Information Management software

Redpaper

Mike Bracey

DB2 9 for z/OS: Buffer Pool Monitoring and Tuning

Introduction

DB2® buffer pools are a key resource for ensuring good performance. This is becoming increasingly important because the difference between processor speed and disk response time for a random access I/O widens in each new generation of processor. A System z® processor can be configured with large amounts of storage which, if used wisely, can help compensate by using storage to avoid synchronous I/O.

The purpose of this paper is to:

- Describe the functions of the DB2 buffer pools
- Introduce a number of metrics for read and write performance of a buffer pool
- Make recommendations on how to set up and monitor the DB2 buffer pools
- Provide results from using this approach at a number of customers

The paper is intended for DB2 system administrators, but may be of interest to any z/OS® performance specialist. It is assumed that the reader is familiar with DB2 and performance tuning. If this is not the case, the *DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide,* SC18-9851, is recommended reading. The buffer pool parameters that are discussed in this paper can be adjusted by using the ALTER BUFFERPOOL command, which is documented in the *DB2 Version 9.1 for z/OS Command Reference,* SC18-9844. DB2 manuals can be found at the following Web address:

http://www.ibm.com/software/data/db2/zos/library.html

The ALTER BUFFERPOOL command is documented here in the DB2 Information Center at:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.comref/db2z_cmd_alterbufferpool.htm

There are a number of decisions that a DB2 system administrator must make, such as choosing many parameters. The key questions are:

- How many pools should be defined for each page size?

- How large should they be (VPSIZE)?
- Which table spaces and index spaces should be allocated to each pool?
- What values should be used for:
  - Page Fix - PGFIX - Yes or No
  - Page steal method - PGSTEAL - LRU or FIFO
  - Virtual Pool Sequential Threshold - VPSEQT
  - Deferred Write Queue Threshold - DWQT
  - Vertical Deferred Write Queue Threshold - VDWQT

In this paper we describe some tools and techniques which can be used to answer these questions.

This paper does not cover all the topics concerned with buffer pools, such as buffer pool usage with compressed indexes, group buffer pool tuning when using DB2 data sharing, and DB2 latch contention analysis and resolution.

Background

This section provides background information on buffer pool management and I/O accesses by DB2.

DB2 Buffer Pools

DB2 tables are stored in table spaces and indexes are stored in index spaces. Simple (deprecated with DB2 9) and segmented table spaces can contain one or more tables; whereas *partitioned*, *universal partitioned by growth*, and *universal partitioned by range* table spaces contain only one table. An index space contains one and only one index. The term page set is used generically to mean either a table space or an index space. A page set, whether partitioned or not, consists of one or more VSAM linear data sets. Within page sets, data is stored in fixed sized units called *pages*. A page can be 4 KB, 8 KB, 16 KB, or 32 KB. Each data set page size has a number of buffer pools that can be defined, such as BP0 or BP32K with corresponding buffer size.

DB2 functions that process the data do not directly access the page sets on disk, but access virtual copies of the pages held in memory, hence the name *virtual buffer pool*. The rows in a table are held inside these pages. Pages are the unit of management within a buffer pool. The unit of I/O is one or more pages chained together.

Prior to Version 8, DB2 defined all data sets with VSAM control intervals that were 4 KB in size. Beginning in Version 8, DB2 can define data sets with variable VSAM control intervals. One of the biggest benefits of this change is an improvement in query processing performance. The VARY DS CONTROL INTERVAL parameter on installation panel DSNTIP7 allows you to control whether DB2-managed data sets have variable VSAM control intervals.

What is a GETPAGE?

DB2 makes a GETPAGE request operation whenever there is a need to access data in a page, either in an index or a table, such as when executing an SQL statement. DB2 checks whether the page is already resident in the buffer pool and if not, performs an I/O to read the page from disk or the group buffer pool if in data sharing.

A getpage is one of three types:

- Random getpage
- Sequential getpage
- List getpage

The type of getpage is determined during the bind process for static SQL and the prepare process for dynamic SQL, and depends on the access path that the optimizer has chosen to satisfy the SQL request. The number and type of getpages for each page set can be reported from the IFCID 198 trace record. This trace is part of the performance trace class and can be CPU intensive to collect due to the very high number of getpage requests per second in a DB2 subsystem. See the *DB2 Version 9.1 for z/OS Command Reference,* SC18-9844 for documentation of the START TRACE command.

The DB2 command DISPLAY BUFFERPOOL and the DB2 statistics trace also report the number of getpages; however, there are only two counters: one for random and one for sequential. The list getpage is counted as a random getpage for both the DISPLAY and statistics trace counters. A random getpage is always counted as random irrespective of whether the page is subsequently read with a synchronous or dynamic prefetch I/O.

See “Appendix B – Sample DISPLAY BUFFERPOOL” on page 21 for sample output from the DISPLAY BUFFERPOOL command and “Appendix C – Sample statistic report” on page 22 for a sample from a statistics report.

DB2 checks whether the page requested is in the local buffer pool or is currently being read by a prefetch engine, in which case DB2 will wait for the prefetch to complete. If the requested page is not found in the local buffer pool in data sharing, then the global buffer pool is also checked. If the page is still not found then a synchronous I/O is scheduled.

How is I/O performed?

DB2 can schedule four different types of read I/O for SQL calls:

- Synchronous I/O of a single page, also called a random read
- Sequential prefetch of up to 32 (V8) or 64 (V9) contiguous 4 KB pages
- List prefetch of up to 32 contiguous or non-contiguous 4 KB pages
- Dynamic prefetch of up to 32 contiguous 4 KB pages

Note that the prefetch quantity for utilities can be twice that for SQL calls. A synchronous I/O is scheduled whenever a requested page is not found in the local or, if data sharing, global buffer pool and always results in a delay to the application process whilst the I/O is performed. A prefetch I/O is triggered by the start of a sequential or list prefetch getpage process or by dynamic sequential detection. The I/O is executed ahead of the getpage request, apart from the initial pages, for a set of pages which are contiguous for sequential and dynamic prefetch. List prefetch normally results in a non-contiguous sequence of pages however these pages may also be contiguous.

The I/O is executed by a separate task, referred to as a prefetch read engine, which first checks whether any of the pages in the prefetch request are already resident in the local pool. If they are then they are removed from the I/O request. Sequential prefetch does not start until a sequential getpage is not found in the pool, referred to as a “page miss.” Once triggered, it continues the prefetch process irrespective of whether the remaining pages are already resident in the pool. List and dynamic prefetch engines are started whether or not the pages are already in the pool. This can result in prefetch engines being started only to find that all pages are resident in the pool. The prefetch engines can be disabled at the buffer pool level by setting VPSEQT=0. There is a limit, dependent on the version of DB2, on the total number of read engines. The DB2 Statistics report shows when prefetch has been disabled due to the limit being reached. The actual number of pages fetched depends on the buffer pool page size, the type of prefetch, the size of the buffer pool, and the release of DB2. For V8 refer to the *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413 and for V9 refer to the *DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide,* SC18-9851 for the specification.

Dynamic prefetch is triggered by a sequence of synchronous getpages which are close to sequential and can be either ascending or descending in direction. If a page is not in the virtual pool, which includes the group buffer pool for data sharing, then DB2 will schedule a synchronous I/O irrespective of the type of getpage. This is why the first one or two pages of a sequential prefetch will be read with synchronous I/Os.

In DB2 9, sequential prefetch is only used for table space scans. Other access paths that used to use sequential prefetch will now rely on dynamic prefetch. The result of this change is more use of dynamic prefetch I/O than in previous versions of DB2. This change in behavior takes place in V9 conversion mode and is automatic with no REBIND required.

Page stealing

Page stealing is the process that DB2 performs to find space in the buffer pool for a page that is being read into the pool. The only pages that can be stolen are available pages; see “Page processing” on page 5 for more information.

The page that is stolen depends on a number of factors:

- The choice of the page steal parameter PGSTEAL
- The value of the virtual pool sequential threshold VPSEQT
- The current number of random and sequential pages in the pool
- Whether the page being read in is random or sequential

DB2 marks each page in the pool as either random or sequential and therefore knows the proportion of sequential pages in the pool at any time. There are two options for the page steal parameter PGSTEAL:

- Least Recently Used (LRU)

  This option ensures that the least recently used page is chosen.

- First In First Out (FIFO)

  Choosing this option means that pages are stolen in the chronological sequence in which they were first read into the pool irrespective of subsequent usage.

Use of FIFO page stealing can reduce both CPU and latch contention because DB2 does not need to track each time a page is used. Buffer pool latch contention is recorded in latch class 14, which can be found in the Tivoli® OMEGAMON® XE for DB2 Performance Expert on z/OS Statistics Long report. A rate of less than 1,000 per second is considered acceptable, but a rate of more than 10,000 per second should be investigated. FIFO is recommended for *data in memory* buffer pools that have a 100% buffer pool hit ratio such that no or very limited page stealing occurs.

The pages in a pool are classified as either random or sequential depending on the following conditions: the type of getpage request, the I/O that was used to bring the page into the pool, whether the page was actually touched and the DB2 Version.

