length of the following field
QW0404F2_Var  DS 0CL128 %U Schema name
*
QW0404F3_D    DSECT
QW0404F3_Len  DS H      Length of the following field
QW0404F3_Var  DS 0CL128 %U Object name
*
QW0404CM      DC CL1'C'
QW0404SQ      DC CL1'S'
QW0404UL      DS CL1'U'
*  (S) = FOR SERVICEABILITY
```

IFCID 413: Beginning of pipe wait for Insert Algorithm 2

IFCID 413 records the beginning of a wait for a pipe for Insert Algorithm 2. The new IFCID 413 that indicates the resource that is related to the pipe wait for Insert Algorithm 2 is shown in Example 12-4.

Example 12-4 New IFCID 413

```
**********************************************************
*  IFCID 0413 THE BEGINNING OF A WAIT FOR A PIPE           *   SUSPEND
************************************************************
QW0413    DSECT          IFCID(QWHS0413)
QW0413PN  DS    CL8      PROC NAME
QW0413RN  DS    0CL6     RESOURCE NAME
QW0413DB  DS    XL2      DATA BASE ID
QW0413PS  DS    XL2      PAGE SET ID
QW0413PT  DS    XL2      PARTITION NUMBER
          DS    CL2      RESERVED
QW0413DMS DS    XL4      (S)
QW0413CNT DS    XL2      (S)
QW0413LMT DS    XL2      (S)
QW0413FL  DS    XL2      (S)
          DS    XL2      (S)
*  (S) = FOR SERVICEABILITY
```

IFCID 414: End of pipe wait for Insert Algorithm 2

IFCID 414 records the end of a wait for a pipe for Insert Algorithm 2. The new IFCID 414 that indicates the reason for the pipe wait resume is shown in Example 12-5.

Example 12-5 New IFCID 414

```
*****************************************************************
*  IFCID 0414 RECORDS THE END OF THE WAIT FOR PIPE SUSPEND      *
*****************************************************************
QW0414    DSECT            IFCID(QWHS0414)
QW0414R   DS    CL1        REASON FOR PIPE WAIT RESUME
QW0414FL  DS    XL2        (S)
          DS    XL2        (S)
          DS    CL1        (S)
*   (S) = FOR SERVICEABILITY
```

IFCID 477: Allocation and deallocation for fast index traversal

IFCID 477 records the allocation and deallocation of structures for fast index traversal, as shown in Example 12-6.

Example 12-6 New IFCID 477 to indicate FTBs allocation or deallocation

```
*****************************************************************
*   IFCID 0477 trace record for each allocated or deallocated   *
*   Index Fast Traverse Block (FTB). If action is "create FTB"  *
*   then QW0477CO bit is ON. If action is "FREE FTB" then       *
*   QW0477CO bit is OFF.                                        *
*****************************************************************
QW0477   DSECT              IFCID(QWHS0477)
QW0477DB DS    CL2          DATA BASE ID
QW0477OB DS    CL2          INDEX PAGE SET ID
QW0477PT DS    CL2          PARTITION NUMBER
QW0477LV DS    CL2          NUMBER OF INDEX LEVELS IN FTB
QW0477SZ DS    XL4          SIZE OF FTB IN BYTES
QW0477FL DS    X            IFCID477 FLAGS
QW0477CO EQU   X'80'        THIS BIT IS ON IF ACTION -CREATE FTB
         DS    CL3          AVAILABLE
*
```

12.1.3 Changed IFCIDs

Db2 12 introduced the following changes to existing IFCIDs:

- Application compatibility IFCID changes
- Dynamic SQL plan stability IFCID changes
- Insert Algorithm 2 IFCID changes
- Lift partition limits IFCID changes
- Large object compression IFCID changes
- Transfer ownership IFCID changes
- UNLOAD utility IFCID changes
- Other changed IFCIDs

Application compatibility IFCID changes

At times, Db2 might need to deliver changes that can influence how your applications run. This change is called an *incompatible change*.

If Db2 changes its behavior, you can collect trace information to identify which applications might be affected. Before Db2 12, IFCID 366 and IFCID 376 record such changes in your applications. Starting in Db2 12, IFCID 376 records this information and IFCID 366 is deprecated.

IFCID 366: Deprecated application compatibility trace

IFCID 366 is deprecated in Db2 12. Db2 does not accept trace commands for IFCID 366 and it does not write out this trace record. IFCID 376 should be used to identify incompatibilities.

IFCID 376: Application compatibility trace

Constants that were used in IFCID 366 were moved to IFCID 376. The constants indicate to which incompatible changes your application is made available.

The updated IFCID 376 that includes constants that indicate incompatibility types is shown in Example 12-7.

Example 12-7   Updated IFCID 376

```
QW0376       DSECT
QW0376FN     DS F          Incompatible change indicator
*...........................QW0376FN CONSTANTS..................
C_QW0376_CHAR     EQU 0001   V9 SYSIBM.CHAR(decimal-expr)
*                            function
C_QW0376_VCHAR    EQU 0002   V9 SYSIBM.VARCHAR(decimal-expr)
*                            function
*                            CAST (decimal as VARCHAR or CHAR)
C_QW0376_TMS      EQU 0003   Unsupported character string
*                            representation of a timestamp
C_QW0376_IMPCAST  EQU 0007   Use the pre-v10 server compatibility
*                            behavior which is not to implicitly
*                            cast input host variables during
*                            server host bind-in processing
C_QW0376_SPPARMS  EQU 0008   data types of the returned output
*                            data match the data types of the
*                            corresponding CALL statement
*                            arguments
C_QW0376_IGNORETZ EQU 0009   Use the pre-V10 server compatibility
*                            behavior which is to ignore time
*                            zone information when bind in TMSTZ
*                            hostvar to TMS target
C_QW0376_TRIM     EQU 0010   V9 RTRIM, LTRIM and STRIP functions
C_QW0376_SETOPINT EQU 0011   Set operator with SELECT INTO
C_QW0376_XMLINS   EQU 1101   Insert into an XML column without
*                            XMLDOCUMENT function
C_QW0376_XPATHERR EQU 1102   XPath evaluation error
C_QW0376_RLF      EQU 1103   RLF governing
C_QW0376_CLIENTAC EQU 1104   Long CLIENT_ACCTNG Special Reg value
C_QW0376_CLIENTAP EQU 1105   Long CLIENT_APPLNAME Special Reg
*                            value
C_QW0376_CLIENTUS EQU 1106   Long CLIENT_USERID Special Reg value
C_QW0376_CLIENTWK EQU 1107   Long CLIENT_WRKSTNNAME Special Reg
*                            value
C_QW0376_CLIENTSR EQU 1108   Long client Special Reg value for
*                            RLF
C_QW0376_TMSCAST  EQU 1109   CAST(string AS TIMESTAMP)
C_QW0376_SPACEINT EQU 1110   SPACE integer argument greater than
*                            32764
C_QW0376_VCHARINT EQU 1111   VARCHAR int argument greater than
*                            32764
C_QW0376_XMLEMPT  EQU 1112   XML_RESTRICT_EMPTY_TAG ZPARM is used
*                            and empty XML element is serialized
*                            to <X></X>
*...............................................................
```

Dynamic SQL plan stability IFCID changes

The following IFCIDs were changed in support of the dynamic SQL plan stability enhancement:

- IFCID 002: RDS statistics block
- IFCID 002: EDM pool statistics block
- IFCID 021: Lock types
- IFCID 029: EDM request begin identifier and new block
- IFCID 030: EDM request end identifier and new block
- IFCID 106: New subsystem parameter
- IFCID 316: Stabilization and hash ID information

IFCID 002: RDS statistics block

The QXSTSFND field was added to the RDS statistics block (QXST) to record the number of times a PREPARE request was satisfied by using a stabilized copy that is stored in the catalog. A snippet of IFCID 002 that shows the new field is shown in Example 12-8.

Example 12-8 Changed IFCID 002 QXST block new QXSTSFND field

```
QXSTSFND     DS  D    # of times a PREPARE request was satisfied+
*                     by making a copy from the stabilized      +
*                     statement in SYSIBM.SYSDYNQRY catalog     +
*                     table. The stabilized statement search is +
*                     done only when no matching statement was  +
*                     found in the prepared statement cache.    +
```

IFCID 002: EDM pool statistics block

Several fields were added to the EDM pool statistics block (QISE) to record the number of requests, lookups, and matches for stabilized dynamic SQL statements, as shown in Example 12-9.

Example 12-9 Changed IFCID 002 QISE block new fields

```
QISEDPSL DS    D   /* # of requests to look for DPS       */
QISEDPSC DS    D   /* # of times possible row found       */
QISEDPSM DS    D   /* # of times match thrgh text/bind opt*/
QISEDPSF DS    D   /* # of times match found              */
```

IFCID 021: Lock types

Four lock types were added to IFCID 21 to record concurrent access control for a stabilized query in the SYSDYNQRY table. One new lock type is associated with IM’s fast traverse block.

The new locks types that were added to IFCID 021 for dynamic SQL plan stability and IM’s fast traverse block are shown in Example 12-10.

Example 12-10 Changed IFCID 021

