

Benefits of Configuring More Memory in the IBM z/OS Software Stack

Mark Wisniewski

Brenda Beane

David Betten

Clark Goodrich

Akiko Hoshikawa

David Herr

Catherine Moxey

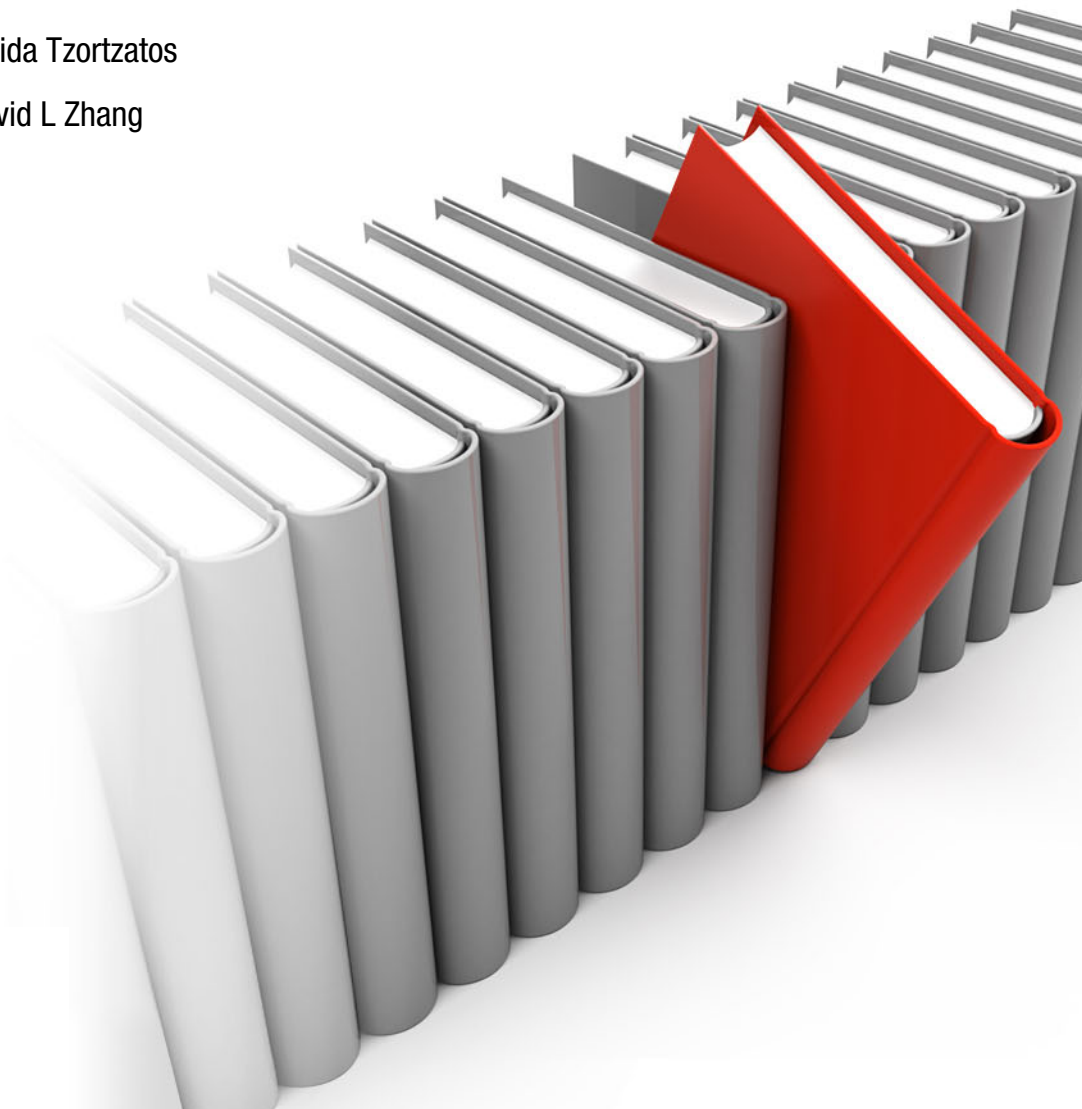
Tony Sharkey

Pete Siddall

Robin Tanenbaum

Elpida Tzortzatos

David L Zhang





International Technical Support Organization

**Benefits of Configuring More Memory in the IBM z/OS
Software Stack**

January 2017

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

Second Edition (January 2017)

This edition applies to Version 2, Release 2, of IBM z/OS (product number 5650-ZOS).

This document was created or updated on February 17, 2017.

© Copyright International Business Machines Corporation 2016, 2017. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	v
Trademarks	vi
Preface	vii
Authors	vii
Now you can become a published author, too!	ix
Comments welcome	ix
Stay connected to IBM Redbooks	x
Summary of changes	xi
January 2017, Second Edition	xi
Chapter 1. Introduction	1
1.1 Improving SAN Volume Controller memory dump capture with large memory	2
Chapter 2. DB2 buffer pools and related tuning	5
2.1 DB2 buffer pools	6
2.1.1 Buffer pool simulation	8
2.2 DB2 for z/OS monitoring and tuning to using memory	10
2.2.1 DB2 and the effect of paging	10
2.2.2 Reusing DB2 threads	12
2.2.3 RELEASE(DEALLOCATE) option for persistent threads	14
2.3 IBM InfoSphere MDM with large DB2 group buffer pool	16
2.3.1 Environment	16
Chapter 3. Java and large memory	19
3.1 Background	20
3.2 IBM z13 Java large memory with large page measurements	20
3.2.1 Initial set of Java options	20
3.2.2 Measurements	21
Chapter 4. IBM MQ and CICS	23
4.1 Using larger memory with CICS	24
4.1.1 Considerations for the use of large memory with CICS	25
4.1.2 Real memory challenges for CICS	26
4.1.3 Measuring memory use by CICS	26
4.1.4 Using more real memory to open opportunities in CICS	30
4.2 IBM MQ v8 64-bit buffer pools	31
4.2.1 How to right-size IBM MQ buffer pools	34
Chapter 5. New applications and analytics	39
5.1 Cognos Dynamic Cubes	40
Chapter 6. Other benefits	43
6.1 DFSORT I/O reduction	44
6.1.1 Conclusion	51
6.2 z/OS Communications Server large memory performance results	52
Chapter 7. Conclusion	55

Related publications 57

IBM Redbooks 57

Online resources 57

Help from IBM 58

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 grant 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

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:

AIX®	IBM z13®	System z10®
CICS®	IBM z13s™	Tivoli®
CICSPIlex®	IMS™	WebSphere®
Cognos®	InfoSphere®	z Systems®
DB2®	Language Environment®	z/OS®
Hiperspace™	OMEGAMON®	z10™
IBM®	Redbooks®	z13™
IBM MobileFirst™	Redpaper™	z13s™
IBM z™	Redbooks (logo)  ®	
IBM z Systems®	RMF™	

The following terms are trademarks of other companies:

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Preface

Significant performance benefits can be realized by increasing the amount of memory that is assigned to various functions in the IBM® z/OS® software stack, operating system, and middleware products. IBM DB2® and IBM MQ buffer pools, dump services, and large page usage are just a few of the functions whose ease of use and performance can be improved when more memory is made available to them.

The following benefits can be realized:

- ▶ Reduced I/O operations
- ▶ Reduced CPU usage
- ▶ Improved transaction response time
- ▶ Potential cost reductions

Although the magnitude of these improvements can vary widely based on several factors, including potential I/Os to be eliminated, resource contention, workload, configuration, and tuning, clients must carefully consider whether their environment can benefit from the addition of memory to the software functions that are described in this IBM Redpaper™ publication.