Assuming that the page is neither in the virtual pool nor the group buffer pool for data sharing, then:

- A random getpage which results in a synchronous I/O is classified as a random page. A random page stays random irrespective of subsequent getpage type.
- A random getpage which results in a dynamic prefetch is classified as a random page in V8 and a sequential page in V9. Any pages that were read in as part of the dynamic prefetch that are not touched by a random getpage remain as sequential pages irrespective of DB2 version.
- A sequential getpage which results in a sequential prefetch I/O is classified as a sequential page. A subsequent random or list prefetch getpage will reclassify the page as random in V8 but not in V9.
- A list getpage which results in a list prefetch is classified as a random page in V8 and a sequential page in V9. Note that the getpage is still counted as random in both the statistics report and the DISPLAY BUFFERPOOL command.

If a page is resident in the pool, then in V8, a random getpage classifies the page as random irrespective of the I/O that brought the page into the pool. In V9 a page is never reclassified after the initial I/O.

If the page being read is sequential and the proportion of sequential pages is at or above the limit defined by VPSEQT then the least recently used or first read sequential page is chosen for stealing. If the page is random or the proportion of sequential pages is less than the limit defined by VPSEQT then the least recently used or first read page is stolen irrespective of whether it is random or sequential. This procedure ensures that the number of sequential pages in the pool does not exceed the threshold set by VPSEQT. If VPSEQT is set to 0 then all pages are treated equally; however, note that setting VPSEQT to 0 turns off all prefetch read I/O and also turns off parallel query processing. Setting VPSEQT to 100 means that all pages are treated equally for page stealing.

Query parallelism and the utilities COPY (in V9 but not V8), LOAD, REORG, RECOVER, and REBUILD all make a page that has been finished with available to be stolen immediately. This avoids flooding the pool with pages that are unlikely to be reused.

Page processing

The pages in a buffer pool are further classified into the following three types:

- In use pages

  As the name suggests, these are the pages that are currently being read or updated. This is the most critical group because insufficient space for these pages can cause DB2 to have to queue or even stop work. The number of these pages is generally low, tens or hundreds of pages, relative to the size of the pool.
- Updated pages

  These are pages that have been updated but not yet written to disk storage, sometimes called dirty pages. The pages may be reused by the same thread in the same unit of work and by any other thread if row locking is used and the threads are not locking the same row. Under normal operating conditions, the pages are triggered for writing asynchronously by the deferred write threshold (DWQT) being reached, the vertical deferred write threshold (VDWQT) for a data set being reached, or when a system checkpoint falls due. By holding updated pages on data set queues beyond the commit point, DB2 can achieve significant write efficiencies. The number of pages in use and updated can be shown with the DISPLAY BUFFERPOOL command. There could be hundreds or even thousands of pages in this group.

- Available pages

  These are the pages that are available for reuse and thus avoid I/O; however, they are also available to be overwritten or stolen to make space for any new page that has to be read into the pool. Available pages are normally stolen on a *least recently used* basis (LRU/sequential LRU), but first-in-first-out (FIFO) can also be specified. An important subset of the available pages consists of those that have been prefetched into the pool in advance of use by a sequential list or dynamic prefetch engine. These pages are like any available page in that they can be stolen if the page stealing algorithm determines that they are a candidate, even if the page has not yet been referenced. If these pages are stolen and subsequently a getpage occurs, then DB2 will schedule a synchronous I/O as it would for any page that is not in the buffer pool when needed. This can happen when the pool is under severe I/O stress or when the size reserved for sequential I/Os is too small for the work load.

DB2 tracks the number of unavailable pages, which is the sum of the in-use and the updated pages, and checks the proportion of these to the total pages in the pool against three fixed thresholds:

- Immediate write threshold (IWTH): 97.5%
- Data management threshold (DMTH): 95%
- Sequential prefetch threshold (SPTH): 90%

Refer to *DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide,* SC18-9851, for a description of these thresholds and the action that DB2 takes when a threshold is exceeded. In normal operation a buffer pool should be sized to never or very rarely hit these thresholds. The Statistics report shows the number of times these thresholds have been reached. These values should be zero or close to zero under normal operating conditions.

Page writes

In most cases updated pages are written asynchronously from the buffer pool to disk storage. Updated pages are written synchronously when:

- An immediate write threshold is reached.
- No deferred write engines are available.
- More than two checkpoints pass without a page being written.

DB2 maintains a chain of updated pages for each data set. This chain is managed on a *least recently used* basis. When a write is scheduled, DB2 takes up to 128 pages off the queue on an LRU basis (depending on the threshold that triggered the write operation), sorts them by page number and schedules write I/Os for 32 pages at a time, but with the additional constraint that the page span be less than or equal to 180 pages. This could result in a single write I/O for all 32 pages or in 32 I/Os of one page each. An insert of multiple rows in a clustering sequence would be expected to result in more pages per I/O and so fewer I/Os, whereas transaction processing with more random I/O would tend to result in fewer pages per write I/O and therefore require more I/Os.

DB2 maintains a counter of the total number of updated pages in the pool at any one time. DB2 adds one to this counter every time a page is updated for the first time and then checks two thresholds to see whether any writes should be done. The first threshold is the deferred write queue threshold (DWQT). This threshold applies to all updated pages in the buffer pool and is expressed as a percentage of all pages in the buffer pool. If the number of updated pages in the buffer pool exceeds this percentage, asynchronous writes are scheduled for up to 128 pages per data set until the number of updated pages is less than the threshold. The default for DWQT in DB2 9 for z/OS is 30%, which should be the starting point for most customers.

The second threshold is the vertical deferred write queue (VDWQT). This threshold applies to all pages in a single data set. It can be specified as a percentage of all pages in the buffer pool, such as 5, or as an absolute number of pages such as 0,128 meaning 128 pages. If VDWQT=0 then this is equivalent to 0,40 for a 4 KB pool, which means the threshold is 40 pages.

In DB2 9 for z/OS the default is set to 5% of all pages in the pool. So, for example, if a pool had 100,000 pages, DWQT=30 and VDWQT=5, then writes would be scheduled if the number of updated pages in the pool exceeded 30,000 or the number of pages for any one data set exceeded 5,000. If DWQT is triggered, then DB2 continues to write until the number of updated pages in the pool is less than (DWQT-10)%. So for DWQT=30%, writes will continue until the number of updated pages in the pool is 20%. If the VDWQT is triggered, DB2 writes out up to 128 pages of the data set that triggered the threshold.

Write operations are also triggered by checkpoint intervals. Whenever a checkpoint occurs, all updated pages in the pool at that time are scheduled for writing to disk. If a page is in use at the time of the scheduled I/O, it may be left until the next checkpoint occurs. Writes are also forced by a number of other actions such as a STOP DATABASE command or stopping DB2.

Buffer pool statistics

Buffer pool statistics are available from the DB2 statistics and accounting trace data, which can be formatted using a product such as the Tivoli OMEGAMON XE for DB2 Performance Expert for z/OS, details for which can be found at:

http://www.ibm.com/software/tivoli/products/omegamon-xe-db2-peex-zos/

The *Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, V4R1 Report Reference*, SC18-9984, manual can be found at:

http://www.ibm.com/software/data/db2imstools/db2tools-library.html#omegaxepe-lib

This book includes a description of each of the fields in the statistics report. Statistical data is also available by using the DISPLAY BUFFERPOOL command (a sample output is listed in “Appendix B – Sample DISPLAY BUFFERPOOL” on page 21. These sources should be the starting point for any tuning exercise.

Buffer pool read efficiency measures

In this section we examine the following three metrics of buffer pool read efficiency:

- Hit ratio
- Page residency time
- Reread ratio

Hit ratio

This is the most common and easiest to understand metric. There are two variations:

- System Hit Ratio = (total getpages – total pages read) / total getpages
- Application Hit Ratio = (total getpages – synchronous pages read) / total getpages

where:

total getpages = random getpages + sequential getpages
(Field Name: QBSTGET)

synchronous pages read = synchronous read I/O for random getpages + synchronous read I/O for sequential getpages
Field Name: QBSTRIO)

total pages read = synchronous read I/O for random getpages + synchronous read I/O for sequential getpages + sequential prefetch pages read asynchronously + list prefetch pages read asynchronously + dynamic prefetch pages read asynchronously
(Field Name: QBSTRIO + QBSTSPP + QBSTLPP + QBSTDPP)

These values can be obtained from the DISPLAY BUFFERPOOL command messages DSNB411I, DSNB412I, DSNB413I, and DSNB414I. They are also collected in the statistics trace by IFCID 002. The data from a statistics trace can be loaded into a performance database by using a product such as IBM® Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS. See “Appendix D - Hit ratio and system page residency time” on page 23 for a query that calculates the hit ratio and the page residency times.

The application hit ratio ignores prefetch pages and so will be equal to or higher than the system hit ratio. The hit ratio is simple to calculate; however, it is workload dependent and is not so useful when comparing the efficiency of two or more pools to decide whether a pool should be allocated more or less storage. If the number of pages read during an interval is greater than the number of getpages, then the hit ratio is negative. This can be caused by a prefetch I/O reading more pages than are actually touched.