```
QW0021HI EQU   X'42'    *       SYSDYNQRY HASH_ID lock         *
QW0021SG EQU   X'43'    *       SYSDYNQRY STBLGRP lock         *
QW0021OD EQU   X'47'    *       SYSDYNQRY object dep lock      *
QW0021AD EQU   X'48'    *       SYSDYNQRY auth ID dep lock     *
QW0021IF EQU   X'49'    *       IM's fast traverse block lock  *
```

IFCID 029: EDM request begin identifier and new block

A new identifier (ID) value, DY, was added to IFCID 029 to indicate that the object is a table for dynamic SQL plan stability. Also, a section was added to the trace record specifically for dynamic SQL plan stability, as shown in Example 12-11.

Example 12-11 Changed IFCID 029

```
QW0029ID DS    CL2    DB=DBDID, CT=CURSOR TABLE, PT=PACKAGE TABLE
*                     DY=DPS TABLE
. . .
         ORG   QW0029DB   DY DPS TABLE MAPPING FOLLOWS
QW0029SC DS    CL18     %U SCHEMA SHORT NAME
*                         Truncated if QW0029SC_Off¬=0
QW0029QH DS    CL16     HASHID
QW0029CP DS    H        COPY ID
QW0029QD DS    XL8      SDQE_STMTID
QW0029QC DS    XL4      RESERVED
QW0029SC_Off   DS H (FIXED 15)
QW0029RB DS    CL1      release bound
QW0029FL DS    XL1      RESERVED
. . .
*
QW0029SC_D   Dsect        Use if QW0029SC_Off¬=0
QW0029SC_Len DS    H      Length of the following field
QW0029SC_Var DS    0CL128 %U SCHEMA
```

IFCID 030: EDM request end identifier and new block

As with IFCID 029, a new identifier (ID) value, DY, was added to the IFCID 030 to indicate that the object is a table for dynamic SQL plan stability. Also, a section was added specifically for dynamic SQL plan stability, as shown in Example 12-12.

Example 12-12 Changed IFCID 030

```
QW0030ID DS    CL2    DB=DBDID, CT=CURSOR TABLE, PT=PACKAGE TABLE
*                     DY=DPS TABLE
. . .
         ORG   QW0030DB   DY DPS TABLE MAPPING FOLLOWS
QW0030SC DS    CL18     %U SCHEMA SHORT NAME
*                         Truncated if QW0030SC_Off¬=0
QW0030QH DS    CL16     HASHID
QW0030CP DS    H        COPY ID
QW0030QD DS    XL8      SDQE_STMTID
QW0030QC DS    XL4      Number of records read
QW0030SC_Off   DS H (FIXED 15)
QW0030RB DS    CL1      release bound
QW0030FL DS    XL1      FLAG
. . .
QW0030SC_D   Dsect        Use if QW0030C_Off¬=0
QW0030SC_Len DS    H      Length of the following field
QW0030SC_Var DS    0CL128 %U SCHEMA
```

IFCID 106: New subsystem parameter

The QWP4CDST field was added to trace the internal setting of the new subsystem parameter CACHEDYN_STABILIZATION, as shown in Example 12-13.

Example 12-13 Changed IFCID 106

```
QWP4CDST DS    CL1       CACHEDYN_STABILIZATION          s17830
*                          B = BOTH                      s17830
*                          C = CAPTURE                   s17830
*                          L = LOAD                      s17830
*                          N = NONE                      s17830
```

IFCID 316: Stabilization and hash ID information

Several fields were added to the IFCID 316 to include hash ID and version information for the stabilized statement and the stabilization group name, as shown in Example 12-14.

Example 12-14 Changed IFCID 316

```
QW0316_SDQ_STMTID     DS   XL8   Stabilized statement ID
QW0316_QUERY_HASH_ID  DS   CL16  Query's hash ID
QW0316_QUERY_HASH_VER DS   F     Version of query's hash ID
QW0316_STBLGRP_Off    DS   H     Offset from QW0316 to
*                                stabilization group
. . .
QW0316_STBLGRP_D    Dsect         Use if QW0316STBLGRP_Off¬=0
QW0316_STBLGRP_Len  DS    H       Length of the next field
QW0316_STBLGRP_Var  DS   0CL128  %U Stabilization group name
```

Insert Algorithm 2 IFCID changes

Instrumentation was added to track the time that is spent in waiting for formatting pages in Db2. Tracking is done at package, statement, and plan level accounting. The following IFCIDs were changed in support of the Insert Algorithm 2 enhancement:

- IFCID 002: Data Manager statistics block
- IFCID 003: Accounting control block
- IFCID 018: End of inserts and scans
- IFCID 058: Accumulated pipe wait time
- IFCID 106: New subsystem parameter
- IFCID 239: Package level pipe wait information
- IFCID 316: Statement level pipe wait information
- IFCID 401: Accumulated pipe wait time

Note: APAR PI81731, which was still open at the time of this writing, introduces more fields in IFCID 1, 2, 3, and 148 that are related to the use of Insert Algorithm 2.

IFCID 002: Data Manager statistics block

Two new counts, QISTINPA and QISTINPD, were added to the Data Manager Statistics Block (DSNDQIST) of IFCID 002 to record Insert Algorithm 2 pipe allocation and disable counts since Db2 restart, as shown in Example 12-15.

Example 12-15 New fields QISTINPA and QISTINPD of IFCID 002

```
QISTINPA     DS D          /* Number of DM Fast Insert   @fi1*/
*                          /* (Insert Algorithm Level 2) @fi1*/
*                          /* pipes allocated since DB2  @fi1*/
*                          /* restart.                   @fi1*/
QISTINPD     DS D          /* Number of DM Fast Insert   @fi1*/
*                          /* (Insert Algorithm Level 2) @fi1*/
*                          /* pipes disabled since DB2   @fi1*/
*                          /* restart.                   @fi1*/
```

IFCID 003: Accounting control block

Two new fields, QWAX_PIPE_WAIT and QWAX_PIPEWAIT_COUNT, were added to the accounting control block (QWAX) to indicate pipe wait information, as shown in Example 12-16.

Example 12-16 Changed IFCID 003

```
QWAX_PIPE_WAIT DS   CL8    /* Accumulated wait time for pipe
*                             wait                           */
*
QWAX_PIPEWAIT_COUNT DS   F /* Number of wait trace events
*                             processed for pipe wait        */
```

IFCID 018: End of inserts and scans

Four new counts were added to the IFCID 018 to record with various Insert Algorithm 2 processing information, as shown in Example 12-17.

Example 12-17 Changed IFCID 018

```
QW0018FI DS    XL8          ROWS INSERTED VIA FAST INSERT
*
QW0018FS DS    XL8          ROWS COULD NOT USE FAST INSERT
*
QW0018FA DS    XL8          NBR TIMES FAST INSERT PIPE REFILLED
*
QW0018FW DS    XL8          NBR TIMES DB2 WAITED FOR FAST INSERT
```

IFCID 058: Accumulated pipe wait time

A new timer, QW0058PW, was added to IFCID 058 to record the accumulated Insert Algorithm 2 pipe wait time, as shown in Example 12-18.

Example 12-18 New QW0058PW field of IFCID 058

```
QW0058PW DS  CL8          Accumulated wait time for pipe
```

IFCID 106: New subsystem parameter

The QWP4DINA field was added to trace the internal setting of the new subsystem parameter DEFAULT_INSERT_ALGORITHM, as shown in Example 12-19.

Example 12-19 New QWP4DINA field of IFCID 106

```
QWP4DINA DS    H         DEFAULT_INSERT_ALGORITHM
```

IFCID 239: Package level pipe wait information

Two new fields, QPAC_PIPE_WAIT and QPAC_PIPEWAIT_COUNT, were added to the accounting control block (QPAC) to indicate pipe wait information for the package, as shown in Example 12-20.

Example 12-20 New fields QPAC_PIPE_WAIT and QPAC_PIPEWAIT_COUNT of IFCID 239

```
QPAC_PIPE_WAIT DS      XL8 /* accumulated wait time for a pipe */
*                          /* while executing this package     */
QPAC_PIPEWAIT_COUNT DS F   /* number of wait trace events      */
*                          /* processed for waits for a pipe   */
*                          /* while executing this package     */
```

IFCID 316: Statement level pipe wait information

A new field, QW0316_PIPE_WAIT, was added to IFCID 316 to record the Insert Algorithm 2 pipe wait for a statement, as shown in Example 12-21.

Example 12-21 New field QW0316_PIPE_WAIT of IFCID 316

```
QW0316_PIPE_WAIT DS  wait time for pipe wait
```

IFCID 401: Accumulated pipe wait time

A new field, QW0401WH, was added to IFCID 401 to record the accumulated pipe wait time, as shown in Example 12-22.

Example 12-22 New QW0401WH field of IFCID 401