This paper describes the performance implications of increasing memory in the following areas:

- ▶ DB2 buffer pools
- ▶ DB2 tuning
- ▶ IBM Cognos® Dynamic Cubes
- ▶ MDM with larger DB2 buffer pools
- ▶ Java heaps and Garbage Collection tuning and Java large page use
- ▶ IBM MQ v8 64-bit buffer pool tuning
- ▶ Enabling more in-memory use by IBM CICS® without paging
- ▶ TCP/IP FTP
- ▶ DFSort I/O reduction
- ▶ Fixed pages and fixed large pages

Authors

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center.

Mark Wisniewski is a Senior Technical Staff Member in the z Brand organization. He has spent over 30 years working on various aspects of high-end mainframe systems' performance.

Brenda Beane is a Senior Software Engineer at IBM in Poughkeepsie, NY. She has 33 years of experience in development and testing at IBM. For the last 19 years, she has been a member of the SAP on IBM z™ Systems performance test team, which specializes in DB2 for z/OS. Brenda holds a Bachelor of Science degree in Computer Science and Mathematics from the University of Massachusetts and a Master's degree in Computer Science from Marist College, Poughkeepsie, NY.

David Betten is a Senior Software Engineer in the IBM Cloud and Systems Performance team. He specializes in helping customers worldwide in optimizing the performance of their z/OS systems, which specializes in batch and I/O tuning. Dave has over 30 years of experience in z/OS performance, including eight years as the IBM DFSORT Performance Lead.

Clark Goodrich is a Senior Software Engineer for IBM z Systems® Performance Organization at IBM in Poughkeepsie, NY. Clark's focus is performance analysis of Java on z Systems. He was one of the original developers of the z/OS UNIX Kernel. Before joining IBM, Clark worked on Space Shuttle real-time ground systems at NASA Goddard Space Flight Center.

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. After working in parallel sysplex center in IBM Japan with Japanese customers, she joined performance team in DB2 for z/OS development in 1998. She is currently the technical lead for DB2 for z/OS performance and drives performance initiated development in new DB2 releases. She loves the hands-on working with customers, providing performance evaluation, tuning, benchmarks, designing performance features to solve customers challenges.

David Herr is a Senior Software Engineer at the IBM RTP site. David's primary areas of expertise are in z/OS Communications Server development and performance. Most recently, he developed the SMC-R function for Communications Server.

Catherine Moxey is an IBM Senior Technical Staff Member in CICS Strategy and Architecture, based at IBM Hursley near Winchester. She holds an M.A. in Chemistry from Oxford University, and has over 30 years of experience as a software engineer, 26 of those years with IBM. Her areas of expertise include CICS, z Systems, event processing, and analytics. She is a CICS Senior Technical Staff Member (STSM) for Performance and Optimization, and is the architect for Event Processing support in CICS. Catherine frequently presents at conferences, and has written a number of articles, papers, and IBM Redbooks® publications on the capabilities of CICS.

Tony Sharkey is a Performance Engineer on IBM MQ for z/OS. Tony has worked on IBM mainframes throughout his career inside and outside of IBM, but has now been in residence in the IBM Hursley Development laboratory in a performance role for 9 years.

Pete Siddall is an IBM STSM and leads software engineering for IBM MQ on z/OS. Pete has worked with IBM mainframes throughout his career. He is based in the IBM Hursley Development Laboratory in the UK.

Robin Tanenbaum is a Senior Software Engineer in the z Systems Performance Organization at IBM Poughkeepsie. She leads a team that is focused on the performance of Business Analytics applications on IBM mainframes.

Elpida Tzortzatos is a Distinguished Engineer and z Systems Architect working on IBM z/OS Core Design.

David L Zhang has been a DB2 for z/OS Performance Analyst for IBM Software Group for 11 years. He was involved in several ITSO Redbooks publications about connectivity to mainframe applications and DB2 for z/OS related performance topics for version 9, 10, and 11. He is currently working on performance evaluation of DB2 for z/OS vNext features and functions.

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

Lydia Parziale, IBM Project Manager
International Technical Support Organization, Poughkeepsie Center

Brenda Beane
David Betten
Seewah Chan
Akiko Hoshikawa
Richard Ko
Robert Miller
Dave Herr
IBM US

Chris Baker, Ian Burnett
IBM Hursley

Radoslaw Skorupka
Bank SA, Poland

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 papers to be as helpful as possible. Send us your comments about this paper 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:
<https://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>

Summary of changes

This section describes the technical changes that were made in this edition of the paper and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for Benefits of Configuring More Memory in the IBM z/OS Software Stack
as created or updated on February 17, 2017.

January 2017, Second Edition

This revision includes the following new and changed information.

New information

- ▶ Added Right-sizing MQ Buffer Pools in section 4.2.1, “How to right-size IBM MQ buffer pools” on page 34.
- ▶ Added information on using large TCP buffers for sending and receiving data in Chapter 6.2, “z/OS Communications Server large memory performance results” on page 52.
- ▶ Additional information on IBM RMF™ and DFSORT SMF type 16 analysis to assess potential benefit to DFSORT from large memory in Chapter 4.2, “IBM MQ v8 64-bit buffer pools” on page 31.



Introduction

IBM z Systems continues to deliver large memory improvements that are designed for use in new data hungry applications and for performance improvements to existing workloads. Because the IBM z13® provides an attractive price point for large amounts of real memory, it is an opportune time to evaluate what large memory can do for your installation. Large real memory on the IBM z Systems platform offers many benefits for online transaction processing (OLTP) workloads, big data analytics, in-memory databases, and performance advantages for existing work. Consider the following examples:

- ▶ CPU performance is improved with large memory when all paging is avoided. Batch workload processing time is reduced. The CPU savings that are gained when applications are no longer causing paging reduces the need for application and system redesign to meet service goals. When all of the data that is needed for transaction processing is in real memory, the CPU time per transaction is reduced.
- ▶ For OLTP workloads, large memory provides substantial latency reduction, which leads to significant response time reductions and increased transaction rates.
- ▶ Batch workloads also benefit from large memory. More memory allows increased parallelism for sorts and improve single threaded performance of complex queries. For more information about CICS and DFSORT performance gains from large memory, see Chapter 4, “IBM MQ and CICS” on page 23.
- ▶ Defining large memory systems allows analytic workloads to process big data more efficiently. Real-time availability to analytic results allows organizations to keep pace, react to trends, and identify business opportunities.
- ▶ Take advantage of large memory to define 2G large pages. DB2 and JAVA performance is improved when DB2 fixed buffer pools and JAVA HEAPS are defined to use 2G large pages. When defining and using the largest fixed memory pages, increase the system total real memory to accommodate the large pages for DB2 and JAVA. Do not reduce your 4 K memory definition. For more information about JAVA large memory, see Chapter 3, “Java and large memory” on page 19.

The use of large memory to avoid paging benefits the performance for each application on the system. Consider and plan for enough memory for SAN Volume Controller memory dumping as well. SANSAN Volume Controller memory dump capture times are improved when the data for all address space identifiers (ASIDs) that are being memory dumped is available in real memory. SAN Volume Controller memory dump capture times are reduced when enough memory is available to capture the memory dump in real memory.

1.1 Improving SAN Volume Controller memory dump capture with large memory

Avoiding real memory storage shortages during a SAN Volume Controller memory dump is critical to overall system performance and minimizing affect on the failing application. During SAN Volume Controller memory dump capture, the address spaces in the memory dump are non-dispatchable; that is, no work is occurring in these address spaces. For example, if DB2 is performing a SAN Volume Controller memory dump, no transactions can process until the memory dump capture phase for the DB2 address is complete.

If real memory is not available to capture the data that is needed for the memory dump, z/OS pages workload data to auxiliary storage to free memory for the SAN Volume Controller memory dump. However, this process delays the memory dump capture phase, which can result in delays in the transaction processing. More delays can be seen after the capture phase when the workload data (that was paged out) must be brought back in for processing.

Table 1-1 lists the SAN Volume Controller memory dumps that were performed in a test environment for the purposes of comparing memory dump capture times in environments where there was a real storage constraint versus environments in which there was no real storage constraint. In all of these scenarios, there was 40 GB of real storage on the system and the address space being memory dumped had used 20 GB of real storage by using the allocation of and references to 64-bit virtual storage.