Page residency time

This metric is just as simple to calculate as the hit ratio but is intuitively more meaningful. The idea is to calculate the average time that a page is resident in the buffer pool. There are three measures that can be used:

- System residency time (seconds) = buffer pool size / total pages read per second
- Random page residency time (seconds) = Maximum (System residency time, (buffer pool size * (1-VPSEQT/100) / synchronous pages read per second)
- Sequential page residency time (seconds) = Minimum (System residency time, (buffer pool size * (VPSEQT/100) / asynchronous pages read per second)

where:

synchronous pages read = synchronous read I/O for random getpages + synchronous read I/O for sequential getpages
(Field Name: QBSTRIO)

total pages read = synchronous read I/O for random getpages + synchronous read I/O for sequential getpages + sequential prefetch pages read asynchronously + list prefetch pages read asynchronously + dynamic prefetch pages read asynchronously
(Field Name: QBSTRIO + QBSTSPP + QBSTLPP + QBSTDPP)

The page residency time can be used as a measure of the stress a pool is under and therefore as an indicator of the likelihood of page reuse. The metric is an average and so some pages will be in the pool for much less time while other pages that are in constant use, such as the space map pages and non-leaf pages of an index, may remain permanently resident. The three measures produce the same result when the proportion of random to sequential pages read I/O is less than the virtual pool sequential threshold, such that the threshold is not actually being hit. If the threshold is being hit then the measures are different in this sequence:

Random page residency > System residency > Sequential page residency

The page residency time is an indicator of stress rather like a person's pulse rate or temperature. Experience suggests that a residency time of less than 5 minutes is likely to indicate an opportunity for improvement by increasing the size of the pool. It may be acceptable for the sequential page residency time to be less than 5 minutes, but if it is less than 10 seconds then it is possible that pages will be stolen before being used, resulting in synchronous I/Os and more pressure on the buffer pool.

An indicator of buffer pool stress is the quantity of synchronous reads for sequential I/O. As explained above, these may occur for unavoidable reasons. However, if the rate is excessive then this can be a clear confirmation of the deleterious effect of a buffer pool that is too small for the workload, or it could be that the indexes and/or table spaces are disorganized.

The DISPLAY BUFFERPOOL command shows the synchronous reads for sequential I/O which ideally should be a small percentage of total I/O. These reads can either occur because the page has been stolen or because sequential prefetch is being used on an index, but the index is disordered and therefore out of sequence. Note that DB2 9 relies on dynamic prefetch for index access with or without data reference rather than sequential prefetch.

Conversely, if the residency time is more than 15 minutes, then there may be an opportunity to reduce the size unless this is a "data in memory" pool. Ideally, measurements should be taken at between 5 and 15-minute intervals across a representative 24-hour period.

The hit ratio and buffer pool residency time can be calculated from the data captured by the statistics trace and stored in the IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on a z/OS database. An example of the SQL that can be used is shown in "Appendix D - Hit ratio and system page residency time" on page 23.

Reread ratio

The reread ratio was proposed by Mike Barnard of GA Life, UK (see "References" on page 26). The idea is to measure how often a page is synchronously read more than once in an interval of time usually between 5 and 10 minutes. This gives an absolute measure of the maximum number of I/Os that could have been avoided for that interval. It can therefore be used as a comparative metric between pools to help with deciding which pool may have a higher payback in terms of I/O avoidance. The reread ratio does not provide guidance on how much the pool size should be changed nor predict the outcome of a change. A way to calculate the reread ratio is to turn on the IFCID 6 trace and then analyze the output using a product such as DFSORT.

Buffer pool write efficiency measures

There are two metrics of buffer pool write efficiency:

- Number of page updates for each page written
- Number of pages written for write I/O

Number of page updates for each page written

The expression is:

Page updates per page written = buffer page updates / pages written

where:

buffer page updates is the total number of page updates in a given statistics interval. (trace field QBSTSWS)

pages written is the total number of page writes in a given statistics interval. (trace field QBSTPWS)

This is a measure of how many write I/Os are avoided by keeping the updated page in the pool long enough to be updated more than once before being written to disk. The lowest value for this metric is one and higher is better.

Number of pages written for write I/O

The expression is:

Pages written per write I/O = pages written / write I/Os

where:

pages written is the total number of pages writes in a given statistics interval. (trace field QBSTPWS)

write I/Os is the total of asynchronous and synchronous writes in a given statistics interval. (trace field QBSTIMW + QBSTWIO)

This is a measure of the efficiency of the write process. Remember that a write I/O consists of up to 32 pages (64 for a utility) from a single data set where the span of pages (first page to last page) is less than or equal to 180. The lowest value for this metric is one and higher is better, though it should be noted that a higher value can lead to increased OTHER WRITE I/O wait as well as page latch contention wait, because those pages being written out are unavailable and all updates to those pages must be suspended.

Considerations on buffer pools write measures

The metrics we examined are dependent on the type of workload being processed and can be expected to vary across a normal working day as the mix of batch and transaction processing varies. The metrics are more likely to increase when you:

- Increase the number of rows per page, for example by using compression
- Increase the size of the buffer pool
- Increase the deferred write thresholds
- Increase the checkpoint interval

All these increase the likelihood of a page being updated more than once before being written, and also of another page being updated that is within the 180-page span limit for a write I/O. Of course, there may be other consequences of taking any of the above actions.

Tuning recommendations

We now examine the best settings of buffer pools for the following:

- How many buffer pools are needed
- Buffer pool sizing
- Virtual Pool Sequential Threshold (VPSEQT)
- Vertical Deferred Write Queue Threshold (VDWQT)
- To page fix or not to page fix

How many buffer pools are needed

The first question is how many pools are required. The preference is the fewer the better. However, there are a number of good reasons for having more than one for each page size. DB2 can manage pools of multiple gigabytes; however, what sometimes occurs is that many small pools are defined. Inevitably, as the workload varies during the day and week, some pools will be overstressed while others are doing very little.

Ideally there should be no more than one pool for each combination of buffer pool parameters, though in practice there may be good reasons for having more. A common approach for the 4 KB pools would be as follows:

1. Restrict BP0 for use by the DB2 catalog and other system page sets. The rate of I/O is usually nontrivial and so PGFIX is recommended because this pool does not need to be very large.
2. Define a pool for sort and work data sets (Database TYPE='W'). This has often been BP7 to correspond with DSNDB07, but any 4 KB pool will do. If the pool is used solely for sort and work, then set VPSEQT=99. This will reserve a small amount of space for the space map pages. If the pool is used for declared or created global temporary tables, which behave more like user tables in that pages can be reused, then monitor the residency time to determine the optimum value for VPSEQT.

   Because I/O will occur for this pool, PGFIX=YES is normally recommended. The pool should be sized to ensure that most small sorts are performed without I/O occurring. Set PGSTEAL=LRU. Do not use PGSTEAL=FIFO as sort/merge performance could be severely impacted.
3. Allocate one or more pools for “data in memory” table spaces and index spaces. This pool is for small tables that are frequently read or updated. The pool should be sized to avoid read I/O after initialization and so should be at least equal to the sum of the number of pages in all page sets that are allocated to the pool. If I/Os occur or you wish to prevent paging of this pool under any circumstance due to the high priority of the pool and its small size, then use PGFIX=YES. In this case do not oversize the pool. If you do oversize the pool and can tolerate z/OS paging during a page shortage and if all objects in the pool are primarily read only, then PGFIX=NO should be used.

   Consider setting VPSEQT=0, which will prevent the scheduling of a prefetch engine for data that is already in the pool. If performance of this pool is critical following either a planned or unplanned outage, then you may wish to prime the pool by selecting the whole table. If this is done, then use the ALTER BUFFERPOOL command to set VPSEQT=100 before the priming process and back to 0 afterwards. Consider setting PGSTEAL=FIFO, which will save the CPU that is used for LRU chain management, but only if the read I/O rate is zero or minimal. DB2 buffer pool latch contention can be lowered by separating these types of objects out of the other pools.

   The challenge is finding out which page sets meet the criteria. The best way is to use a tool such as the IBM DB2 Buffer Pool Analyzer, which can capture the data from a trace and create data suitable for loading into DB2 tables for analysis. If that is not available, then it is possible to run a performance trace for IFCID 198 with a product such as IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS and either analyze the output with DFSORT or run a trace report. IFCID 198 produces one record for every getpage and may result in a very large number of records very quickly, so you are advised to use it with caution. You can check how much data is to be expected from the statistics trace or DISPLAY BUFFERPOOL command.

It is also possible that the application developers may know which tables are used in this manner.

4. Define other 4 KB pools for the user data, as required. A simple and widely used approach is to separate by table space and index space. The table space pool would be expected to have more asynchronous I/O and so could be smaller than the index space pool. The table space pool would also tend to have a higher value of VPSEQT, say 40% to 80%, compared to the index space pool, say 20% to 40%. If the system residency time = random page residency time = sequential page residency time for most periods, then lower VPSEQT until the threshold becomes effective. This is on the basis that any object can have both random and sequential access, but that random access is more painful than sequential because it causes a real wait to the application and so ought to be given a higher chance of being avoided.