```
QW0401WH      DS   CL8  Accumulated wait for pipe wait
```

IFCID changes that are related to PBR RPN table spaces

Before Db2 12, absolute page numbers were used in certain IFCID trace records. With the enhancements to lift partition limits, new fields are added to store the 6-byte page numbers and new subsystem parameter values. The following IFCIDs were changed in support of the enhancement to lift partition limits:

- IFCID 006: Pre-read page number flag and partition number
- IFCID 007: Post-read page number flag and partition number
- IFCID 021: Resource name
- IFCID 106: New subsystem parameter
- IFCID 124: Page number within page set
- IFCID 127: Agent suspend
- IFCID 128: Agent resume
- IFCID 150: Resource name
- IFCID 172: Resource name
- IFCID 196: Resource name
- IFCID 198: Page numbering flag

- IFCID 223: Identifier, new constant, and partition number
- IFCID 226: Page numbering flag
- IFCID 227: Page numbering flag
- IFCID 255: Partition number and relative page number
- IFCID 259: Partition number and relative page number
- IFCID 305: Table space partition number and type

IFCID 006: Pre-read page number flag and partition number

IFCID 006 was modified to include a flag that indicates whether the page number was relative or absolute and to record the partition number, when applicable. The new flag and partition number field in IFCID 006 is shown in Example 12-23.

Example 12-23 Changed IFCID 006

```
QW0006P   DS    X           FLAGS
QW0006P1 EQU    X'80'       1 = Relative page number in QW0006PG
*                           0 = Absolute page number in QW0006PG
         DS     CL2         unused
QW0006PT DS     F           Partition number or 0 if non-
*                           partitioned
```

IFCID 007: Post-read page number flag and partition number

As with IFCID 006, IFCID 007 was modified to include a flag that indicates whether the page number was relative or absolute and to record the partition number, when applicable. The new flag and partition number field in IFCID 007 is shown in Example 12-24.

Example 12-24 Changed IFCID 007

```
QW0006P   DS    X           FLAGS
QW0006P1 EQU    X'80'       1 = Relative page number in QW0006PG
*                           0 = Absolute page number in QW0006PG
         DS     CL2         unused
QW0006PT DS     F           Partition number or 0 if non-
*                           partitioned
```

IFCID 021: Resource name

The resource name in IFCID 021 was changed to indicate resources that uses relative page numbers, as shown in Example 12-25.

Example 12-25 Changed IFCID 021

```
         ORG    QW0021KR
QW0021KE DS     0CL7    *  ID of small resource when QW0021KL=16 *
QW0021KF DS     XL2     *  partition number                      *
QW0021KG DS     CL4     *  page number                           *
QW0021KH DS     XL1     *  record id within page                 *
         DS     CL17    * insure that offset of QW0021FC is      *
*                       * 24 bytes from QW0021KR                 *
```

IFCID 106: New subsystem parameter

The QWP4PSPN field was added to trace the internal setting of the new subsystem parameter PAGESET_PAGENUM, as shown in Example 12-26.

Example 12-26 New QWP4PSPN field of IFCID 106

```
QWP4PSPN DS    CL1       PAGESET_PAGENUM:
*                          A=ABSOLUTE, R=RELATIVE
```

IFCID 124: Page number within the page set

IFCID 124 was modified to include the page number within the page set, QW01244N, as shown in Example 12-27.

Example 12-27 New QW0124N field of IFCID 124

```
QW01244N DS    XL6            ! PAGE NUMBER WITHIN PAGESET
*                             !   For RPN obj=2 bytes of part#
*                             !               4 bytes of page#
*                             !   For Non-RPN obj= ignore first
*                             !     2 bytes, next 4 bytes are
*                             !     absolute page number
         DS    CL2            ! unused
```

IFCID 127: Agent suspend because I/O is in progress by another agent

As with IFCID 006 and 007, IFCID 127 was modified to include a flag that indicates whether the page number is relative or absolute and to include the partition number, when applicable. The new flag and partition number field in IFCID 127 is shown in Example 12-28.

Example 12-28 Changed IFCID 127

```
QW0127P  DS    X             Flags
QW0127P1 EQU   X'80'         1 = Relative page number in QW0127PG
*                            0 = Absolute page number in QW0127PG
         DS    CL2           unused
QW0127PT DS    F             Part number or 0 if non-partitioned
         DS    CL4           unused
```

IFCID 128: Agent resume after other agent I/O ends

As with IFCID 127, IFCID 128 was modified to include a flag that indicates whether the page number was relative or absolute and to include the partition number, when applicable. IFCID 128 also records the page number.

The new flag and partition number field in IFCID 128 is shown in Example 12-29.

Example 12-29 Changed IFCID 128

```
QW0128P  DS    X           Flags
QW0128P1 EQU   X'80'       1 = Relative page number in QW0128PG
*                          0 = Absolute page number in QW0128PG
         DS    CL1         unused
. . .
QW0128PG DS    F           PAGE NUMBER
*                          Based on QW0128P1,either absolute or
*                          relative page number is stored here,
*                          partition# can be found in QW0128PT
QW0128PT DS    F           Part number or 0 if non-partitioned
*   (S) = FOR SERVICEABILITY
```

IFCID 150: Resource name

As with IFCID 021, IFCID 150 was updated to indicate the resource name that uses relative page numbers, as shown in Example 12-30.

Example 12-30 Changed IFCID 150

```
         ORG   QW0150KR *
QW0150KE DS    OCL7     *  ID of small resource when QW0150KL=16
QW0150KF DS    XL2      *  partition number
QW0150KG DS    CL4      *  page number
QW0150KH DS    XL1      *  record id within page
```

IFCID 172: Resource name

As with IFCID 021 and 150, IFCID 172 was updated to indicate the resource name that uses relative page numbers, as shown in Example 12-31.

Example 12-31 Changed IFCID 172

```
         ORG   QW0172KR *
QW0172KE DS    OCL7     *  ID of small resource when QW0172RL=16
QW0172KF DS    XL2      *  partition number
QW0172KG DS    CL4      *  page number
QW0172KH DS    XL1      *  record id within page
         DS    CL17       REST OF SPACE FOR 28 BYTE RESOURCE NAME
```

IFCID 196: Resource name

IFCID 196 was updated to indicate the resource name that uses relative page numbers, as shown in Example 12-32.

Example 12-32 Changed IFCID 196

```
         ORG   QW0196KR *
QW0196KE DS    0CL7     *  ID of small resource when QW0196RL=16
QW0196KF DS    XL2      *  partition number
QW0196KG DS    CL4      *  page number
QW0196KH DS    XL1      *  record id within page
         DS    CL17       REST OF SPACE FOR 28 BYTE RESOURCE NAME
```

IFCID 198: Page numbering flag

IFCID 198 was modified to include a flag that indicates whether the page number is relative or absolute, and to include the partition number, when applicable. The new fields in IFCID 198 are shown in Example 12-33.

Example 12-33 Changed IFCID 198

```
QW0198PN DS    F         PAGE NUMBER
*                         Based on QW0198P1, either absolute or
*                         relative page number is stored here,
*                         the partition# can be found in QW0198PT
. . .
QW0198P  DS    X         Flags
QW0198P1 EQU   X'80'     1 = Relative page number in QW0198PN
*                        0 = Absolute page number in QW0198PN
         DS    XL2       unused
QW0198PT DS    F         Partition number or 0 if non-
*                  partitioned
```

IFCID 223: Identifier, new constant, and partition number

IFCID 223 was modified to include a resource identifier, a new constant that indicates a partitioned table space that uses relative page numbers, and a new field to record the partition number. The new fields in IFCID 223 are shown in Example 12-34.

Example 12-34 Changed IFCID 223

```
QW0223KY DS    0CL5     ID of small resource when QW0223TY='L' or
*                        'R'
. . .
QW0223TR EQU   C'R'       PBR UTS that uses relative page numbers
QW0223PT DS    XL2        1-based partition number if partitioned
*   (S) = FOR SERVICEABILITY
```

IFCID 226: Page numbering flag

IFCID 226 was modified to include a flag that indicates whether the page number was relative or absolute, partition number, and page number. The new fields in IFCID 226 are shown in Example 12-35.

Example 12-35 Changed IFCID 226

```
QW0226PG DS    F         PAGE NUMBER TO READ/WRITE
*                         Based on QW0226P1,either absolute or
*                         relative page number is stored here,
*                         partition# can be found in QW0226PT
. . .
QW0226P  DS    X         Flags
QW0226P1 EQU   X'80'     1 = Relative page number in QW0226PG
*                        0 = Absolute page number in QW0226PG
         DS    XL2       unused
QW0226PT DS    F         Partition number or 0 if non-partitioned
*   (S) = FOR SERVICEABILITY
```

IFCID 227: Page numbering flag

As with IFCID 226, IFCID 227 was modified to include a flag that indicates whether the page number was relative or absolute, partition number, and page number. IFCID 226 and 227 are used to trace buffer manager page latch contention.

The new fields in IFCID 227 are shown in Example 12-36.

Example 12-36 Changed IFCID 227