In the constrained cases, more workloads and applications were started so that a large amount of real storage can be used before taking the SAN Volume Controller memory dump. These tests were run on z/OS V2R2 on a z13 processor.

Table 1-1 SAN Volume Controller memory dump summary

Scenario	SAN Volume Controller memory dump capture time in hh:mm:ss	Available frames before SAN Volume Controller memory dump start
No real storage constrained		
40 GB real storage; 20 GB SAN Volume Controller memory dump	00:00:18.150629	4,082,779 (15.57 GB)
40 GB real storage; 20 GB SAN Volume Controller memory dump	00:00:17.564853	4,083,614 (15.58 GB)
Real storage constrained		
40 GB real storage; 20 GB SAN Volume Controller memory dump	00:00:27.837104 (58% increase over fastest non-constrained time)	133,349 (520 MB)
40 GB real storage; 20 GB SAN Volume Controller memory dump	00:00:30.375583 (73% increase over faster non-constrained time)	32,817 (128 MB)

Processing delays during a SAN Volume Controller memory dump can be avoided by defining enough spare memory for taking the maximum size memory dump for your system as specified with the CHNGDUMP MAXSPACE parameter in COMMNDxx. This storage is more available real storage that is more than the peak workload available frame counts. The AFC counts can be tracked by using the RMF Monitor 2 SRCS report.



DB2 buffer pools and related tuning

This chapter describes how configuring more memory for product buffers can result in appreciable performance improvements by reducing I/O operations.

This chapter includes the following topics:

- ▶ 2.1, “DB2 buffer pools” on page 6
- ▶ 2.2, “DB2 for z/OS monitoring and tuning to using memory” on page 10
- ▶ 2.3, “IBM InfoSphere MDM with large DB2 group buffer pool” on page 16

2.1 DB2 buffer pools

Tuning DB2 buffer pools is one of the most significant types of DB2 performance tuning. One knob for DB2 buffer pool tuning is adjusting the size of the buffer pool. You must analyze the server performance for varying sizes of DB2 buffer pools to determine the optimum DB2 buffer pool size for your server. When running DB2 data sharing, consider the sizes of the DB2 group buffer pools and the DB2 local buffer pools.

Configuring more memory for DB2 buffer pools can improve response time and CPU time if it reduces I/O or group buffer pool access. Namely, the performance benefits depend on the workload, primarily the size of the active data and the reference pattern, and the configuration, data sharing versus single system, CPU utilization, and so on.

In this section, we show the results of some experiments where more memory was configured for DB2 buffer pools. An internal online banking transaction with a simulation workload was run on a 12 CP zEC12 that was running DB2 v11 for z/OS and z/OS 2.1.

Figure 2-1 shows the environment that was used to test the tuning of the buffer pools¹.

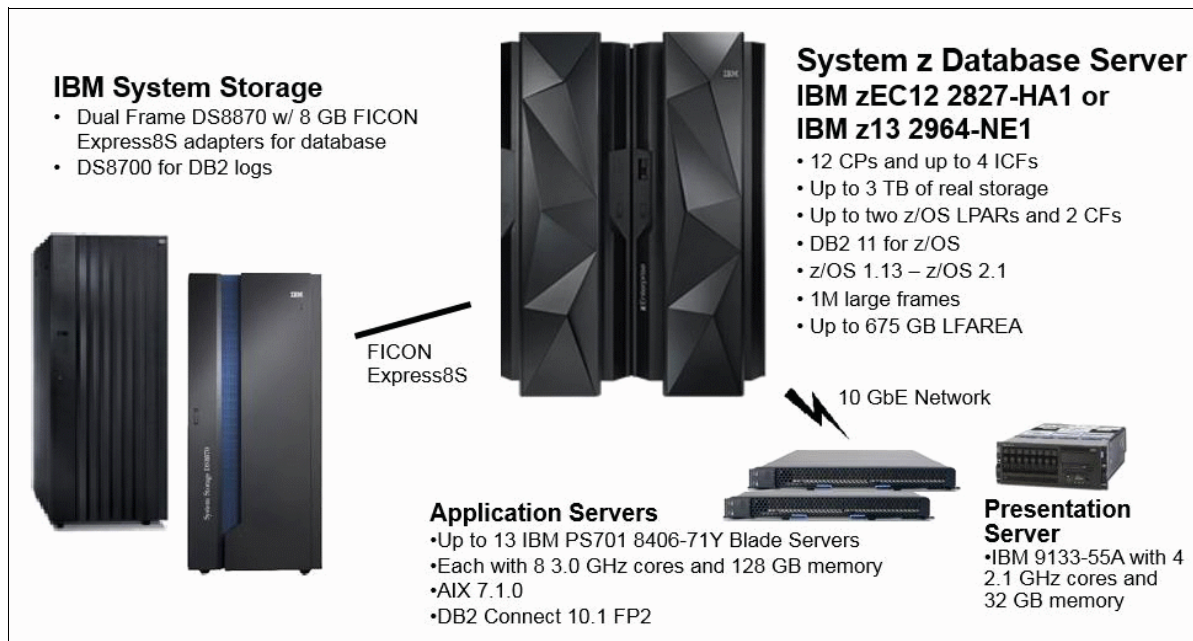


Figure 2-1 Test environment for large memory DB2 buffer pool study

Figure 2-2 on page 7 and Figure 2-3 on page 7 show the results of these measurements where different memory and buffer pool sizes were used. The amount of memory that was configured for local buffer pools for optimal performance was found (in this particular workload environment with a random data access pattern) to be 320 - 640 GB. Random hits in buffers can yield more CPU reduction as buffer pool sizes grow.

¹ For more information about our environment that was used for testing, see this website:
<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102461>

Memory	BP Size	CPU %	ITR	ITR Delta	ETR	ETR Delta	Txn response time(sec)	DB Request time delta	Sync Read IO/sec	Sync IO delta
256 GB	160 GB	72	992	n/a	709	n/a	.695	n/a	38.4k	n/a
512 GB	320 GB	73	1124	13.3%	819	15.5%	.428	-38%	11.7k	-69%
1024 GB	638 GB	79	1237	24.7%	976	37.7%	.209	-70%	0.9k	-97%

Figure 2-2 Single system bank online account posting workload

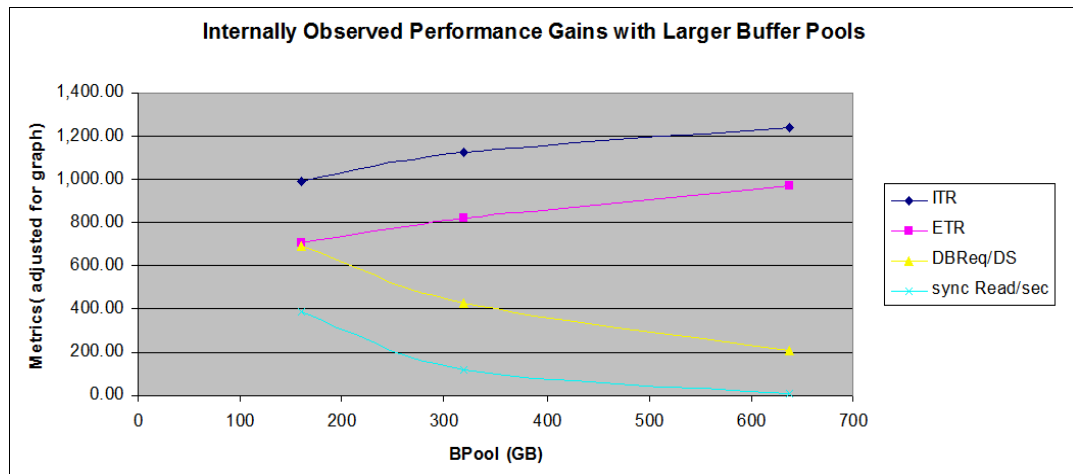


Figure 2-3 Single system bank online account posting workload

Large CPU time reductions are due mainly to avoiding hardware cache disruptions, which are directly tied to the task switching that is associated with I/O operations. Eliminating synchronous database I/Os through the configuration of larger buffer pools can have a direct influence on CPU time that is used.