   If tools are available to measure the degree of random to sequential access by page set, then another possibility is one or more pools for random access and others for sequential access. The difficulty here is measuring whether the access is random or sequential. The IFCID 198 getpage trace differentiates between random, sequential and list. However, a random getpage could still result in a dynamic prefetch and so just using the count of getpages may not be sufficient. You could trace IFCID 6 and 7, which track the start and finish of an I/O, respectively. IFCID 6 tells you the page number for a synchronous I/O but you need to trace IFCID 7 to get the page numbers actually read by a prefetch I/O. The I/O trace will only tell you the random-to-sequential split for a page set that requires I/O, but not for a page set that is in memory. As above, the pools for sequential access can be smaller than those for random access. All these pools should normally use PGSTEAL=LRU and PGFIX=YES.

5. If a non-partitioning index (NPI) is heavily updated during LOAD or REORG PART, then this can cause a flood of writes to the group buffer pool (GBP) in data sharing, which can be damaging for other processing. If these objects are separated out, then by setting the VDWQT=DWQT=very low value will cause a throttling effect on the writes to the GBP and subsequent castout processing. Note that the BUILD2 phase of a REORG is no longer performed in V9 because the whole NPI is rebuilt via a shadow.

6. Separate pools are recommended for LOB table spaces due to the different nature of getpage and I/O activity compared to regular table spaces.

7. Separate pools have to be used to prioritize one set of objects over another, for example an application that is so critical that it demands dedicated resources irrespective of other applications.

8. It may be necessary to separate out some objects due to special performance requirements and the need to monitor access to the object closely.

Buffer pool sizing

Now that the buffer pools are above the bar, the previous restrictions on size have effectively been lifted. However, the consequences are now felt across the whole LPAR and so any change of size must be discussed and agreed with those responsible for monitoring and tuning the use of storage at the LPAR level. Even the use of PAGEFIX=YES should be agreed because the pool cannot then be paged out under any circumstances. This can be an issue during times of needing large amounts of real storage, such as when taking a dump. A sign of overcommitting storage can be seen when the PAGE-INS REQUIRED for either read or write in the buffer pool display is greater than 1%. Some non-zero values are to be expected, that is, on first frame reference. This counter is always zero if PGFIX=YES. If this occurs in contiguous statistics intervals in a pool with non-zero I/O, then either more storage should be allocated to the LPAR or the buffer pools should be reduced in size. The residency time can be used as a simple way to balance the size of the pools relative to each other by trial and error.

There is one constraint on buffer pool size that data sharing users should bear in mind. This is the work that must be done by DB2 when a page set changes to group buffer pool dependent. DB2 registers all pages in the local buffer pool for the subject object into the group buffer pool, which is achieved with a scan of the local virtual pool. The cost is therefore proportional to the size of the pool and the rate of page set registration.

The number of page sets being registered can be gauged by examining the rate of page set opens (DB2 PM Statistics report NUMBER OF DATASET OPENS or Field Name: QBSTDSO), which should be less than 1 per second, and also the rate of pseudo close (DB2 PM DELETE NAME counter for the group buffer pool; field QBGLDN), which should be less than 0.25 per second. The page set CLOSE YES/NO parameter and the PCLOSEN and PCLOSET DSNZPARMs can be used to adjust the rate of data set close and pseudo close. Based on feedback from customers, it is advisable to monitor the above counters and to limit a local pool that contains page sets which can become group buffer pool dependent to around 1 GB unless monitoring confirms that the open/close rate is below the suggested limit.

Virtual Pool Sequential Threshold (VPSEQT)

The purpose of this parameter is to restrict the amount of storage in a buffer pool that can be used by sequential pages. Intuitively, a random page has more value than a sequential page and so should be given a higher chance of survival in the buffer pool by setting the value for VPSEQT lower than 100%. The default for this parameter is 80%, which was chosen when pools were relatively small. Now that pools can be much larger, up to one or more gigabytes, the parameter may need to be reduced.

One way to determine the value is to monitor the random versus sequential residency time. If the values are equal then it is unlikely that the threshold is being hit, and therefore random and sequential pages are being valued equally when the choice is made about which page should be stolen next. If the threshold is set such that the ratio of random to sequential page residency time is between 2 and 5 times and that the random page residency time is above 5 minutes for the critical periods of time, typically the on-line day, and the sequential is above 1 minute, then intuitively this seems a reasonable balance.

Using the equations on residency time defined previously, it is simple to define a formula for calculating a value for VPSEQT. If we let F represent the ratio of random to sequential residency time, then

> Random page residency time = F * Sequential page residency time

which implies:

> (buffer pool size * (1-VPSEQT/100) / synchronous pages read per second) = F * (buffer pool size * (VPSEQT/100) / asynchronous pages read per second)

Rearranging this equation results in:

> VPSEQT = 100 * asynchronous pages read per second / (F * synchronous pages read per second + asynchronous pages read per second)

So by choosing a value for F, it is a simple calculation to derive a value for VPSEQT. This formula will work best for those pools where the buffer pool size is much smaller than the sum of all pages of all objects assigned to that pool and should not be used for data in memory, sort/work, or pools for objects needing customized settings.

One other important consideration for VPSEQT is the effect on prefetch quantity when using DB2 9. The increase in prefetch size for sequential prefetch and list prefetch for LOBs from 32 to 64 for the 4 KB pools applies when:

VPSIZE * VPSEQT >= 40,000

With the default of 80% you would need to have a pool of at least 50,000 pages. If VPSEQT was halved to 40%, then the size would need to be 100,000 pages to get this benefit. Furthermore, the increase from 64 to 128 pages for utility sequential prefetch for the 4 KB pools applies when:

VPSIZE * VPSEQT >= 80,000

So with the default of 80% you would need to have a pool of at least 100,000 pages. If VPSEQT was halved to 40%, then the size would need to be 200,000 pages to get this benefit. Note that these increases do not apply to dynamic prefetch nor to non-LOB list prefetch.

Setting VPSEQT=0 disables the initiation of all prefetch engines as discussed in the section on using a pool for data in memory objects; however, this setting also disables parallelism.

Vertical Deferred Write Queue Threshold (VDWQT)

The purpose of this parameter is to trigger the writing of updated pages out of the buffer pool to disk storage. Unlike the DWQT, where the default of 30% is a good starting point, the choice of VDWQT is more complex because it is dependent on the workload. The recommendation is to start with a default setting of 5% and then adjust according to the following factors.

Increase the value for:

- In-memory index or data buffer pools
- Fewer pages written per unit of time
- Reduced frequency of same set of pages written multiple times
- Reduced page latch contention (when page is being written, first update requestor waits on “other write I/O”, subsequent updaters wait on page latch). The page latch suspension time is reported in the field PAGE LATCH in the accounting report.
- Reduced forced log writes (log records must be written ahead of updated pages)
- Random insert, update, or delete operations

Decrease the value for:

- Reduced total of updated pages in pool so more buffers are available for stealing
- Reduce burst of write activity at checkpoint intervals
- More uniform and consistent response time
- Better performance of the buffer pool as a whole when writes are predominantly for sequential or skip sequential insert of data
- For data sharing, free page p-lock sooner, especially NPI index page p-lock

To page fix or not to page fix

The recommendation is to use long term page fix (PGFIX=YES) for any pool with a high buffer I/O intensity. The buffer intensity can be measured by:

Buffer Intensity = (pages read + pages written) / buffer pool size

where:

pages read = total number of pages read in a given statistics interval, including synchronous read and sequential, dynamic and list prefetch.
(trace fields QBSTRIO + QBSTSPP + QBSTDPP + QBSTLPP)

pages written = total number of pages written in a given statistics interval including synchronous write and deferred write.
(trace field QBSTPWS)

It cannot be emphasized too strongly that using PAGEFIX=YES has to be agreed with whomever manages the z/OS storage for the LPAR, because running out of storage can have catastrophic effects for the whole of z/OS.

It is worth remembering that any pool where I/O is occurring will utilize all the allocated storage. This is why the recommendation is to use PGFIX=YES for DB2 V8 and above for any pool where I/Os occur, provided all virtual frames above and below the 2 GB bar are fully backed by real memory and DB2 virtual buffer pool frames are not paged out to auxiliary storage. If I/Os occur and PGFIX=NO then it is possible that the frame will have been paged out to auxiliary storage by z/OS and therefore two synchronous I/Os will be required instead of one. This is because both DB2 and z/OS are managing storage on a least recently used basis. The number of page-ins for read and write can be monitored either by using the DISPLAY BUFFERPOOL command to display messages DSNB411I and DSNB420I or by producing a Statistics report.

There is also a CPU overhead for PGFIX=NO because DB2 has to fix the page in storage before the I/O and unfix afterwards. Some customers have seen measurable reductions in CPU by setting PAGEFIX=YES, hence the general recommendation is to use YES for this parameter. Note PGFIX=YES is also applicable to GBP reads and writes. The disadvantage of PGFIX=YES is that the page cannot be paged out under any circumstances. This could cause paging to occur on other applications to their detriment should a need arise for a large number of real frames, such as during a dump.

Measurement

No tuning should be attempted until a solid base of reliable data is available for comparison. The simplest measures are the getpage and I/O rates. The number of getpages per second is a measure of the amount of work being done. If the intention is to reduce I/O, then by measuring the getpages per I/O before and after any change, an estimate can be made of the number of I/Os saved. Another metric is the wait time due to synchronous I/O, which is tracked in the accounting trace data class 3 suspensions.