```
QW0227PG DS    F         PAGE NUMBER TO READ/WRITE
*                          Based on QW0227P1,either absolute or
*                          relative page number is stored here,
*                          partition# can be found in QW0227PT
. . .
QW0227P  DS    X         Flags
QW0227P1 EQU   X'80'     1 = Relative page number in QW0227PG
*                        0 = Absolute page number in QW0227PG
         DS    XL2       unused
QW0227PT DS    F         Partition number or 0 if non-partitioned
*   (S) = FOR SERVICEABILITY
```

IFCID 255: Partition number and relative page number

IFCID 255 now features two new fields for the partition number and relative page number, as shown in Example 12-37.

Example 12-37 Changed IFCID 255

```
QW0255P1 DS    CL2       2-byte Pageset piece/partition number
QW0255P2 DS    CL4       4-byte Relative page# (within the piece)
```

IFCID 259: Partition number and relative page number

As with IFCID 255, IFCID 259 features two new fields for the partition number and relative page number, as shown in Example 12-38.

Example 12-38 Changed IFCID 259

```
QW0259K4 DS    CL4     4-byte Relative page number for RPN object
*                       For QW0259G1, relative pg# is stored here
         DS    CL16         16 BYTES OF ZEROS
. . .
         ORG   QW0259K4
QW0259KQ DS    CL3        Relative page number for non-RPN object
         DS    CL1        reserved
         DS    CL16       16 bytes of zeros
```

IFCID 305: Table space partition number and type

IFCID 305 was modified to record the partition number for a table space that uses relative page numbers in addition to introducing a new table space type value. The new fields in IFCID 305 are shown in Example 12-39.

Example 12-39 Changed IFCID 305

```
QW0305PT DS    XL2         Partition number.
*                           Valid when QW0305TY='R'
. . .
QW0305TY DS    CL1          TABLE SPACE TYPE
*                            POSSIBLE VALUES ARE :
*                             'N' - NON LARGE TABLE SPACE
*                             'L' - NON-EA 5-BYTE RID TABLE SPACE
*                             'V' - EA 5-BYTE RID TABLE SPACE
*                             'R' - PBR UTS that uses relative
*                                   page numbers
```

Large object compression IFCID changes

Db2 12 introduces LOB compression capabilities with the new zEnterprise Data Compression (zEDC) hardware. IFCID 003 and IFCID 106 were modified to trace LOB compression wait times and the number of waits. A new subsystem parameter also was introduced.

IFCID 003: Accounting control block

Two new fields, QWAX_LOBCOMP_WAIT and QWAX_LOBCOMP_COUNT, were added to the accounting control block (QWAX) to indicate the accumulated wait time and the number of wait trace events processed for LOB compression. No equivalent information at the package level (IFCID 0239) is available.

The new fields in IFCID 003 is shown in Example 12-40.

Example 12-40 New fields for LOB compression in IFCID 003

```
QWAX_LOBCOMP_WAIT DS   CL8 /* Accumulated wait time for LOB
*                             compression                      */
QWAX_LOBCOMP_COUNT DS    F /* Number of wait trace events
*                             processed for LOB compression    */
```

IFCID 106: New subsystem parameter

The QWP4CDRL field was added to trace the setting of the new subsystem parameter COMPRESS_DIRLOB, as shown in Example 12-41.

Example 12-41 Changed IFCID 106

```
QWP4MISD DS    X
*        EQU   X'80'     Not available
QWP4CDRL EQU   X'40'     COMPRESS_DIRLOB
```

Transfer ownership IFCID changes

Db2 12 provided support with which you can change the ownership of certain database objects by using a new TRANSFER OWNERSHIP SQL statement. The following IFCIDs were modified to support this functionality:

- IFCID 002: RDS statistics block
- IFCID 062: Statement type
- IFCID 140: Source object owner and name
- IFCID 361: Source object owner and name

IFCID 002/003/148: RDS statistics block

The QXTRNOWN field was added to the RDS statistics block (QXST) to indicate the number of times ownership was transferred.

A snippet of the QXST section in IFCID 002/003/148 that shows the new field is shown in Example 12-42.

Example 12-42 Changed IFCID 002/003/148

```
QXTRNOWN     DS  D          # of TRANSFER OWNERSHIP
```

IFCID 062: Statement type

A new statement type of X'AB' was added to indicate the statement was a TRANSFER OWNERSHIP statement in IFCID 062, as shown in Example 12-43.

Example 12-43 Changed IFCID 062

```
QW0062TO EQU   X'AB'         TRANSFER OWNERSHIP
```

IFCID 140: Source object owner and name

The QW0140SC and QW0140SN fields were added in IFCID 140 to record the source object owner and name that were updated, as shown in Example 12-44.

Example 12-44 Changed IFCID 140

```
QW0140SC DS    CL8     %U SOURCE OBJECT OWNER:  Three cases
*                       1) If object type not equal User Auth -
*                         Qualifier of the object against
*                         which authorization was checked.
*                         Valid for qualifiable objects.
*                       2) If object type equals User Auth -
*                         Qualifier of ALIAS being created.
*                         Valid for CREATE ALIAS privilege.
*                       3) If object type equals User Auth -
*                         Qualifier of the object being
*                         transferred. Valid for TRANSFER
*                         OWNERSHIP statement.
*                       Truncated if QW0140SC_Off¬=0
*
QW0140SN DS    CL18    %U SOURCE OBJECT NAME:  Three cases
*                       1) If object type not equal User Auth -
*                         Name of the object against which
*                         authorization was checked.
*                       2) If object type equals User Auth -
*                         Name of the object being created. Valid
*                         when privilege is CREATE ALIAS,
*                         CREATEDBA, CREATEDBC, or CREATE
*                         STOGROUP.
*                       3) If object type equals User Auth -
*                         Name of the object being transferred.
*                         Valid for TRANSFER OWNERSHIP statement.
*                       Truncated if QW0140SN_Off¬=0
*
```

IFCID 361: Source object owner and name

The descriptions for the QW0361SC_Var and QW0361SN_Var fields for the source object owner and name were updated in IFCID 361, as shown in Example 12-45 . IFCID 361 is used to audit the use of administrative authorities.

Example 12-45 Changed IFCID 361

```
QW0361SC_Var DS  0CL128   %U SOURCE OBJECT QUALIFIER/OWNER
*                         If object type equals User Auth -
*                         Qualifier of the object being
*                         transferred. Valid for TRANSFER
*                         OWNERSHIP statement.
*
. . .
QW0361SN_Var DS  0CL128   %U Source object Name
*                         If object type equals User Auth -
*                         Name of the object being transferred.
*                         Valid for TRANSFER OWNERSHIP statement.
*
```

UNLOAD privilege for UNLOAD utility IFCID changes

Depending on the value of the new subsystem parameter, AUTH_COMPATIBILITY, a new UNLOAD privilege might be required when the UNLOAD utility is used. IFCID 404 was introduced as a serviceability trace to identify the use of the SELECT privilege for UNLOAD utility access, as described in 12.1.2, “New IFCIDs” on page 343.

In addition to the new IFCID 404 record, IFCID 106 was modified to trace the value of the new AUTH_COMPATIBILITY subsystem parameter.

IFCID 106: New subsystem parameter

The QWP4AUTC field was added to trace the internal setting of the new subsystem parameter AUTH_COMPATIBILITY. The QWP4AUTCSU indicates whether the SELECT_FOR_UNLOAD option of AUTH_COMPATIBILITY was specified, as shown in Example 12-46.

Example 12-46 Changed IFCID 106

```
QWP4AUTC DS    XL1       AUTH_COMPATIBILITY               s20166
QWP4AUTCSU EQU X'80'     - SELECT_FOR_UNLOAD              s20166
```

Other IFCID changes

In addition to the IFCID changes that were made, Db2 12 introduces changes to the following IFCIDs:

- IFCID 001: New fields in the statistics record command data section that are related to new Db2 commands in Db2 12.
- IFCID 002: New fields that are related to contiguous buffer pools within the statistics record buffer manager data section.
- IFCID 002: New fields were added to indicate the status of index FTB.
- APAR PI82539, still open at the time of this writing, removes QISEDFRE, QISEKFRE, and QISECFRE fields from the EDM pool statistics sections (QISE) in IFCID 002. With the Db2 12 changes to EDM storage management, these fields are no longer used.
- IFCID 106: New subsystem parameters.
- IFCID 125: Ddaptive index processing.
- IFCID 003: Work file usage fields at thread level in QWAC.
- IFCID 199: New fields in the Dataset I/O Statistics record for synchronous and asynchronous I/O delay times in microseconds.
- IFCID 053/058: New field and section number added.
- IFCID 316/401/058: New fields added for more granular wait times.
- Correlation Header (QWHSHC): Added batch job step field.

IFCID 001: Command Data Section Statistics

The following fields were added to command data section (Q9ST) in IFCID 001:

- Q9STCTEN
- Q9STCTDQ
- Q9STCTSQ
- Q9STCTXQ

These new fields accumulate the statistics that are related to number of times the newly introduced commands in Db2 12 are executed. The new fields that were added to IFCID 001 and their descriptions are shown in Example 12-47.

Example 12-47 IFCID 001 additions