Although the performance benefits start to flatten out after about 320 GB in this environment, more substantial gains can be on the lower end of the curves. Projecting to lower buffer pool sizes, for example, moving 10 - 24 GB, indicates continued performance benefits; response time might improve approximately 18%. This result is an indication that even small amounts of memory that are configured for DB2 buffer pools can have huge benefits, depending on the environment.

The following factors should be considered when memory is added to buffer pools:

- Consider your current buffer pool tuning. If your buffer pools are well-tuned with minimal synchronous I/O, good buffer pool hit ratios, and little group buffer pool directory reclaim activity, adding memory to your buffer pools does not provide any significant performance benefits.
- Consider your DB2 configuration. If you are running data sharing, consider the number of DB2 members in your data sharing group. This amount affects the total amount of memory that is needed when increasing the size of local buffer pools. The total amount of memory that is needed for the group buffer pools is twice the defined size for redundancy. In a data sharing environment, storage must be available in a second CF for the GBP cache structures to be rebuilt in a recovery scenario. When adding memory to local buffer pools in a data sharing environment, be sure to tune group buffer pools to avoid directory entry reclaims.

- ▶ Consider your workload characteristics. The data access patterns of a particular workload influence the results of adding memory to buffer pools. Eliminating random I/O, which results from random data access, is more likely to reduce DB2 synchronous reads than sequential I/O. In data sharing, the update intensity of the workload, the amount of GBP-dependent data, and the use of the DB2 member cluster feature can influence where more memory should be added to the local or the group buffer pools.
- ▶ Consider the use of the DB2 buffer pool simulation that was introduced in DB2 11 APAR PI22091 to identify buffer pools that might benefit from more memory. Good candidates for buffer pool simulation are buffer pools with a significant number of synchronous reads and that contain pages that are likely to be referenced again.

2.1.1 Buffer pool simulation

Having larger buffer pools might improve the elapsed time and CPU time of applications with large sync I/O waits as described in 2.1, “DB2 buffer pools” on page 6. However, estimating a realistic improvement is difficult in complex production environments where the mixtures of various DB2 applications are executed.

The actual benefit from larger buffer pools depends on the size of active data and the access pattern. For example, if you have a workload with mostly random access to data, increasing the buffer pool can be beneficial because it is likely that the same pages are referenced multiple times. However, if your application processes data sequentially (for example, only once per month), keeping the data in the buffer pools does not help.

The buffer pool simulation feature in DB2 11 and 12 helps provide a realistic estimation of the effect of having larger buffer pools without allocating the necessary storage. The feature is developed in DB2 12 and retrofitted to DB2 11 by using APAR PI22091.

Buffer pool simulation helps you determine the right size for your buffer pools. Simulated buffer pools require significantly less real storage and CPU overhead than do real buffer pools because only information *about* pages (rather than the actual pages) are stored in the simulated buffer pools.

Because DB2 applies the actual LRU algorithm to the simulated pools, the results of simulation are accurate if the workload is consistent. The results from the simulation are collected in buffer pool statistics (statistics trace class 1) and can be displayed by using a display buffer pool command. The information includes avoidable sync I/Os, async I/Os, avoidable GBP accesses, and elapsed time savings.

In addition to SPSIZE, DB2 introduces SPSEQT attribute for simulated pools to define the sequential threshold for simulation pools. This attribute can be used to simulate the larger pool with different sequential thresholds compared to the original buffer pools.

The simulation requires more memory for DB2 to store simulation information. It is approximately 2% of simulated pool size for 4 K page size buffer pools or 1% or less for 8 - 32 K page size. Our measurements indicate that there is approximately a 1% CPU overhead running simulation with many synchronous I/Os.

The buffer pool simulation can simulate only the case of adding pools and does not support reducing the size of current buffer pools.

Using simulation

This section uses an example to describe how to use the simulation. In this example, we want to simulate the possible benefit of adding 500,000 buffers (2 GB) to buffer pool BP1 with lower sequential threshold, as shown in Figure 2-4 on page 9.

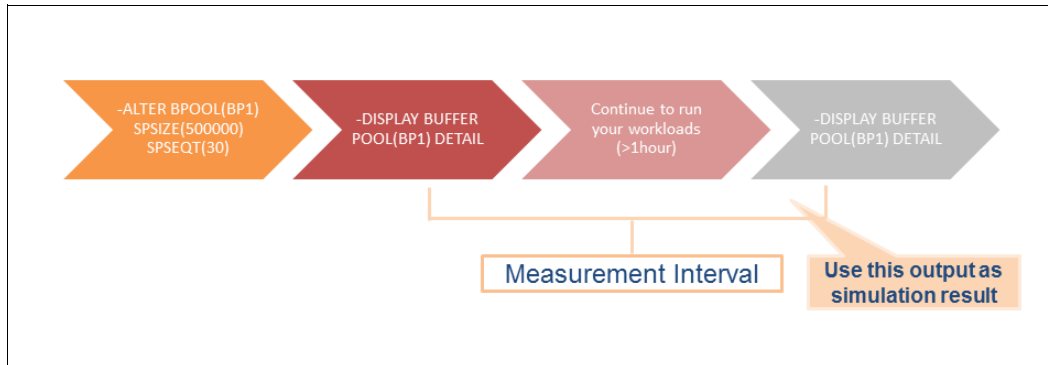


Figure 2-4 How to use simulation

Complete the following steps to run the simulation:

1. To start the simulation for an extra 500,000 buffers with simulation sequential threshold of 30, issue the following DB2 buffer pool command:
`-ALTER BPOOL (BP1) SPSIZE(500,000) SPSEQT(30)`
 DB2 allocates the simulation blocks.
2. Issue the following buffer pool display command to ensure that the simulation pool is set correctly:
`-DISPLAY BPOOL(BP1) DETAIL`
3. Continue to run your production workload, ideally 1 - 3 hours.
4. Issue the following buffer pool display command to collect the simulation results:
`-DISPLAY BPOOL(BP1) DETAIL`
5. Issue the following **ALTER** command to stop simulation:
`-ALTER BPOOL(BP1) SPSIZE(0)`

The output from the **DISPLAY BUFFER POOL** command in Step 5 contains the results of simulations. The same output also is published in DB2 statistics buffer pool statistics. Example 2-1 shows the sample output from the simulated pool activities. This output shows the simulation results from the interval between two **DISPLAY BUFFER POOL** commands (see steps 3 and 5).

Example 2-1 Sample output from simulated pool

```

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
  
```

As shown in Example 2-1 on page 9, there are 25463982 avoidable synchronous I/O requested as random read during this interval. SYNC_READ I/O(S) indicates the avoidable sync I/Os that are requested as sequential reads but ends up as synchronous I/Os. Asynchronous READ I/O indicates the avoidable prefetch read I/Os. GBP READS counters indicate the avoidable access to group buffer pools because the pages can be found in the simulated pools.

Avoidable sync I/O delay indicates the possible elapsed time saving by avoiding synchronous I/Os from this buffer pool.

2.2 DB2 for z/OS monitoring and tuning to using memory

In this section, we describe DB2 tuning opportunities that are outside of local and group buffer pools to use more memory and possibly reduce the CPU usage in DB2 applications.

2.2.1 DB2 and the effect of paging

Paging activities add CPU cost and affect response time. A cascading effect from paging, such as a thread holding locks or latches longer, also occurs. In a data sharing environment, heavy paging activities from one member can cause contention to other members that are running on healthy LPARs.

DFSORT or DB2sort (that DB2 invokes from the DB2 utility) might drive the high consumption of real storage and trigger paging during the batch period.