The CPU cost of synchronous I/O is accounted to the application and so the CPU is normally credited to the agent thread. However, asynchronous I/O is accounted for in the DB2 address space DBM1 SRB measurement. The DB2 address space CPU consumption is reported in the DB2 statistics trace which can be captured and displayed by a tool such as IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS.

The author has developed a tool written in the REXX programming language that can be used to collect the data from a DISPLAY BUFFERPOOL. The program calculates the hit ratios, page residency time and other statistics that can be used to judge which pools may need adjustment or further investigation. The tool is available as additional material of this document.

Conclusion

Buffer pool tuning is an important part of the work of a database administrator. Simple techniques are available that can be used to monitor the performance of the buffer pools and then tune it by adjusting the parameters, which can significantly reduce both CPU and elapsed times.

Appendix A – Results from customer studies

These results are intended to illustrate the types of data analysis that can be performed based on the output from the various traces and displays provided by DB2. It is not expected that all users will find the same results.

Residency time and reread count

Table 1 shows the relationship between residency time and reread ratio. Notice that the buffer pools with the lowest residency times are the ones with the highest reread count. The calculations were based on a 5-minute sample collected during the middle of the day from one member of a data sharing group.

Table 1 Residency time and reread count

BPID	Virtual pool size Pages	VPSEQT (%)	Getpage	Synchronous page reads	Asynchronous page reads	Hit Ratio System	Hit Ratio Application	Random Page Residency Time (sec.)	Random Page Reread Count
BP0	40000	80	629376	10364	24529	94%	98%	344	13
BP1	190000	60	2621897	121035	171768	89%	95%	195	3444
BP2	230000	40	6007582	144475	814305	84%	98%	287	2468
BP3	20000	50	699145	474	978	100%	100%	6329	2
BP7	40000	100	585630	0	1114	100%	100%	n/a[a]	0
BP9	90000	50	105315	9781	13614	78%	91%	1380	0
BP10	75000	50	23314	7499	186	67%	68%	2928	0
BP15	65000	50	27297	527	2099	90%	98%	18501	0
BP16	110000	50	156185	771	637	99%	100%	23438	0
BP32	1000	80	12013	128	5	99%	99%	2256	36

a. The random page residency time is not applicable for this pool because the number of synchronous pages read during the period was zero and so resulted in a value of infinity.

Data in memory buffer pool

The chart in Figure 1 shows the potential value of isolating small high-use objects into a separate buffer pool. The chart was derived from a 15-second IFCID 198 trace captured at hourly intervals across a 24-hour period. The page sets were then sorted into a descending sequence by the number of getpages per page in the page set.

The chart shows that a buffer pool of just 10,000 pages could be defined to hold 140 page sets, which would account for 18% of the total getpages. If the size of the pool were increased to 50,000 pages, the getpage count would account for 40% of the total.

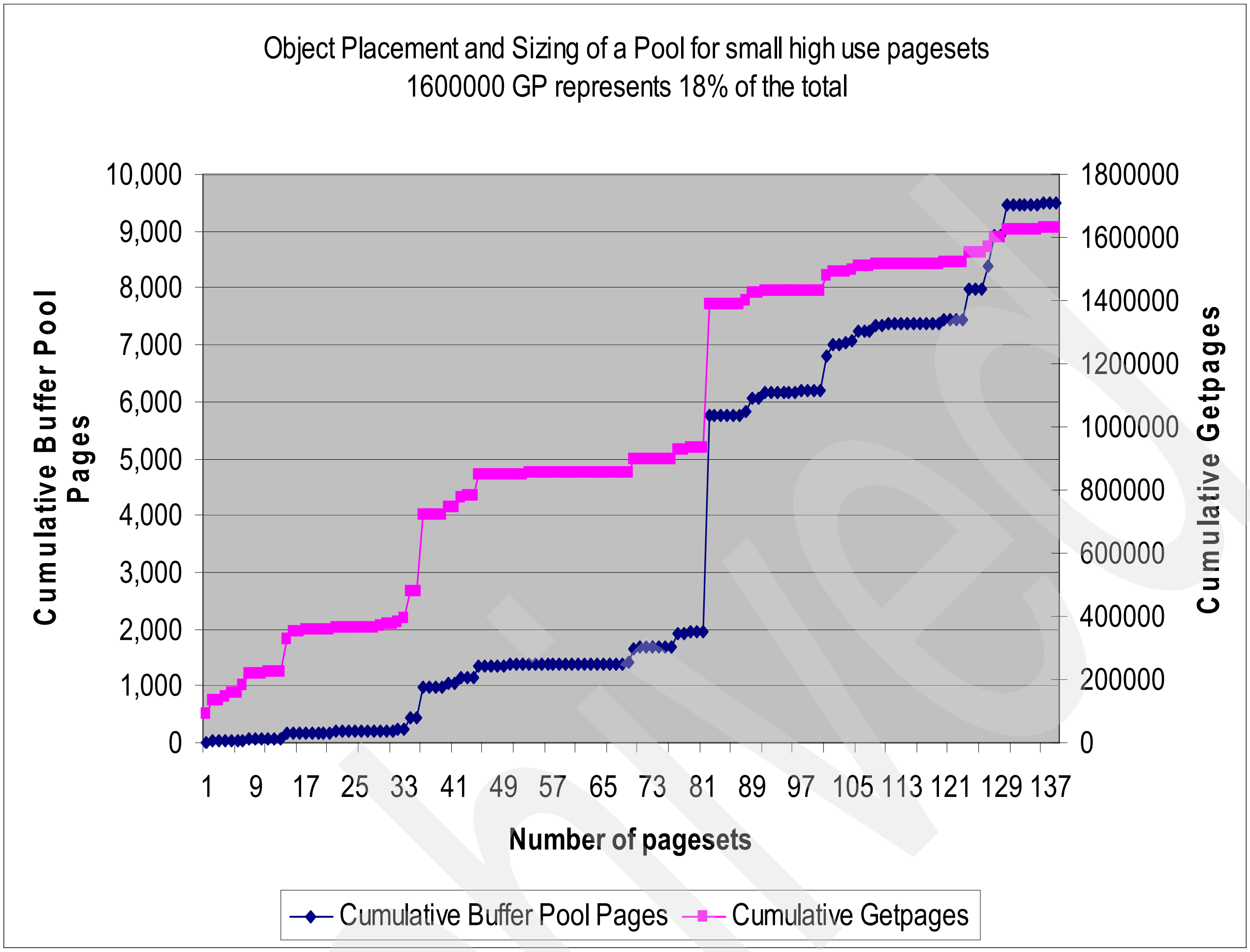

Figure 1   Data in memory buffer pool

Object placement: Analysis by access method

Table 2 shows the breakdown by index space and table space of the number of getpages for each page set, grouped by the proportion of random to sequential access for each page set. By far the majority of page sets are randomly accessed with not many page sets having both random and sequential access. Those few that are sequentially accessed have a proportionately much higher number of getpages per page set than those that are randomly accessed.

Table 2   Object placement: breakdown by sequential versus random access

	Index space			Table space			Total		
Random %	Object Count	Getpage Random	Getpage Sequential+ RIDlist	Object Count	Getpage Random	Getpage Sequential+ RIDlist	Object Count	Getpage Random	Getpage Sequential+ RIDlist
0	33	434	119736	62	6836	1528143	95	7270	1647879
10	6	1240	14193	18	76269	625486	24	77509	639679
20	5	41006	160789	13	35780	137074	18	76786	297863
30	0	0	0	10	488	1013	10	488	1013
40	0	0	0	8	6698	9665	8	6698	9665

50	5	756	697	34	8800	8273	39	9556	8970
60	4	278	175	6	2319	1553	10	2597	1728
70	3	2079	963	13	7665	3573	16	9744	4536
80	3	1314	366	7	14516	4307	10	15830	4673
90	1	99378	9811	13	16892	2006	14	116270	11817
100	2081	1959726	976	1140	1291516	10885	3221	3251242	11861
Total	2141	2106211	307706	1324	1467779	2331978	3465	3573990	2639684

The chart supports the proposal that separation by access type is worthwhile because most page sets analyzed in this sample are either sequentially or randomly accessed. Separation of the pure sequentially accessed objects would mean that better settings of VPSEQT and VPSIZE could be used to optimize performance for the resulting pools.

The chart was derived from a 15-second IFCID 198 trace captured at hourly intervals across a 24-hour period.

Relationship of work rate to the residency time

The chart in Figure 2 shows the relationship between the buffer pool pages per getpage per second plotted against the random page residency time for a set of buffer pools. The advantage of the metric “buffer pool pages per getpage per second” is that it removes the actual size of the buffer pool from consideration and focuses on the workload being processed for each page in the pool.