```
Q9STCTEN DS    F  NUMBER OF "-ACTIVATE FUNCTION LEVEL" COMMANDS.
Q9STCTDQ DS    F  NUMBER OF "-DISPLAY DYNQUERYCAPTURE" COMMANDS.
Q9STCTSQ DS    F  NUMBER OF "-START DYNQUERYCAPTURE" COMMANDS.
Q9STCTXQ DS    F  NUMBER OF "-STOP DYNQUERYCAPTURE" COMMANDS.
```

IFCID 002: Buffer manager data statistics: Contiguous buffer pools

The following fields were added to IFCID 002:

- QBSTAGET
- QBSTASGE
- QBSTASYN
- QBSTASSE

These fields accumulate the statistics that are related to contiguous buffer pool overflow getpage requests. The fields that were added to IFCID 002 and their descriptions are shown in Example 12-48.

Example 12-48 IFCID 002 additions

```
QBSTAGET FIXED(64),       /*OVERFLOW total random getpages*/
QBSTASGE FIXED(64),       /*OVERFLOW total sequential     */
                          /*  getpages                    */
QBSTASYN FIXED(64),       /*OVERFLOW total sync read I/Os */
                          /*  for random getpages         */
QBSTASSE FIXED(64);       /*OVERFLOW total sync read I/Os */
                          /*  for sequential getpages     */
```

IFCID 002: Data Manager Statistics - FTB

The following fields were added to the data manager section in IFCID 002:

- QISTTRAVMIN
- QISTFTBCANT
- QISTFTBCAN
- QISTFTBSIZE
- QISTFTBNUMP
- QISTFTBNUMC

These fields, as shown in Example 12-49, indicate the status of Index FTBs for the member.

Example 12-49 IFCID 002 additions

```
QISTTRAVMIN  DS F       /* FTB threshold: minimum     @io1*/
*                       /* number of index traversals @io1*/
QISTFTBCANT  DS F       /* Total number of indexes    @io1*/
*                       /* which meet FTB criteria    @io1*/
QISTFTBCAN   DS F       /* Total number of indexes    @io1*/
*                       /* which meet FTB criteria    @io1*/
*                       /* and the traverse count is  @io1*/
*                       /* above the threshold        @io1*/
QISTFTBSIZE  DS F       /* total memory allocation    @io1*/
*                       /* for all FTBs for this      @io1*/
*                       /* member                     @io1*/
QISTFTBNUMP  DS F       /* Number of indexes for      @io1*/
*                       /* which FTB existed in the   @io1*/
*                       /* previous run of in-memory  @io1*/
*                       /* optimization               @io1*/
QISTFTBNUMC  DS F       /* Number of indexes for      @io1*/
*                       /* which FTB exists in the    @io1*/
*                       /* current run of in-memory   @io1*/
*                       /* optimization               @io1*/
*/*---------------------------------------------------@io1*/
```

IFCID 106: New subsystem parameters

IFCID 106 was modified to remove the following fields that are no longer used in Db2 12:

- The QWP4RIFS field, which was used to trace the internal setting of the REORG_IGNORE_FREESPACE subsystem parameter. However, that parameter was deprecated in Db2 10 and Db2 11 and is removed in Db2 12.
- Starting in Db2 12, the storage management of LOB and XML data is managed autonomically by Db2. Therefore, the following subsystem parameters are no longer recorded:
  - LOBVALA
  - LOBVALS
  - XMLVALA
  - XMLVALS

The following enhancements were added to the IFCID 106:

- Eight new field constants to trace internal settings of the following new subsystem parameters introduced for Resource Limit Facility (RLF) for static SQL enhancements:
  - Subsystem parameter RLFERRSTC: QWP1RLFFS, QWP1RLFUS, and QWP1RLFNS
  - Subsystem parameter RLFENABL: QWP1RLFDNEN and QWP1RLFSTEN
  - Subsystem parameter RLFERRDSTC: QWP9RLFRS, QWP9RLFNS, and QWP9RLFLS
- The field QWP1PFASY was added to trace the internal setting of the new subsystem parameter PROFILE_AUTOSTART.
- The field QWP4RSO was added to trace the internal settings of the new subsystem parameter RETRY_STOPPED_OBJECTS.
- The field QWP4MNSU was added to trace the internal setting of the MATERIALIZE_NODET_SQLTUDF subsystem parameter.

- The field QWP4ERTS was added to trace the internal settings of the new subsystem parameter RENAMETABLE.
- For online backup and recovery enhancements, the QWP4CYFR field is added for the COPY support that was added to FASTREP(REQ). Also, the following new subsystem parameters were introduced and their internal settings are being recorded:
  - QWP4BSACP for the ALTERNATE_CP parameter
  - QWP4UDBSG for the UTIL_DBBSG parameter
  - QWP4ULBSG for the UTIL_LGBSG parameter
  - QWP4UHMDH for the UTILS_HSM_MSGDS_HLQ parameter
- The QWP4DDLM field was added to trace the internal setting of the new subsystem parameter DDL_MATERIALIZATION.
- The QWPAPEERREC was added to trace the internal setting of the new subsystem parameter PEER_RECOVERY.
- The QWP4SFPR field was added to trace the internal setting of the new subsystem parameter STATFDBK_PROFILE.

Another field, QWP4TPTM, was introduced in Db2 12 and retrofitted to Db2 11 and Db2 10. The QWP4TPTM was added to trace the internal setting of the new subsystem parameter TEMPLATE_TIME.

The modified IFCID 106 for these items are shown in Example 12-50.

Example 12-50 Changed IFCID 106

```
QWP1RLFFS EQU  X'10'    IF 1 INDICATE NOLIMIT (STATIC)     S20000
QWP1RLFUS EQU  X'08'    IF 1 INDICATE NORUN (STATIC)       S20000
QWP1RLFDNEN EQU X'04'   1 when RLFENABLE=DYNAMIC or ALL    S20000
QWP1RLFSTEN EQU X'02'   1 when RLFENALBE=STATIC or ALL     S20000
. . .
QWP1PFASY EQU  X'10'    PROFILE_AUTOSTART=YES              dp1897
. . .
         DS    CL4      Do not reuse                       s22058
QWP1LVS  DS    F        (s)                                s22058
. . .
         DS    CL4      Do not reuse                       s22058
QWP1XVS  DS    F        (s)                                s22058
. . .
QWP1RLFNS DS   F        RLF static limit in SU's           S20000
. . .
QWP4MS4D DS    X
*              X'80'     Not available                      s2593
. . .
QWP4RSO   EQU  X'01'     RETRY_STOPPED_OBJECTS             DM1851
. . .
QWP4MNSU EQU   X'80'     MATERIALIZE_NODET_SQLTUDF         DM1912
. . .
QWP4ERTS EQU   X'40'     RENAMETABLE                       DN1840
*                          '0' = DISALLOW_DEP_VIEW_SQLTUDF DN1840
*                          '1' = ALLOW_DEP_VIEW_SQLTUDF    DN1840
. . .
QWP4BSACP  DS  CL16      ALTERNATE_CP                      s12997
QWP4UDBSG  DS  CL8       UTIL_DBBSG                        s12997
```

```
QWP4ULBSG  DS  CL8       UTIL_LGBSG                        s12997
QWP4CYFR DS    CL1       COPY_FASTREPLICATION              s23187
QWP4DDLM DS    CL1       DDL_MATERIALIZATION               n22751
*                          I=ALWAYS_IMMEDIATE              n22751
*                          P=ALWAYS_PENDING                n22751
. . .
QWP4UHMDH DS   CL8       UTILS_HSM_MSGDS_HLQ               s17863
. . .
QWP4SFPR EQU   X'20'     STATFDBK_PROFILE                  s24345
. . .
QWP9RLFLS EQU  X'20'     If 1, indicate NOLIMIT (static)   S20000
QWP9RLFRS EQU  X'10'     If 1, indicate NORUN (static)     S20000
. . .
QWP9RLFNS DS   F         RLF static limit in SU's          s20000
. . .
QWPAPEERREC  DS CL1      PEER_RECOVERY                     DP1857
*                        'N' = NONE                        DP1857
*                        'R' = RECOVER                     DP1857
*                        'A' = ASSIST                      DP1857
*                        'B' = BOTH                        DP1857
```

IFCID 125: Adaptive index processing

The following new fields were added to the IFCID 125:

- QW0125TI
- QW0125QI
- QW0125_TRSN
- QW0125_PRSN
- QW0125_ORSN
- QW0125_TRSN

These fields record the RID list processing statistics for adaptive index processing, and information about the adaptive index processing that was performed. The new fields are shown in Example 12-51.

Example 12-51 IFCID 125 new fields