A DB2 dump is another common cause for paging and might affect availability. The recommendation is to reserve enough real storage for the possible DB2 dump process as shown in the following formula:

MAXSPACE for DB2 dumps = (DBM1 + MSTR+ IRLM + DIST + COMMON/ ECSA) - (total DB2 Buffer pool allocation)

Monitoring paging in DB2 subsystems

While the IBM z/OS Resource Monitoring Facility (RMF) reports LPAR level paging, DB2 subsystem reports paging activities in DB2 statistics traces.

Two sections report paging activities in DB2. One section is the overall subsystem storage usage, including threads and pool storages in DBM1 address space, as in “DBM1 real and auxiliary storage usage report”. As shown in Figure 2-5, AUXILIARY storage usage counters should be all zero to avoid an affect on performance. The second section is specific to buffer pools in buffer pool statistics. Although it is normal to have nonzero values immediately after DB2 restart, page-ins that are required for read and write counters should be zero or close to zero at steady state.

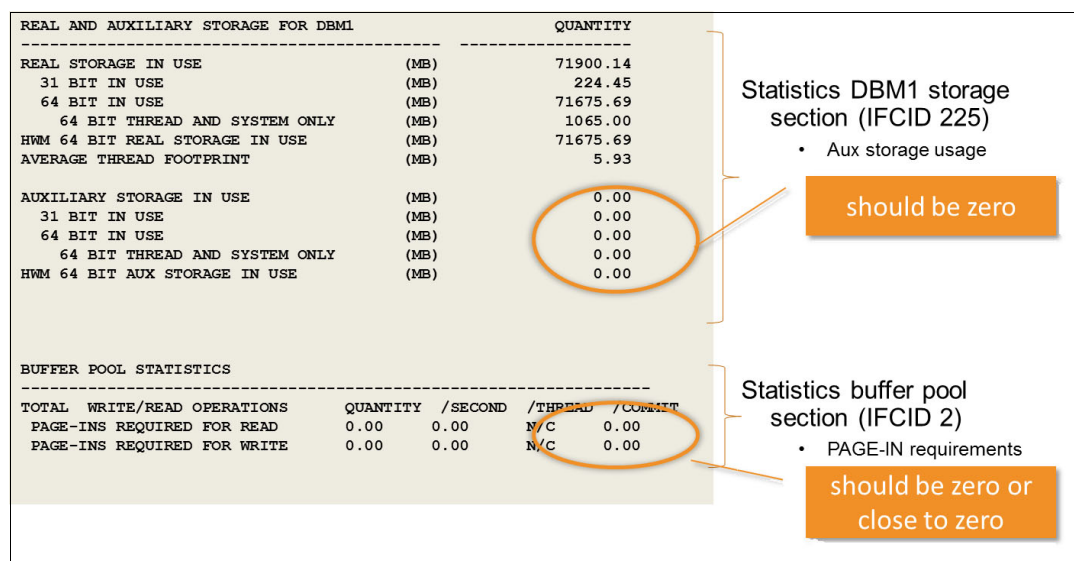


Figure 2-5 Paging report from the DB2 statistics trace

If paging is observed in DB2, it is strongly recommended to review your memory allocation strategy on the LPAR to avoid paging.

2.2.2 Reusing DB2 threads

Noticeable CPU savings can be achieved by optimizing DB2 thread management. Although reusing the thread can reduce the overhead of allocation and termination of the thread, it requires more memory because the thread-related information is kept across transactions or commit boundaries. Thread reuse is effective when processing short running transactions. Figure 2-6 shows the CPU savings of reusing the thread by using IBM Relational Warehouse Workload. We compare the case with reusing threads all of the time (100% thread reuse) against the case when threads are reused 66 - 67% of the time.

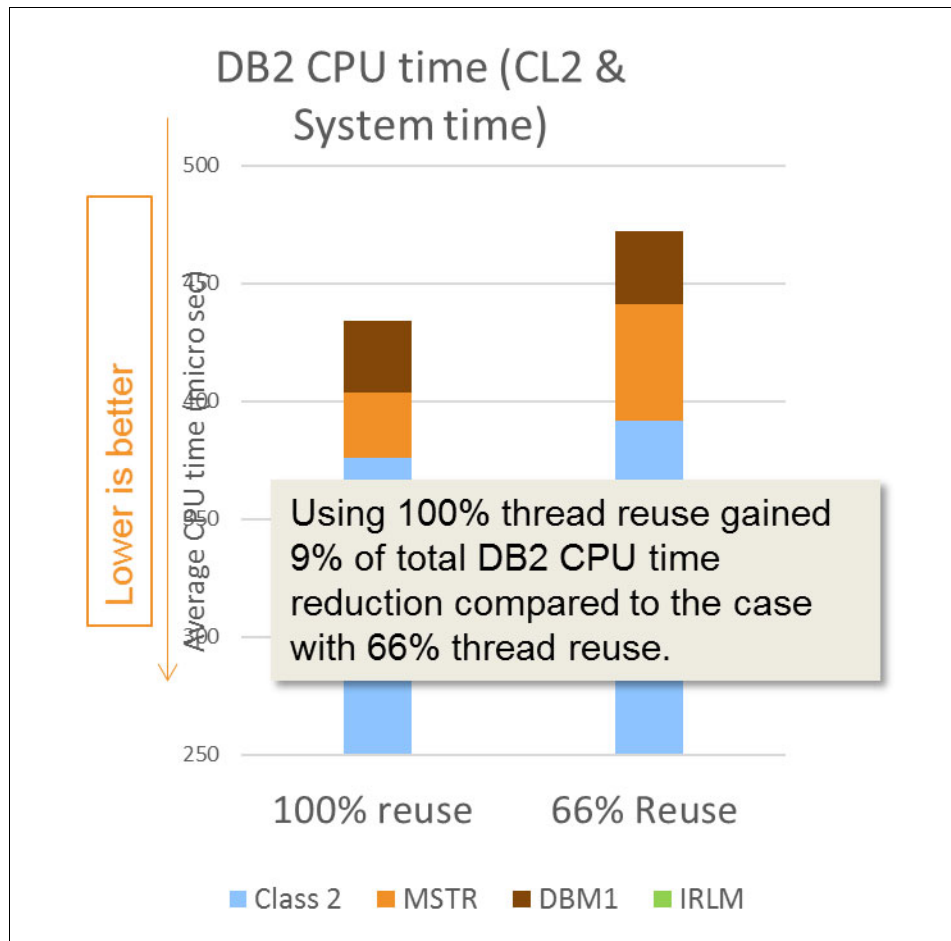


Figure 2-6 Reusing thread: IBM Relational Warehouse Workload

For local attached transactions, thread reuse can be achieved through transaction managers, such as CICS or IBM IMS™. In CICS, thread reuse can be achieved with the use of protected threads and without the use of protected threads by queuing requests to the pool and entry thread.

For distributed transactions, threads normally are pooled and reused when CMTSTATS INACTIVE (default and recommended) is used.

Finding thread reuse rate in your DB2 workloads

The DB2 accounting report provides the information about thread reuse in the Accounting termination section, as shown in the example in Figure 2-7 on page 13.

NORMAL TERM.	AVERAGE	TOTAL
-----	-----	-----
NEW USER	0.67	539864
DEALLOCATION	0.33	270518
APPL.PROGR. END	0.00	0
RESIGNON	0.00	0
DBAT INACTIVE	0.00	0
TYPE2 INACTIVE	0.00	0
RRS COMMIT	0.00	0

Figure 2-7 DB2 accounting: termination section

The termination section indicates the reasons for normal transaction terminations. The counters, such as NEW USER, RESIGNON, TYPE2 INACTIVE, and DBAT INACTIVE indicate that the threads are reused while DEALLOCATION means that the thread is terminated without being reused. In the example that is shown in Figure 2-7, the threads are reused (NEW USER) for 67% of time while they are terminated and reallocated for 33% of the time. By increasing the rate of thread reuse, CPU savings from the transactions can be observed.