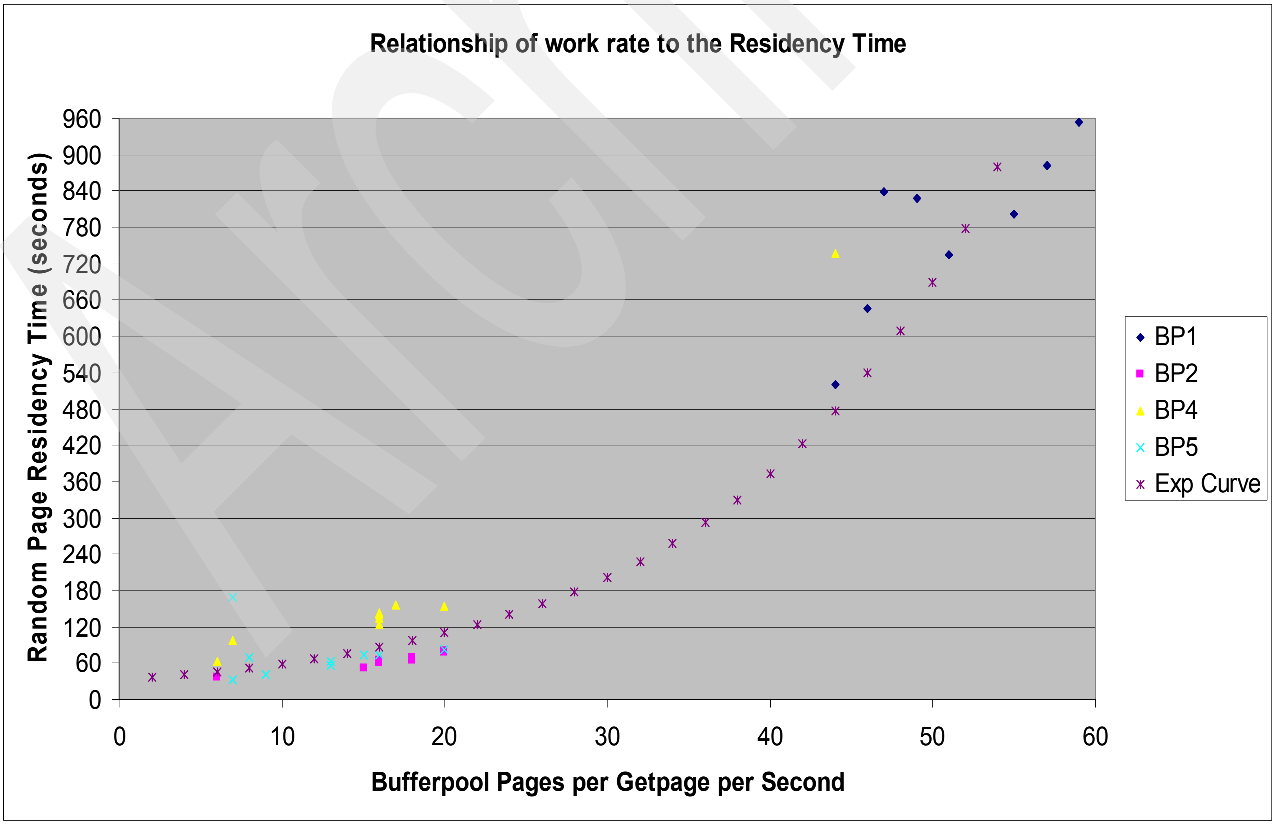

Figure 2 Work rate versus residency time

The data was analyzed for the working day (09:00 to 17:30) for a 9-way data sharing group. Each point represents the value for one member of the group. Note that not all points are shown for all members because some points lie off the scale of this graph.

The results confirm the common sense notion that the higher the degree of work, represented by the lower value of buffer pool pages per getpage per second, the lower the residency time. The chart also shows a slight curve upwards, which again conforms to the notion that as the size of the pool is increased, doubled say, the random page residency time more than doubles such that for some finite buffer pool size the residency time would tend to infinity. The exponential curve shown has a good degree of fit and results in this equation for this sample of data:

Random page residency time = 32.14 * 1.06 ^ BP Pages per GP per sec.

From the chart you can deduce that to achieve the objective of 5 minutes random page residency time would require a buffer pool size of 36 times the getpage per second rate for the pool.

The data was analyzed for the working day from 09:00 to 17:00 at hourly intervals (see Figure 3) so each point represents the value for one member of the group for a 1-hour period.

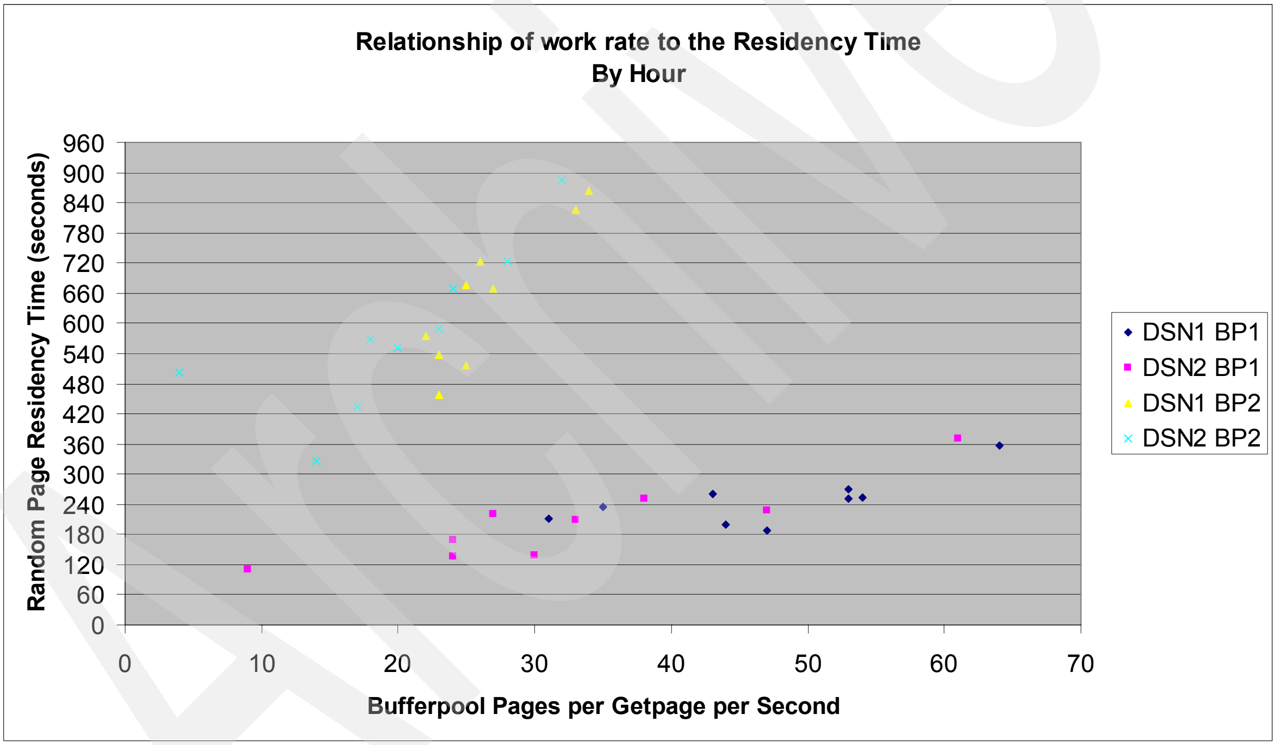

Figure 3 Work rate versus residency time by hour

These results show quite clearly that the profile of work in BP1 is very different from BP2 and that adding storage to BP2 is more likely to increase the residency time than adding storage to BP1. The different profile is, not surprisingly, due to using BP1 for data and BP2 for indexes. The relationship in this case appears to be more linear than the previous example.

Appendix B – Sample DISPLAY BUFFERPOOL

A sample output of the **- DISPLAY BUFFERPOOL** command is shown in Figure 4.

```
-DISPLAY BUFFERPOOL(BP0) DETAIL

DSNB401I  -DB41 BUFFERPOOL NAME BP0, BUFFERPOOL ID 0, USE COUNT 398
DSNB402I  -DB41 BUFFER POOL SIZE = 4000 BUFFERS  AUTOSIZE = NO
             ALLOCATED       =     4000   TO BE DELETED    =        0
             IN-USE/UPDATED  =      138   BUFFERS ACTIVE   =     4000
DSNB406I  -DB41 PGFIX ATTRIBUTE -
             CURRENT = NO
             PENDING = NO
           PAGE STEALING METHOD = LRU
DSNB404I  -DB41 THRESHOLDS -
            VP SEQUENTIAL    = 80
            DEFERRED WRITE   = 30   VERTICAL DEFERRED WRT  =  5,  0
            PARALLEL SEQUENTIAL =50   ASSISTING PARALLEL SEQT=  0
DSNB409I  -DB41 INCREMENTAL STATISTICS SINCE 18:56:59 AUG 26, 2009
DSNB411I -DB41 RANDOM GETPAGE  = 2096879 SYNC READ I/O (R) = 7731
           SEQ.   GETPAGE    =  9095868 SYNC READ I/O (S) =      172
           DMTH HIT          =        0 PAGE-INS REQUIRED =        0
DSNB412I  -DB41 SEQUENTIAL PREFETCH -
             REQUESTS        =      479   PREFETCH I/O     =      317
             PAGES READ      =     6576
DSNB413I  -DB41 LIST PREFETCH -
             REQUESTS        =     6208   PREFETCH I/O     =       15
             PAGES READ      =       16
DSNB414I  -DB41 DYNAMIC PREFETCH -
             REQUESTS        =    43886   PREFETCH I/O     =      866
             PAGES READ      =    25273
DSNB415I  -DB41 PREFETCH DISABLED -
             NO BUFFER       =        0   NO READ ENGINE   =        0
DSNB420I -DB41 SYS PAGE UPDATES = 1999996 SYS PAGES WRITTEN = 370978
           ASYNC WRITE I/O   =    28336 SYNC WRITE I/O    =    26628
           PAGE-INS REQUIRED =        0
DSNB421I -DB41 DWT HIT      =    0 VERTICAL DWT HIT = 1390
DSNB440I  -DB41 PARALLEL ACTIVITY -
            PARALLEL REQUEST =        0  DEGRADED PARALLEL=        0
DSNB441I  -DB41 LPL ACTIVITY -
            PAGES ADDED      =        0
DSN9022I  -DB41 DSNB1CMD '-DISPLAY BUFFERPOOL' NORMAL COMPLETION
```

Figure 4   DISPLAY BUFFERPOOL output

Appendix C – Sample statistic report

The statistic report is shown in Example 1.