```
QW0125TI DS    XL8     index probing estimate: total number of
*                      RIDs in the index (set to MAX BIGINT for
*                      full leg)
QW0125QI DS    XL8     index probing estimate: number of RIDs
*                      within the keyrange, adjusted for filter
*                      factor
QW0125_TRSN  DS    CL1 reason leg was terminated
*                      F: leg was marked 'full'
*                      T: leg with < 32 RIDs
QW0125_PRSN  DS    CL1 reason leg not probed
*                      A: all legs fetched all RIDs
*                      B: this leg fetched all RIDs (<1 RIDblock)
*                      E: probing failed
*                      F: leg was marked 'full'
*                      K: cannot probe - missing high/low key
*                      M: mix of 'R'/'I'/'U' entries does not get
*                         reordered
*                      O: APS indicated to not probe
*                      S: earlier leg of index AND-ing was
*                          'likely' very filtering
*                       V: leg 'likely' very filtering
QW0125_ORSN  DS    CL1 reason leg was reodered
*                       V: leg 'likely' very filtering
*                       P: probing
QW0125_FRSN  DS    CL1 reason leg was marked 'full'
*                       L: non-filtering LIKE
*                       R: range predicate (non-LIKE)
*                       P: if OR-ing and est. # of RIDs > 30%
*                          or if AND-ing and est. # of RIDs > 50%
*                          of the table
*                       M: if not reason 'P', but est. # of RIDs
*                          > 'ridlist logical limit'
*                       G: aggressive termination
*                       T: most selective leg of AND-ing
*                          with FF >= 35%
```

IFCID 002: New workfile usage fields in accounting macro QWAC

Two new fields, QWAC_WORKFILE_MAX and QWAC_WORKFILE_CURR, were added to the QWAC control block to provide thread level information about the work file use in the Db2 accounting record, IFCID 003.

The changes to the QWAC are shown in Example 12-52.

Example 12-52 Changed QWAC

```
QWAC_WORKFILE_MAX DS XL8  /* Maximum number of workfile blocks */
*                         /* being used by this agent at any   */
*                         /* given point in time (traditional  */
*                         /* workfile use, DGTT and DGTT       */
*                         /* indexes)                          */
QWAC_WORKFILE_CURR DS XL8 /* Current number of workfile blocks */
*                         /* being used by this agent          */
*                              /* (traditional workfile use, DGTT   */
*                              /* and DGTT indexes)                 */
```

IFCID 199: New fields in the data set I/O statistics record

New fields QW0199S1, QW0199S2, QW0199A1, and QW0199A2 were added to IFCID 199. With the state-of-the-art DASD, I/O can complete in less than a millisecond; therefore, these fields are added to report the data set I/O response time in microsecond units for better precision and accuracy.

The four new fields that were added to IFCID 199 to report I/O delay in microseconds are shown in Example 12-53. The I/O fields from older releases, which report I/O delays in milliseconds, still exist. However, users are encouraged to use the new Db2 12 fields.

Example 12-53 IFCID 199 new data set I/O fields

```
*  **....FOLLOWING 2 FIELDS MATCH DSNB455I MSG CONTENTS ....   **
*  **....SYNCHRONOUS I/O FOR WRITE AND READ IN MICROSECONDS ...**
QW0199S1 DS    CL8        AVERAGE I/O DELAY (IN MICROSECONDS)
QW0199S2 DS    CL8        MAXIMUM I/O DELAY (IN MICROSECONDS)
*  **....FOLLOWING 2 FIELDS MATCH DSNB456I MSG CONTENTS   .... **
*  **..ASYNCH. I/O FOR WRITE, READ AND CASTOUT IN MICROSEC.... **
QW0199A1 DS    CL8        AVERAGE I/O DELAY (IN MICROSECONDS)
QW0199A2 DS    CL8        MAXIMUM I/O DELAY (IN MICROSECONDS)
```

IFCID 053/058: New field section number added

New field QW0053SECTN was added to IFCID 053 and QW0058SECTN was added to IFCID 058. The section number was added to these IFCID records to facilitate identifying an execution statement back to its associated PREPARE statement.

The new field that was added to IFCID 053 to report the statement section number is shown in Example 12-54 on page 367.

Example 12-54 IFCID 053 new section number field

```
QW0053SECTN DS CL2           RDI SECTION NUMBER
```

The new field that was added to IFCID 058 to report the statement section number is shown in Example 12-55.

Example 12-55 IFCID 058 new section number field

```
QW0058SECTN DS CL2           RDI SECTION NUMBER
```

IFCID 316/401/058: New fields adding more granular wait times

New fields were added to IFCID 316, 401, and 058 to provide more granular wait times at the statement level. IFCID 316 represents dynamic statements and IFCID 401 represents static statements. Previously, these fields were included in the accounting record only; however, with this change, these fields are now captured and represented at the statement level.

The new fields that were added to IFCID 316 to report more granular dynamic statement level wait times are shown in Example 12-56.

Example 12-56 IFCID 316 new fields

```
qw0316WC CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global child l-locks   */
qw0316WD CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global other l-locks   */
qw0316WE CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global pageset p-locks */
qw0316WF CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global page p-locks    */
qw0316WG CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global other p-locks   */
QW0316_PQS_WAIT  DS CL8  Accumulated time waiting for
*                        parallel queries to synchronize
*                        between parent and child tasks
```

The new fields that were added to IFCID 401 to report more granular statement level wait times for static SQL are shown in Example 12-57.

Example 12-57 IFCID 401 new fields

```
qw0401WC CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global child l-locks   */
qw0401WD CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global other l-locks   */
qw0401WE CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global pageset p-locks */
qw0401WF CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global page p-locks    */
qw0401WG CHAR(8),    /*   Accumulated wait time due to  */
*                    /*   due to global other p-locks   */
QW0401WPQS    DS   CL8  Accumulated wait for parallel query sync
```

The new fields that were added to IFCID 058 to report more granular statement level wait times are shown in Example 12-58.

Example 12-58 IFCID 58 new fields

```
QW0058AWTK DS  CL8      Accum wait time for global child l-locks
QW0058AWTM DS  CL8      Accum wait time for global other l-locks
QW0058AWTN DS  CL8      Accum wait time for global
*                             Pageset/Partition l-locks
QW0058AWTO DS  CL8      Accum wait time for global page p-locks
QW0058AWTQ DS  CL8      Accum wait time for global other p-locks
*
QW0058PQ   DS  CL8      Accumulated time waiting for parallel
*                       queries to synchronize between
*                       parent and child tasks
```

Correlation Header (DSNDQWHC): New field to identify batch job step

The QWHCJOBSTEP field (as shown in Example 12-59) was added to the trace correlation header that is mapped by macro DSNDQWHC. This information makes it easier to identify which part of a batch job produced the particular trace record or records.

Example 12-59 Correlation Header new field to represent the batch job step

```
QWHCJOBSTEP  DS    CL8            /*  JOBSTEP NAME  */
```

12.2 OMPE accounting/statistics batch report enhancements

This section describes the enhancements of IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS V540 (OMPE) that are related to instrumentation changes that were introduced by Db2 12.

OMPE supports all of the changes to the individual trace records (IFCIDs) and they can be formatted by using the **RECTRACE** command. In this section, we focus on the enhancements to the Db2 accounting and statistics reports.

Note: For more information about the topics that are described in this section, see *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS V540 Report Reference*, SH12-7065.

OMPE accounting and statistics reports were enhanced to monitor the performance that is related to the following new Db2 12 features:

- Dynamic SQL plan stability
- Insert Algorithm 2
- LOB compression
- Contiguous buffer pools: PGSTEAL(NONE)

12.2.1 Dynamic SQL plan stability

Db2 12 introduces the capability to achieve access path stability that is comparable to static SQL statements for cached dynamic SQL statements. When you enable dynamic SQL plan stability, Db2 stores statement cache structures for specified dynamic SQL statements in the Db2 catalog.

Whenever a stabilized dynamic SQL statement is not present in the dynamic statement cache when issued, Db2 can load the statement cache structures from the Db2 catalog and avoid the full prepare operation. This change saves the CPU time to prepare the SQL statement, and ensures the same access path is used after the statement is loaded from the catalog.

The layout for the OMPE accounting report dynamic SQL statement section is shown in Example 12-60 (the new OMPE field is in bold).

Example 12-60 Accounting: Dynamic SQL statement section

```
DYNAMIC SQL STMT       AVERAGE    TOTAL
----------------------------------------
REOPTIMIZATION         0.00       0
NOT FOUND IN CACHE     0.00       0
FOUND IN CACHE         0.00       0
IMPLICIT PREPARES      7.00       7
PREPARES AVOIDED       0.00       0
CACHE_LIMIT_EXCEEDED   0.00       0
PREP_STMT_PURGED       0.00       0
STABILIZED PREPARE     0.00       0 (QXSTSFND)
CSWL - STMTS PARSED    0.00       0
CSWL - LITS REPLACED   0.00       0
CSWL - MATCHES FOUND   0.00       0
CSWL - DUPLS CREATED   0.00       0
```

The layout for the OMPE statistics report dynamic SQL statement section with the new and changed OMPE fields highlighted in bold is shown in Example 12-61.

Example 12-61 Statistics: Dynamic SQL statement section