Identifying the transactions benefit from thread reuse

Not all the transactions must reuse the thread to see benefits. Our recommendation is to start with the frequently run transactions because they benefit from thread reuse the most. To identify the frequently run transactions, DB2 accounting trace can be examined, as listed in Table 2-1. In the table, plans are sorted in a descending order of commits per second. The plans in the first two rows of Table 2-1 are good candidates for reusing threads.

Table 2-1 DB2 plan name and commit frequency, CPU usage

PLANANME	TOT CL2CPU	COMMT s/sec
XKB1BR1	433.93	37.16
XKS1BR1	276.06	10.49
HCIH007	25.89	5.82
HCIH002	50.53	5.82
XKB1ER1	89.63	5.16
XKB10R1	80.53	5.03
XKB1DR1	45.63	4.74
XKB02R1	63.61	3.81
XB06R1	37.52	3.78
XKP1BR1	39.76	3.71
XKB1LR1	54.85	3.47

2.2.3 RELEASE(DEALLOCATE) option for persistent threads

After the threads are reused, the RELEASE(DEALLOCATE) option in DB2 BIND/REBIND becomes applicable to achieve further CPU reduction by using more memory. When the packages are bound with RELEASE(DEALLOCATE) instead of RELEASE(COMMIT), DB2 keeps part of resources that are allocated in the thread across the commit boundary. This configuration eliminates some processing of repeated executions of the same packages. The RELEASE(DEALLOCATE) option is relevant only when thread reuse is used. If the threads are not reused, there is no difference in DB2 behavior between RELEASE(DEALLOCATE) and RELEASE(COMMIT) as a commit boundary becomes the same as a thread boundary.

The resources that DB2 keeps across the commit in RELEASE(DEALLOCATE) vary, depending on whether the statements are dynamically prepared or statically bound. For static statements or dynamic statements that use KEEP_DYNAMIC(YES), DB2 keeps packages and statement information and parent locks. Table 2-2 lists the parent and child lock information.

Table 2-2 DB2 lock hierarchy

Percent locks	Child locks
Simple table space	Data pages and rows
Partitioned table space	Data pages and rows
Segmented table space	Data pages and rows
Tables in a segmented table space	N/A
LOB table space	LOB

For a dynamic statement without the use of the KEEP_DYNAMIC(YES) parameter, DB2 keeps package information.

Frequently used packages that are associated with high-volume transactions that achieve thread reuse are ideal candidates for the RELEASE (DEALLOCATE) bind option parameter. Because more information is kept in the thread, the RELEASE(DEALLOCATE) parameter uses more thread storage.

It is recommended to apply the RELEASE(DEALLOCATE) parameter for selective packages to avoid unnecessarily using real storage. The benefit is seen in DB2 accounting class 2 and 7 CPU time. In IBM measurements, up to 10% CPU reduction is seen using the RELEASE (DEALLOCATE) parameter instead of the RELEASE(COMMIT) parameter.

Identifying the packages benefit RELEASE(DEALLOCATE) BIND option

To identify the candidate packages to use RELEASE(DEALLOCATE), package accounting information from DB2 accounting traces can be examined. Package accounting can be collected by turning on accounting class 7 and 8.

Figure 2-8 shows the sample packages, execution frequency, and CPU usage sorted in descending order by execution frequency. The top packages that are circled in Figure 2-8 are good candidates to consider the use of `RELEASE(DEALLOCATE)`.

PACKAGE	ALLOC/sec	TOT CL7CPU%	CL7CPU Avg	CL7CPU
KSBUARG	108.77	273.93	6.52%	0.000504
KSBU16D	81.75	413.22	9.83%	0.001517
KSBU16C	30.61	101.48	2.42%	0.001257
HCIU3DA	23.24	24.87	0.59%	0.000275
KGVDVDT	22.12	51.44	1.22%	0.000892
CRFUFDB	21.28	45.44	1.08%	0.013850
CRPUPDB	18.80	23.05	0.55%	0.000441
XWPUI04	18.14	31.52	0.75%	0.000260
XSDULC1	14.64	22.64	0.54%	0.000456
XAUUHL1	12.41	29.62	0.70%	0.000519
KNVU200	12.28	60.85	1.45%	0.000865
HCIU3U1	11.45	3.69	0.09%	0.000131
XKSUS61	11.43	74.82	1.78%	0.000434
KGVDUI	11.40	30.33	0.72%	0.001152
XSDULB1	11.29	11.32	0.27%	0.000338
XWPUI08	11.06	17.77	0.42%	0.000279
XAUUO06	10.35	4.05	0.10%	0.000194
XAUUE06	10.35	6.85	0.16%	0.000249
XAUUE08	10.35	6.84	0.16%	0.000221
XAUUN06	10.34	11.13	0.26%	0.000208
HCIU33G	9.74	3.32	0.08%	0.000116
XMVUM41	9.69	29.61	0.70%	0.000459
XWPUKF4	9.55	62.26	1.48%	0.000364
KUMUP01	9.20	96.71	2.30%	0.002134
KRAUER1	8.92	1.06	0.03%	0.000198
HCIU3U2	8.46	8.36	0.20%	0.000306
CSMURSP	8.15	3.23	0.08%	0.000546
KKDUMCA	7.74	284.34	6.77%	0.000787

Figure 2-8 Sample packages

RELEASE(DEALLOCATE) considerations

The biggest challenge to implement `RELEASE(DEALLOCATE)` in a persistent thread is the difficulty of breaking the thread during database maintenance, such as online REORG, schema changes, or REBIND. This issue occurs because the persistent thread holds the shared lock for packages, DBDs, and parent locks until the thread is deallocated. Another challenge is a possible CPU and memory increase by creating long chains of parent locks and associated control blocks from large numbers of DB2 objects.

DB2 11 introduced features to relieve these challenges. A break-in feature in DB2 11 checks resource contention that is held by persistent threads and releases the locks, if needed. DB2 11 also checks the number of parent locks and control blocks that are held by a thread. If the number exceeds the threshold, DB2 releases the resource to keep the chain to a reasonable length.

2.3 IBM InfoSphere MDM with large DB2 group buffer pool

IBM InfoSphere® Master Data Management (MDM) is an IBM WebSphere® J2EE application that manages master data for single or multiple domains within a large enterprise, such as bank customers and insurance policies. Because it is a mission critical system, it demands high availability, scalability, and performance.

MDM's highly customizable service-oriented architecture, normalized data model, and random database object keys often result in transactions that use many SQL statements and a large quantity of synchronous I/Os from DASD. This usage usually leads to less than wanted transactions per second (TPS) throughput.

Because storage is becoming less expensive, loading data into local or group buffer pools for better I/O performance is increasingly popular and practical, particularly because most of the MDM workload is read only.

In this section, we describe the performance effects of the use of large group buffer pools (GBPs) to store backend MDM database objects, which reduces the need for synchronous disk I/Os.

2.3.1 Environment

We set up a two-way DB2 data sharing environment as shown in Figure 2-9. Each DB2 member is in its own logical partition (LPAR) with equal amounts of CPU and storage defined. Both members are linked by using an internal coupling facility that has 712 GB of storage. We purposely increased the individual group buffer pool to be large enough to store all of the MDM tables and indexes so that essential GBPs can be primed for data storage to eliminate all trips to DASD during an application run.