Example 1   A sample statistic report

```
BP0    READ OPERATIONS       QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
BPOOL HIT RATIO (%)             99.99
BPOOL HIT RATIO (%) SEQU        99.99
BPOOL HIT RATIO (%) RANDOM     100.00

GETPAGE REQUEST              78881.00   398.91  4151.63    32.68
GETPAGE REQUEST-SEQUENTIAL   35618.00   180.13  1874.63    14.75
GETPAGE REQUEST-RANDOM       43263.00   218.79  2277.00    17.92

SYNCHRONOUS READS                0.00     0.00     0.00     0.00
SYNCHRON. READS-SEQUENTIAL       0.00     0.00     0.00     0.00
SYNCHRON. READS-RANDOM           0.00     0.00     0.00     0.00
GETPAGE PER SYN.READ-RANDOM       N/C

SEQUENTIAL PREFETCH REQUEST      0.00     0.00     0.00     0.00
SEQUENTIAL PREFETCH READS        0.00     0.00     0.00     0.00
PAGES READ VIA SEQ.PREFETCH      0.00     0.00     0.00     0.00
S.PRF.PAGES READ/S.PRF.READ       N/C
LIST PREFETCH REQUESTS           0.00     0.00     0.00     0.00
LIST PREFETCH READS              0.00     0.00     0.00     0.00
PAGES READ VIA LIST PREFTCH      0.00     0.00     0.00     0.00
L.PRF.PAGES READ/L.PRF.READ       N/C
DYNAMIC PREFETCH REQUESTED      86.00     0.43     4.53     0.04
DYNAMIC PREFETCH READS           2.00     0.01     0.11     0.00
PAGES READ VIA DYN.PREFETCH      4.00     0.02     0.21     0.00
D.PRF.PAGES READ/D.PRF.READ      2.00

PREF.DISABLED-NO BUFFER          0.00     0.00     0.00     0.00
PREF.DISABLED-NO READ ENG        0.00     0.00     0.00     0.00
PAGE-INS REQUIRED FOR READ       4.00     0.02     0.21     0.00

BP0    WRITE OPERATIONS      QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
BUFFER UPDATES                5335.00    26.98   280.79     2.21
PAGES WRITTEN                    0.00     0.00     0.00     0.00
BUFF.UPDATES/PAGES WRITTEN        N/C

SYNCHRONOUS WRITES               0.00     0.00     0.00     0.00
ASYNCHRONOUS WRITES              0.00     0.00     0.00     0.00

PAGES WRITTEN PER WRITE I/O       N/C

HORIZ.DEF.WRITE THRESHOLD        0.00     0.00     0.00     0.00
VERTI.DEF.WRITE THRESHOLD        0.00     0.00     0.00     0.00
DM THRESHOLD                     0.00     0.00     0.00     0.00
WRITE ENGINE NOT AVAILABLE       0.00     0.00     0.00     0.00
PAGE-INS REQUIRED FOR WRITE      0.00     0.00     0.00     0.00
```

Appendix D - Hit ratio and system page residency time

Hit ratio and system page residency time calculation based on the IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS database is shown in Example 2.

Example 2 Hit ratio and system page residency time calculation

```
SELECT A.SUBSYSTEM_ID AS SSID
  , CASE
      WHEN A.BP_ID BETWEEN 0 AND 9 THEN
        'BP' CONCAT SUBSTR(DIGITS(A.BP_ID), 10, 1)
      WHEN A.BP_ID BETWEEN 10 AND 49 THEN
        'BP' CONCAT SUBSTR(DIGITS(A.BP_ID), 9, 2)
      WHEN A.BP_ID = 80 THEN 'BP32K'
      WHEN A.BP_ID BETWEEN 81 AND 89 THEN
        'BP32K' CONCAT SUBSTR(DIGITS(A.BP_ID - 80),10, 1)
      WHEN A.BP_ID BETWEEN 100 AND 109 THEN
         'BP8K' CONCAT SUBSTR(DIGITS(A.BP_ID - 100), 10, 1)
      WHEN A.BP_ID BETWEEN 120 AND 129 THEN
         'BP16K' CONCAT SUBSTR(DIGITS(A.BP_ID-120),10,1)
    END AS BUFFER_POOL_NAME
  , DEC(A.VIRTUAL_BUFFERS,7,0) AS VPOOL_SIZE_PAGES
  , SUBSTR(CHAR(A.INTERVAL_TSTAMP),1,16) AS INTERVAL_TIME_STAMP
  , DEC(A.INTERVAL_ELAPSED,6,0) AS INTERVAL_ELAP_SEC
  , DEC(A.GET_PAGE/A.INTERVAL_ELAPSED,7,0)
      AS GET_PAGE_PER_SEC
  , DEC(A.SYNC_READ_IO/A.INTERVAL_ELAPSED,7,0)
      AS SYNC_READ_IO_PER_SEC
  , DEC((A.SEQ_PREFETCH_READ+A.LIST_PREFETCH_READ
         +A.DYN_PREFETCH_READ)/A.INTERVAL_ELAPSED,7,0)
      AS ASYNC_READ_IO_PER_SEC
  , DEC((A.SEQ_PREFETCH_PAGE+A.LIST_PREFETCH_PAGE
         +A.DYN_PREFETCH_PAGE)/A.INTERVAL_ELAPSED,7,0)
      AS ASYNC_PAGE_READ_PER_SEC
  , CASE GET_PAGE WHEN 0 THEN 100
      ELSE DEC(100*(GET_PAGE-(A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
           +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE))
           /GET_PAGE,8,2)
    END AS SYSTEM_HIT_RATIO
  , CASE GET_PAGE WHEN 0 THEN 100
      ELSE DEC(100*(GET_PAGE-A.SYNC_READ_IO)/GET_PAGE,8,2)
    END AS APPN_HIT_RATIO
  , CASE A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
         +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE
      WHEN 0 THEN 99999
      ELSE MIN(99999,INT(A.INTERVAL_ELAPSED*A.VIRTUAL_BUFFERS/
               (A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
               +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE) ) )
    END AS SYS_RES_TIME
  , DEC((A.READ_PAGE_INS_REQ+A.WRITE_PAGE_INS_REQ)
      /A.INTERVAL_ELAPSED,8,1)
      AS PAGEIN_RDWRT_PER_SEC
  , DEC((GET_PAGE-GET_PAGE_SEQ)/A.INTERVAL_ELAPSED,8,1)
      AS RGET_PAGE_PER_SEC
  , DEC(GET_PAGE_SEQ/A.INTERVAL_ELAPSED,8,1)
      AS SGET_PAGE_PER_SEC
  , DEC(A.SYNC_READ_SEQ/A.INTERVAL_ELAPSED,7,0)
      AS SYNC_READ_SEQ_PER_SEC
FROM DB2PMPDB.DB2PM_STAT_BUFFER A
```

Note that the system page residency can be calculated but not the random or sequential page residency time. This is because the value of VPSEQT is not captured in the statistics record. It is possible to capture this information in the system parameter table DB2PMSYSPAR_202 and then join the tables, as shown in Example 3.