```
DYNAMIC SQL STMT             QUANTITY   /SECOND    /THREAD  /COMMIT
-----------------------------------------------------------------------
PREPARE REQUESTS             210225     3503.74    N/C      328.99
    FULL PREPARES            42681      711.35     N/C      66.79
    SHORT PREPARES           154592     2576.53    N/C      241.93
SHORT PREPARES               154592     2576.53    N/C      241.93  (SPREPSHT)
    BASED ON CACHE           141640     2360.66    N/C      221.66  (SCACHSHT)
    BASED ON CATALOG         12952      215.87     N/C      20.27  (QISEDPSF)
LOOK-UP IN CATALOG           42685      711.41     N/C      66.80  (QISEDPSL)
CACHE HIT RATIO (%)          73.54      N/A        N/A      N/A  (SCACHHRA)
CACHE+CATALOG HIT RATIO (%)  79.70      N/A        N/A      N/A  (STOTCHRA)
TOTAL PREPARES               210232     3503.86    N/C      329.00 (SPREPSUM)
   EXPLICIT PREPARES         210232     3503.86    N/C      329.00  (QXPREP)
   IMPLICIT PREPARES         0          0.00       N/C      0.00
STABILIZED PREPARES          12948      215.80     N/C      20.26  (QXSTSFND)
PREPARES AVOIDED             0          0.00       N/C      0.00
CACHE LIMIT EXCEEDED         0          0.00       N/C      0.00
PREP STMT PURGED             0          0.00       N/C      0.00
LOCAL CACHE HIT RATIO (%)    N/C        N/A        N/A      N/A
CSWL - STMTS PARSED          0          0.00       N/C      0.00
CSWL - LITS REPLACED         0          0.00       N/C      0.00
CSWL - MATCHES FOUND         0          0.00       N/C      0.00
CSWL - DUPLS CREATED         0          0.00       N/C      0.00
```

The OMPE Statistics Db2 commands section was enhanced to report the number of dynamic plan stability-related administration commands that were run during reporting period. The new and changed fields are highlighted in bold in Example 12-62.

Example 12-62 Statistics: Db2 commands section

```
DB2 COMMANDS                 QUANTITY
--------------------------   ---------------
DISPLAY DATABASE             0.00
DISPLAY THREAD               0.00
DISPLAY UTILITY              0.00
      …..     …..     …..    ……
…..     …..     …..          ……
DISPLAY DYNQUERYCAPTURE      0.00 (Q9STCTDQ)
…..     …..     …..          ……
…..     …..     …..          ……
START PROFILE                0.00
START ACCEL                  0.00
START DYNQUERYCAPTURE        0.00 (Q9STCTSQ)
…..     …..     …..          ……
…..     …..     …..          ……
STOP PROFILE                 0.00
STOP ACCEL                   0.00
STOP DYNQUERYCAPTURE         0.00 (Q9STCTXQ)
```

Note: For more information about dynamic SQL plan stability, see *IBM DB2 12 for z/OS Technical Overview*, SG24-8383.

12.2.2 Insert Algorithm 2

Db2 12 introduces Insert Algorithm 2, which can improve the performance of INSERT operations for unclustered data. The use of this new insert algorithm can result in an increase in insert throughput, especially with data that is not indexed. Insert Algorithm 2 can also lower logging activities and reduce class 2 elapsed time and class 2 CPU time.

Insert Algorithm 2 applies only to universal table spaces the use MEMBER CLUSTER, and is the default algorithm for this table space type. It is not applicable to other table space types.

For more information about the new insert algorithm in Db2 12, see 2.3, "Insert Algorithm 2" on page 50.The OMPE statistics work file database section was enhanced to track the number of times fast insert pipes were allocated and disabled. The highlighted fields that are shown in Example 12-63 on page 372 show the changes to this section of the OMPE statistics report.

Example 12-63 Statistics: Work file database section

```
WORKFILE DATABASE                           QUANTITY
--------------------------                  ---------------
TOTAL STORAGE CONFIG (KB)                   256.00
TOT DGTT STOR CONFIG (KB)                   128.00
TOT WF STOR CONFIG (KB)                     128.00
TOTAL STORAGE THRESHOLD (%)                 90.00
       ….•      ….•    ….•  ….•             ……
….•      ….•    ….•  ….•                    ……
AGENT STORAGE CONFIG (KB)                   0.00
NUMBER OF LIMIT EXCEEDED                    0.00
AGENT STORAGE THRESHOLD (%)                 90.00
MAX AGENT STORAGE USED (KB)                 0.00
DM FAST INSERT PIPES                        0.00  (QISTINPA)
DM FAST INSERT PIPES DISAB                  0.00  (QISTINPD)
```

The accounting class 3 and class 8 suspension times sections of the OMPE accounting report were enhanced to show the accumulated pipe wait time associated with the use of Insert Algorithm 2 and the accumulated number of pipe wait events.

The layout for the class 3 suspension sections of the OMPE accounting report is shown in Example 12-64 on page 373. The layout for the class 8 suspension sections of the report is shown in Example 12-65 on page 373. (The highlighted Fast Insert Pipe fields were added to the suspension time sections of the accounting report.)

12.2.3 LOB compression

Db2 12 supports compressing LOB data when the zEnterprise data compression (zEDC) hardware is available on the z/OS environment.

Compressing LOB data by using the zEDC hardware can reduce the size of the data in a LOB table space. LOB data compression can also be enabled on all LOB table spaces that are associated with the Db2 directory.

LOB compression is enabled for individual table spaces by specifying the COMPRESS YES attribute on the CREATE TABLESPACE or ALTER TABLESPACE statement. To enable LOB compression for the Db2 directory, the COMPRESS_DB2_DIR_LOBS subsystem parameter must be set to YES.

The accounting times class 3 suspension section of the OMPE accounting report was enhanced to show the accumulated wait time that is associated with compressing LOB data.

The layout for the class 3 suspension section of the OMPE accounting report is shown in Example 12-64 . The highlighted LOB Compression field was added to this report to report the accumulated (average) wait time and the number of events.

Example 12-64   Accounting: Class 3 suspensions section

```
CLASS 3 SUSPENSIONS                AVERAGE TIME       AV.EVENT
-------------------                ------------       --------
LOCK/LATCH(DB2+IRLM)               0.000000           0.00
IRLM LOCK+LATCH                    0.000000           0.00
DB2 LATCH                          0.000000           0.00
SYNCHRON. I/O                      0.000254           0.83
DATABASE I/O                       0.000254           0.83
LOG WRITE I/O                      0.000000           0.00
OTHER READ I/O                     0.000446           0.33
OTHER WRTE I/O                     0.000000           0.00
SER.TASK SWTCH                     0.000000           0.00
UPDATE COMMIT                      0.000000           0.00
OPEN/CLOSE                         0.000000           0.00
SYSLGRNG REC                       0.000000           0.00
EXT/DEL/DEF                        0.000000           0.00
OTHER SERVICE                      0.000000           0.00
ARC.LOG(QUIES)                     0.000000           0.00
LOG READ                           0.000000           0.00
DRAIN LOCK                         0.000000           0.00
CLAIM RELEASE                      0.000000           0.00
PAGE LATCH                         0.000000           0.00
NOTIFY MSGS                        0.000000           0.00
GLOBAL CONTENTION                  0.000000           0.00
COMMIT PH1 WRITE I/O               0.000000           0.00
ASYNCH CF REQUESTS                 0.000000           0.00
TCP/IP LOB XML                     0.000000           0.00
ACCELERATOR                        0.000000           0.00
AUTONOMOUS PROCEDURE               N/A                N/A
PQ SYNCHRONIZATION                 N/A                N/A
LOB COMPRESSION (AWLCSUST)         0.000000           0.00 (AWLCSUSC)
FAST INSERT PIPE (AWPISUST)        0.000000           0.00 (AWPISUSC)
TOTAL CLASS 3                      0.000700           1.17
```

The pipe wait time and number of events that were added to the package class 8 suspension time section of the OMPE accounting report is shown in Example 12-65.

Example 12-65   Accounting package times: Class 8 suspensions section

```
   PACKAGE                AVERAGE TIME          AVG.EV
   -------                ------------          ------
   LOCK/LATCH             0.000000              0.00
      IRLM LOCK+LATCH     0.000000              0.00
      DB2 LATCH           0.000000              0.00
   SYNCHRONOUS I/O        0.000000              0.00
   …..     …..            …..                    …..
   …..     …..            …..                    …..
   FAST INSERT PIPE       0.000000 (APPISUST)   0.00 (APPISUSC)
   TOTAL CL8 SUSPENS.     0.000000              0.00
```

No LOB compression suspension counters are available at the package level. At the package level, the time and number of times a transaction is waiting for LOB compression to complete is included in the SERV.TASK SWITCH counters.

12.2.4 Contiguous buffer pools: PGSTEAL(NONE)

In Db2 12, buffer pool management has changed when using the PGSTEAL(NONE) option of the ALTER BUFFERPOOL command. The improved PGSTEAL(NONE) buffer pool management in Db2 12 can retrieve pages that are in the buffer pool faster, which provides CPU reduction for transactions that use these buffer pools. For more information about how contiguous buffer pools are managed and the performance test results, see 2.2, “Contiguous buffer pools” on page 39.

The buffer pool Read Operation section of the OMPE statistics report was enhanced to enable monitoring the health of these contiguous buffer pools, as shown in Example 12-66. The highlighted fields are added to the report to record the number of getpage requests or reads that are performed against the overflow area of the buffer pool. If these values are nonzero, it is recommended to increase the size of the affected buffer pool.

Example 12-66 Statistics report: Buffer pool read operations section