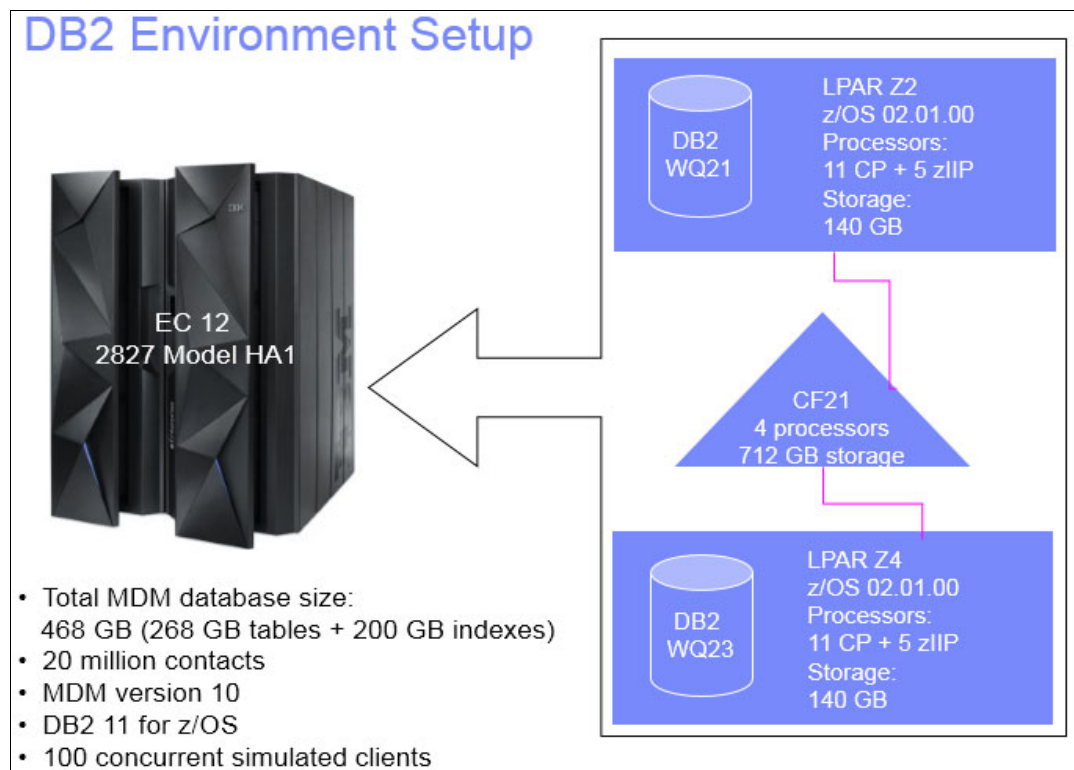


Figure 2-9 DB2 for z/OS environment configuration

For the base test scenario (see Figure 2-10), local DB2 buffer pools and corresponding group buffer pools were configured in such a way that default GBP castout and thresholds are used. GBPCACHE options for MDM tables and indexes were set to CHANGED.

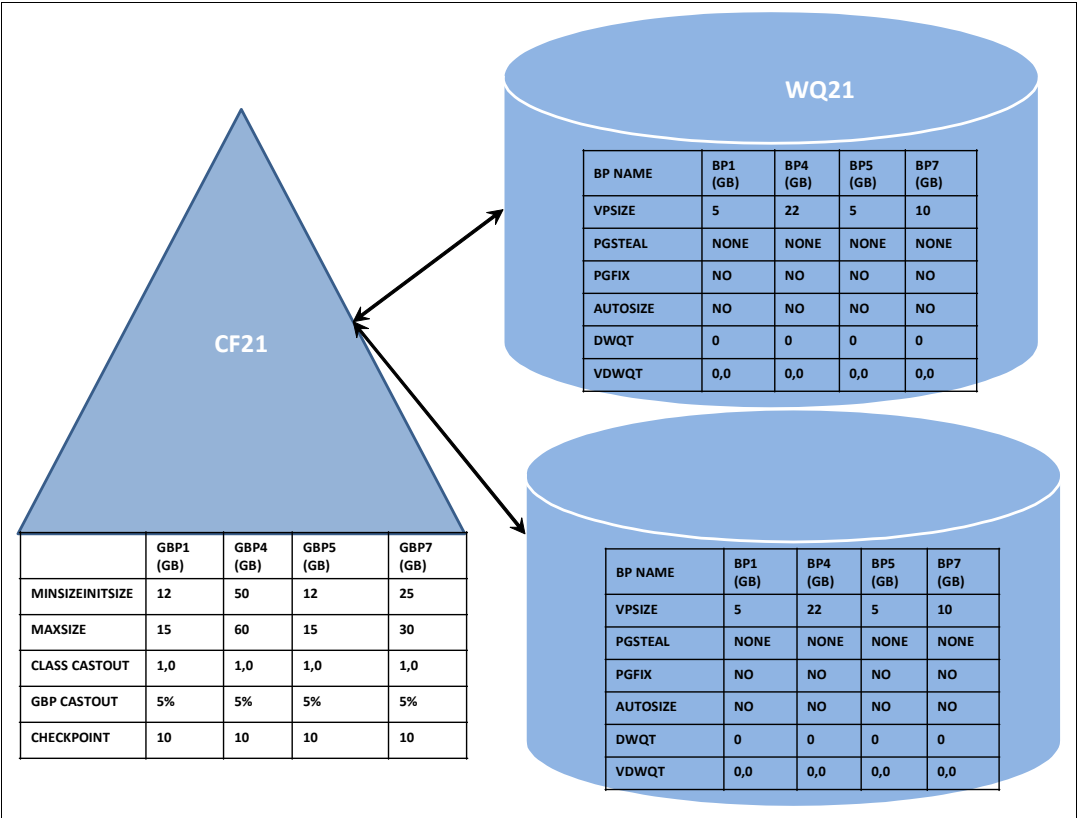


Figure 2-10 Baseline test configuration

For the large GBPCACHE ALL scenario, the GBP castout thresholds were intentionally set high so that pages are retained in the GBP. Also, to prime the GBP, local buffer pool PGSTEAL(NONE) was set to trigger prefetch to speed out page population. Then, PGSTEAL was switched to least recently used (LRU).

Based on the DISPLAY XCF structure command output that shows the actual GBP usage, the directory-to-data ratio was adjusted and the INITSIZE, MINSIZE, and MAXSIZE of the structures also were adjusted. Because we had two DB2 members, the GBP needed two directory entries for every updated page, and each directory entry took about 250 bytes of space. That is, there was 0.5 K (or 512 bytes) of space for directory entries for every 4 K data page. This issue was factored in when sizing the GBPs to allow for data growth as a result of the application adding records. If index compression is used, that compression also must be factored in because index pages are decompressed in the buffer pool.

Local buffer pools also were defined based on the data characteristics of each buffer pool. Frequently changed local buffer pools were defined to be half of the corresponding GBPs. This factor increases the chance of finding a needed page in the local buffer pool (average response time was 1 microsecond), which still outperformed getting a page from the GBP (3.6 microseconds according to the average service time field in the Coupling Facility Structure Activity Report). This rate, in turn, is faster than getting a page from DASD cache (600 microseconds) and DASD (7 milliseconds).

We configured the GBP in simplex mode for ease of use and because of a lack of hardware resources in our lab environment. However, we strongly encourage running the GBP in duplex mode in a production environment because incurring a few percentages of overhead in maintaining redundant structures is better than having a single point of failure and hours or even days of recovery if there is a large coupling facility (CF) outage.

Table 2-3 lists the test results with our MDM workload that uses a large GBP.

Table 2-3 Performance results

Performance measure	MDM mixed workload
Average sync I/O response time (μs)	758
Average response time for reading from GBP (μs)	3.6
Before sync I/O per transaction	35.69
After sync I/O per transaction	1.57
Weight of sync I/O suspension time in total CL1. Elapsed time	11.7%
End-to-end transaction response time reduction	5%
Total CPU time reduction	6.8%

The performance results in this table demonstrate that the use of a large GBP to reduce synchronous I/O is a good tradeoff between CF storage and CPU/TPS for I/O intensive transactions. Most MDM applications benefit from this configuration, especially when running with Probabilistic Matching Engine (PME) enabled.

MDM workloads might see more or less CPU and elapsed time savings than our actual test depending on the number of synchronous I/Os in an MDM transaction, processor model, speed of your DASD devices, and so on.



Java and large memory

Starting with the IBM System z10®, z Systems platforms can support large pages of 1 MB and smaller pages of 4 KB. This capability is a performance item that addresses particular workloads and relates to large main storage usage. Both page frame sizes can be simultaneously used.