Example 3 Random or sequential page residency time

```
SELECT A.SUBSYSTEM_ID AS SSID
  , CASE
      WHEN A.BP_ID BETWEEN 0 AND 9 THEN
        'BP' CONCAT SUBSTR(DIGITS(A.BP_ID), 10, 1)
      WHEN A.BP_ID BETWEEN 10 AND 49 THEN
        'BP' CONCAT SUBSTR(DIGITS(A.BP_ID), 9, 2)
      WHEN A.BP_ID = 80 THEN 'BP32K'
      WHEN A.BP_ID BETWEEN 81 AND 89 THEN
        'BP32K' CONCAT SUBSTR(DIGITS(A.BP_ID - 80),10, 1)
      WHEN A.BP_ID BETWEEN 100 AND 109 THEN
         'BP8K' CONCAT SUBSTR(DIGITS(A.BP_ID - 100), 10, 1)
      WHEN A.BP_ID BETWEEN 120 AND 129 THEN
         'BP16K' CONCAT SUBSTR(DIGITS(A.BP_ID-120),10,1)
    END AS BUFFER_POOL_NAME
  , DEC(A.VIRTUAL_BUFFERS,7,0) AS VPOOL_SIZE_PAGES
  , SUBSTR(CHAR(A.INTERVAL_TSTAMP),1,16) AS INTERVAL_TIME_STAMP
  , DEC(A.INTERVAL_ELAPSED,6,0) AS INTERVAL_ELAP_SEC
  , B.VPOOL_SSEQ_THRESH AS VP_SEQT_PERC
  , B.VPOOL_DWT_THRESH AS VP_DWT_PERC
  , B.VPOOL_VDWT_THRESH AS VP_VDWT_PERC
  , B.VPOOL_VDWT_THR_BUF AS VP_VDWT_PGS
  , B.PGFIX_ATTRIBUTE AS PAGE_FIX
  , B.PSTEAL_METHOD AS PAGE_STEAL
  , DEC(A.GET_PAGE/A.INTERVAL_ELAPSED,7,0)
      AS GET_PAGE_PER_SEC
  , DEC(A.SYNC_READ_IO/A.INTERVAL_ELAPSED,7,0)
      AS SYNC_READ_IO_PER_SEC
  , DEC((A.SEQ_PREFETCH_PAGE+A.LIST_PREFETCH_PAGE
         +A.DYN_PREFETCH_PAGE)/A.INTERVAL_ELAPSED,7,0)
      AS ASYNC_READ_IO_PER_SEC
  , CASE GET_PAGE
      WHEN 0 THEN 100
      ELSE DEC(100*(GET_PAGE-(A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
           +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE))
           /GET_PAGE,8,2)
    END AS SYSTEM_HIT_RATIO
  , CASE GET_PAGE
      WHEN 0 THEN 100
      ELSE DEC(100*(GET_PAGE-A.SYNC_READ_IO)/GET_PAGE,8,2)
    END AS APPN_HIT_RATIO
  , CASE A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
         +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE
      WHEN 0 THEN 99999
      ELSE MIN(99999,INT(A.INTERVAL_ELAPSED*A.VIRTUAL_BUFFERS/
               (A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
               +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE) ) )
    END AS SYS_RES_TIME
  , CASE A.SEQ_PREFETCH_PAGE
         +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE
      WHEN 0 THEN 99999 ELSE
      CASE
        WHEN 100*( A.SEQ_PREFETCH_PAGE+A.LIST_PREFETCH_PAGE
             +A.DYN_PREFETCH_PAGE) >  B.VPOOL_SSEQ_THRESH
             *(A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
+A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE)
        THEN MIN(99999, INT(A.INTERVAL_ELAPSED
             *(B.VPOOL_SSEQ_THRESH*A.VIRTUAL_BUFFERS/100) /
             (A.SEQ_PREFETCH_PAGE+A.LIST_PREFETCH_PAGE
             +A.DYN_PREFETCH_PAGE) ) )
        ELSE MIN(99999,INT(A.INTERVAL_ELAPSED*A.VIRTUAL_BUFFERS/
             (A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
             +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE) ) )
      END
    END AS ASYNC_RES_TIME
  , CASE A.SYNC_READ_IO
      WHEN 0 THEN 99999 ELSE
      CASE
        WHEN 100*( A.SEQ_PREFETCH_PAGE+A.LIST_PREFETCH_PAGE
             +A.DYN_PREFETCH_PAGE) > B.VPOOL_SSEQ_THRESH
             *(A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
             +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE)
        THEN MIN(99999, INT(A.INTERVAL_ELAPSED
             *((100-B.VPOOL_SSEQ_THRESH)*A.VIRTUAL_BUFFERS/100) /
             (A.SYNC_READ_IO) )  )
        ELSE MIN(99999,INT(A.INTERVAL_ELAPSED*A.VIRTUAL_BUFFERS/
             (A.SYNC_READ_IO+A.SEQ_PREFETCH_PAGE
             +A.LIST_PREFETCH_PAGE+A.DYN_PREFETCH_PAGE) ) )
      END
    END AS SYNC_RES_TIME
  , DEC((A.READ_PAGE_INS_REQ+A.WRITE_PAGE_INS_REQ)
      /A.INTERVAL_ELAPSED,8,1)
      AS PAGEIN_RDWRT_PER_SEC
  , DEC((GET_PAGE-GET_PAGE_SEQ)/A.INTERVAL_ELAPSED,8,1)
      AS RGET_PAGE_PER_SEC
  , DEC(GET_PAGE_SEQ/A.INTERVAL_ELAPSED,8,1)
      AS SGET_PAGE_PER_SEC
  , DEC(A.SYNC_READ_SEQ/A.INTERVAL_ELAPSED,7,0)
      AS SYNC_READ_SEQ_PER_SEC
FROM DB2PMPDB.DB2PM_STAT_BUFFER A
   , DB2PMPDB.DB2PMSYSPAR_202 B
WHERE B.MEMBER_NAME = A.MEMBER_NAME
  AND B.SUBSYSTEM_ID = A.SUBSYSTEM_ID
  AND B.GRPNAME = A.GROUP_NAME
  AND A.INTERVAL_TSTAMP = TRUNC_TIMESTAMP(B.TIMESTAMP,'MI') +
            ( 5*INT(ROUND(MINUTE(B.TIMESTAMP)/5.0,0))
                - MINUTE(B.TIMESTAMP)) MINUTES
  AND B.LOCAL_LOCATION = A.LOCAL_LOCATION
  AND B.BUFFERPOOL_ID =
    CASE
      WHEN A.BP_ID BETWEEN 0 AND 9 THEN
        'BP' CONCAT  SUBSTR(DIGITS(A.BP_ID), 10, 1)
      WHEN BP_ID BETWEEN 10 AND 49 THEN
        'BP'  CONCAT SUBSTR(DIGITS(A.BP_ID), 9, 2)
      WHEN A.BP_ID = 80 THEN 'BP32K'
      WHEN  A.BP_ID BETWEEN 81 AND 89 THEN
        'BP32K' CONCAT SUBSTR(DIGITS(A.BP_ID - 80),10, 1)
      WHEN A.BP_ID BETWEEN 100 AND 109 THEN
        'BP8K'  CONCAT SUBSTR(DIGITS(A.BP_ID - 100), 10, 1)
      WHEN A.BP_ID BETWEEN 120 AND 129 THEN
        'BP16K' CONCAT SUBSTR(DIGITS(A.BP_ID-120),10,1)
    END
```

Who wrote this Redpaper

This paper was produced by:

Mike Bracey is a DB2 for z/OS systems engineer based in the United Kingdom. He has been with IBM for over 26 years and has worked in various capacities as a DB2 branch specialist, DB2 consultant for designing, implementing and tuning applications, and DB2 beta program manager for Europe. He has extensive practical DB2 experience, gained over 20 years, of DB2 application design, data modelling, and performance monitoring and tuning. He is currently a member of the IBM Software Business in Europe providing technical support for DB2 for z/OS, DB2 Tools, and related topics. His current areas of interest are optimization, buffer pool tuning, and XML.

Acknowledgements

We would like to acknowledge the contribution of Mike Barnard, who triggered his interest in this topic with a presentation at a Guide meeting in 1997 on the subject of buffer pool tuning, where he outlined the concept of measuring the reread ratio. The idea of page residency time came from John Campbell who contributed greatly to the content of this paper. Akira Shibamiya and Steve Turnbaugh of DB2 Development have also given much valuable advice and suggested many corrections. Paolo Bruni of the ITSO provided the impetus to publish as an IBM Redpaper™ and made it happen.

We would also like to thank the many customers who contributed by providing sample data and by allowing the results of the analysis to be published as illustration of the type of analyses that can be done. We recognize that no two customers' use of DB2 is alike; however, we believe that there are common themes and lessons that can be learned that are applicable to all users.

References

- Tuning DB2 at CGU Life by Mike Barnard

  http://www.gseukdb2.org.uk/techinfo.htm
- DB2 Buffer pool Sizing by Mike Barnard

  http://www.gseukdb2.org.uk/techinfo.htm
- What's new in DB2 for z/OS Buffer Pool Management by Jim Teng

  http://www.ibm.com/support/docview.wss?rs=64&uid=swg27000116
- It's Time to Re-think Your DB2 Buffer Pool Strategy by Martin Hubel

  http://www.responsivesystems.com/papers/peoplesoft/ItsTimetoRethinkYourDB2BufferPoolStrategy.pdf
- DB2 z/OS Buffer Pool Management by Mike Bracey

  http://www.gseukdb2.org.uk/techinfo.htm
- *DB2 for OS/390 Capacity Planning*, SG24-2244-00

  Appendix C documents a technique for calculating the reread ratio using IBM tools.

  http://www.redbooks.ibm.com/redbooks/pdfs/sg242244.pdf

- *DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide,* SC18-9851-05
- *DB2 Version 9.1 for z/OS Command Reference,* SC18-9844-03
- *Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS V4R1 Report Reference,* SC18-9984-02

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document REDP-4604-00 was created or updated on October 25, 2009.

Send us your comments 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 U.S.A.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DB2®
IBM®
OMEGAMON®
Redpaper™
Redbooks (logo) ®
System z®
Tivoli®
z/OS®

Other company, product, or service names may be trademarks or service marks of others.