```
BP0 READ OPERATIONS                       QUANTITY
-------------------                       --------
BPOOL HIT RATIO (%)                       100.00
BPOOL HIT RATIO (%) SEQU                  100.00
BPOOL HIT RATIO (%) RANDOM                100.00
GETPAGE REQUEST                           30390.00
      GETPAGE REQS-SEQUENTIAL             8541.00
           IN-MEM OVFL SEQ REQS           0.00 (QBSTASGE)
      GETPAGE REQS-RANDOM                 21849.00
           IN-MEM OVFL RND REQS           0.00 (QBSTAGET)
SYNCHRONOUS READS                         5013.00
       SYNC READS-SEQUENTIAL              0.00
           IN-MEM OVFL SEQ READS          0.00 (QBSTASSE)
       SYNC READS-RANDOM                  0.00
           IN-MEM OVFL RND READS          0.00 (QBSTASYN)
......•.•......•......•.•......•          ......•.•
......•.•......•......•.•......•          ......•.•
```

A

IBM Db2 workloads

This appendix provides an overview of the workloads that are used to measure Db2 performance.

Traditionally, performance workloads started as heavy-duty batch jobs, which stressed CPU and I/O of the system.

With the advent of teleprocessing and multi-address spaces, online TP became necessary and often terminal simulators were used. At Db2 Version 4 time, the online TP IBM Relational Warehouse Workload (IRWW) was created, which is based on a retail type of environment that uses IMS and later CICS as transaction monitor.

In later years, more workloads related to complex OLTP, distributed environments query processing and data warehousing were added.

Special high-insert batch workloads were designed to compare physical design options and different structures to minimize contention.

As noted in Chapter 1, “Introduction ” on page 1:

- A set of IRWW workloads represents the simple, legacy type, OLTP workload that is typically run through IMS or CICS as transaction server.
- Distributed IRWWs are the same Db2 workloads that are run through remote clients, such as JDBC or remote stored procedure calls.
- Brokerage OLTP represents a complex OLTP with a mixture of lookup, update, and insert and reporting queries.
- SAP Banking Day Posting workloads are relatively simple SQL executions against an SAP banking schema.
- High insert workloads are concurrent inserts that are run through more than 100 JCC T4 connections in data sharing.
- TSO batches represent a batch workload that is running various batch operations, including SELECT, FETCH, UPDATE, INSERT, and DELETE in a loop.

- A set of query workloads are run serially mostly by using CPU parallelism. Each query workload consists of a set of 20 - 100 queries per workload. Individual queries can see wide range of differences between Db2 10 and Db2 11. The average CPU reduction rate is shown here.
- IBM Cognos® BI-day short workload is a set of short running reporting queries that are concurrently executed.
- Utilities scenarios are a set of various utility job executions.
- XML scenarios are also a set of various XML workloads, including OLTP, queries, batch, and utilities against XML data types.
- WebSphere Application Server Portal workload is used to illustrate this process, and the performance effects that this process can produce.

This appendix includes the following topics:

- IBM Relational Warehouse Workload
- Relational Transaction Workload
- IRWW distributed workload
- Brokerage

A.1 IBM Relational Warehouse Workload

IRWW is an OLTP workload that consists of seven transactions. Each transaction consists of one to many SQL statements, each performing a distinct business function in a predefined mix.

The transactions that are used in the Classic IRWW workload are listed in Table A-1.

Table A-1 Classic IRWW transactions

Transaction	Relative % mix	DML executed
Delivery		OPEN/FETCH/SELECT/UPDATE/DELETE
New Order	22	OPEN/FETCH/SELECT/UPDATE/INSERT
Order Status	25	OPEN/FETCH/SELECT
Payment	21	OPEN/FETCH/SELECT/UPDATE/INSERT
Price Change		UPDATE
Price Quote	25	SELECT
Stock Inquiry		SELECT

The following transaction types are listed in Table A-1:

- Delivery

  Performs various SELECT, UPDATE, and DELETE transactions in support of the delivery of a group of orders and runs as 2% of the total transaction mix.

- New Order

  Performs various SELECT, FETCH, UPDATE, and INSERT transactions in support of the receipt of new customer orders and runs as 22% of the total transaction mix.

- Order Status

  Performs various SELECT and FETCHE transactions in support of providing the status of an order and runs as 25% of the total transaction mix.

- Payment

  Performs SELECT, FETCHE, UPDATE, and INSERT transactions in support of received customer payments and runs at 21% of the total transaction mix.

- Price Change

  Performs an UPDATE in support of changing the price of an item and runs as 1% of the total transaction mix.

- Price Quote

  Performs various SELECT transactions in support of providing the price of a set of items and runs as 25% of the total transaction mix.

- Stock Level

  Performs a JOIN and various SELECT transactions in support of providing the current stock level of an item and runs at 4% of the mix.

The IRWW database contains several inventory stock warehouses and sales districts.

The front end transaction manager often is IMS, but CICS is used at times.

Historically the Classic IRWW workload used IMS Fast Path (IFP) regions for transaction execution. IFPs bypass IMS queuing allows for more efficient processing, including thread reuse.

However, the new IMS connection pooling support, is not applicable for IMS IFP regions. Therefore, the Classic IRWW workload setup was changed to use IMS Message Processing Regions (MPPs) to schedule and run the programs corresponding to different transactions.

Note: Because of this change, previously published IRWW results (for example, from Db2 10 for z/OS benchmarks) must not be compared to IRWW runs that use Db2 12 for z/OS.

The Processing Limit Count (PLCT) parameter is used to control the maximum number of messages that is sent to the application program by the IMS control program for processing without reloading the application program in the MPP region. This value is set to 65535 for two-thirds of the transactions, and 0 for the remaining transactions.

PLCT(0) means the maximum number of messages that is sent to the application is one and the application program is reloaded into the MPP region before receiving a subsequent message. PLCT(65535) means that no limit is placed upon the number of messages that is processed at a single program load.

A.2 Relational Transaction Workload

The Relational Transaction Workload (RTW) is a standard workload that is used by the CICS performance team in Hursley to assess changes in performance characteristics for new CICS releases for applications accessing Db2. The Db2 team decided to use the RTW as one of its performance test benchmark workloads to get better performance analysis coverage for CICS-Db2 type of workloads. The workload was modified slightly compared to the version the CICS team uses to better reflect the Db2 behavior of the workload.

The RTW features the following characteristics:

- All programs that make up the workload are written in COBOL.
- The workload uses 20 (data) tables in segmented table spaces, 11 non-unique indexes, and 16 unique indexes.
- It issues on average 200 Db2 calls per transaction.
- The transaction mix consists of 26% select, 3% insert, 7% update, 3% delete, 6% open cursor, 36% fetch cursor, 6% close cursor operations.

The following environment configuration is used for the non-data sharing RTW workload:

- One z13 LPAR with 8 general CPs
- Four CICS regions that are running CICS TS V5.1
- A non-data sharing Db2 subsystem
- z/OS 2.1
- DS8870 DASD controller
- TPNS on another z13 LPAR with four general CPs to drive the workload

The following environment is used by the two-way data sharing RTW:

- Two z13 LPARs with 8 general CPs each
- Eight CICS regions that are running CICS TS 5.1 with four regions that are running on each of the LPARs
- Two z13 ICF LPARs that are running CFLEVEL 21 CFCC
- Two Db2 subsystems, each running on one of the two LPARs
- DS8870 DASD controller
- TPNS on another z13 LPAR with four general CPs to drive the workload

A.3 IRWW distributed workload

The IRWW OLTP workload, consisting of seven transactions in support of a product warehouse, customer order, and order delivery system, was converted to run in distributed environments. Each transaction consists of approximately 25 DML on average with seven of those DMLs being Insert/Update/Delete. The workloads consist of stored procedure and non-stored procedure workloads.

For more information about the seven transaction types, a brief description of each, and the percentage of the transaction mix, see “IBM Relational Warehouse Workload”.

A.4 Brokerage

Brokerage OLTP represents a complex OLTP with a mixture of lookup, update, insert, and reporting queries.

The IBM Brokerage transaction workload is a complex OLTP workload that runs various SQL procedures to simulate transactions in a brokerage farm.

An OLTP workload was run that accesses 33 tables with approximately 15,000 objects. The Brokerage workload represents transactions of customers dealing in the financial market and updating account information. The transactions are complex OLTP transactions that are implemented in stored procedure by using SQL/PL. The workload uses:

- 1.3 TB of data
- 13 buffer pools
- 33 table spaces
- 68 indexes

Redbooks

IBM Db2 12 for z/OS Performance Topics

SG24-8404-00

ISBN 0738442712

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8404-00

ISBN 0738442712

Printed in U.S.A.