In this chapter, we describe large page support with the objective of measuring large memory to demonstrate performance improvements that uses large pages in a z/OS environment that has many Java virtual machines (JVMs). Each JVM uses a 2 GB heap size, which is a known optimal heap size for Java.

This chapter includes the following topics:

- ▶ 3.1, “Background” on page 20
- ▶ 3.2, “IBM z13 Java large memory with large page measurements” on page 20

3.1 Background

Java is a critically important language for z Systems. For data serving and transaction serving, which are traditional strengths of the z Systems platform, Java is now foundational. For example, WebSphere applications that are written in Java and running on z Systems, provide a key advantage through collocation. This advantage results in better response times, greater throughput, and reduced system complexity when driving CICS, IBM IMS, and DB2 transactions. Beyond this fact, Java became a language of choice for CICS, IMS, and DB2 transactions as clients seek to extend and modernize their business logic.

Java also is critical for enabling next generation workloads in cloud, analytics, mobile, and security (CAMS). Cloud and mobile applications can access data and transactions on z/OS and other WebSphere solutions, which all are inherently Java based.

IBM Operation Decision Manager (ODM) is written in Java. ODM is a platform for managing and running business rules and business events to help make decisions faster, improve responsiveness, minimize risks, and seize opportunities. The IBM MobileFirst™ Platform Developer Edition (formerly known as IBM Worklight Developer Edition) provides developers with the tools to quickly deploy mobile solutions that use Java. IBM z Systems Java also provides a full set of cryptographic functions to implement secure solutions.

Because of the importance of Java, it is understandable why a system might use many JVMs.

3.2 IBM z13 Java large memory with large page measurements

The objective of the large memory performance measurement was to demonstrate performance improvements by using large pages in a z/OS environment that has many JVMs. Each JVM uses a 2 GB heap size, which is a known optimal heap size for Java.

3.2.1 Initial set of Java options

The following Java options were used in all the measurement runs and are known to be an excellent starting point for optimal performance when 64-bit Java applications are run:

- ▶ `Xmx2040m` and `Xms2040m`

This option sets the maximum and initial heap size to 2 GB.

- ▶ `Xcompressedrefs`

This option reduces the size of heap by compressing 8-byte addresses to 4 bytes.

Because Java uses many addresses, it reduces the memory usage and cache miss rates.

- ▶ `Xgcpolicy:gencon`

This option is a generational garbage collector for the IBM JVM. The generational scheme attempts to achieve high throughput with reduced garbage collection pause times.

- ▶ `Xmn1300`

This option sets the initial and maximum size of the new area to the specified value when `Xgcpolicy:gencon` is used, which varies for each application object usage.

3.2.2 Measurements

The following measurements are shown in Figure 3-1:

- ▶ 1 JVM that uses a 2 GB heap
- ▶ 2 JVMs that use 4 GB for Java heaps
- ▶ 4 JVMs that use 8 GB for Java heaps
- ▶ 10 JVMs that use 20 GB for Java heaps

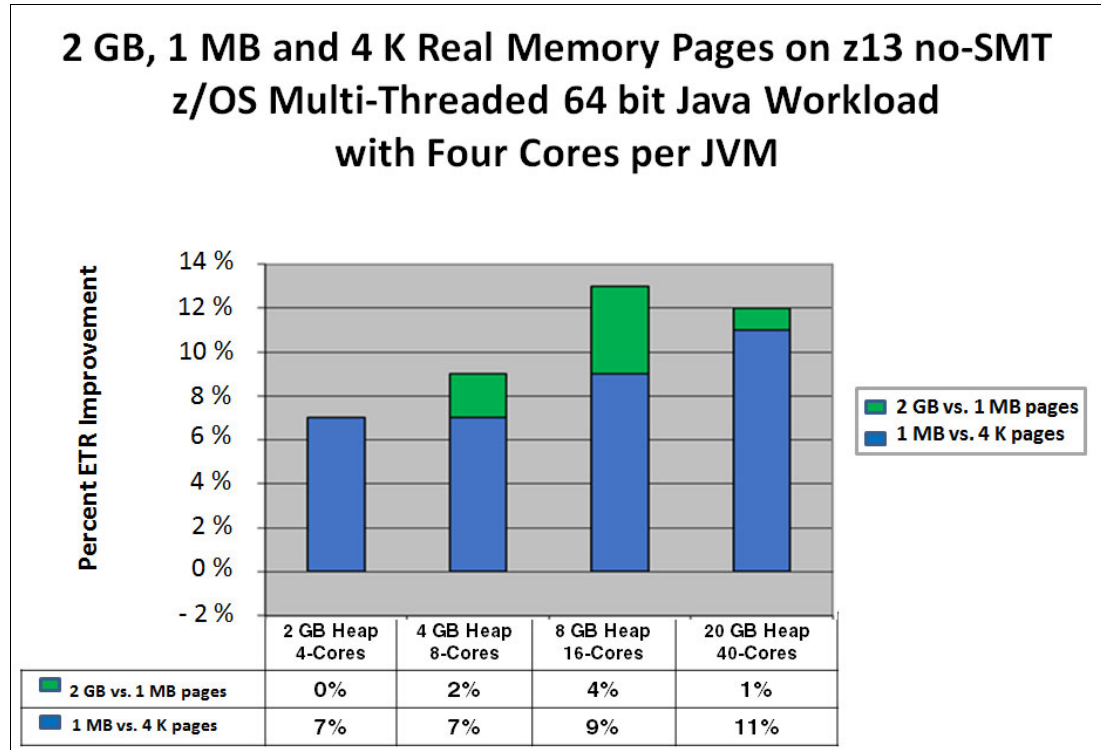


Figure 3-1 2 G, 1 M, and 4 K real memory pages on an IBM z13

Each measurement was run three times with the following requests for the allocation of large pages (the Java option -Xlp):

- ▶ 4 K pages size: Xlp:objectheap:pagesize=4K
- ▶ 1 M large pages: Xlp:objectheap:pagesize=1M
- ▶ 2 GB fixed large pages: Xlp:objectheap:pagesize=2G,nonpageable-

You define the number of 1 MB page frames that are available to z/OS in the LFAREA parameter of SYS1.PARMLIB(IEASYSxx).

Figure 3-1 shows that throughput improved from 7% up to 13% by using large pages versus 4 K pages on the IBM z13. Both 1 MB and 2 GB large pages improved performance. For larger memory usage, 2 GB fixed large pages saw up to 4% better throughput than 1 MB large pages.



IBM MQ and CICS

This chapter describes CICS paging reduction and the use of larger memory with CICS. Also described is the use of DFSORT and its role in reducing work data set I/O and improving sort elapsed time.

This chapter includes the following topics:

- ▶ 4.1, “Using larger memory with CICS” on page 24
- ▶ 4.2, “IBM MQ v8 64-bit buffer pools” on page 31

4.1 Using larger memory with CICS

The CICS Transaction Server (CICS TS) for z/OS provides several facilities that use in-memory data. If insufficient real memory is available for these facilities, paging can result. If you are using such facilities and are experiencing paging (which can be seen from IBM RMF reports), you might benefit from configuring more real memory on the system and reducing the overhead that is associated with paging.

However, it is more likely that you limited your use of in-memory facilities to avoid or minimize paging because this issue has such a detrimental effect on performance. With greater amounts of real memory available at an attractive price point on the IBM z13, you can revisit your use of these facilities and consider increasing your usage.

You can use the following CICS in-memory facilities for more real memory (the facilities are listed in the order that are likely to benefit the greatest number of users):

- ▶ CICS main temporary storage

As of CICS TS V4.2, CICS TS MAIN uses above-the-bar 64-bit storage in the CICS address space. Although this configuration increases the amount of virtual storage that can be used for CICS temporary storage, paging can result if there is insufficient real storage to back it. Configuring more real memory can allow greater use of CICS main temporary storage.

- ▶ CICS shared data tables

CICS-maintained and user-maintained CICS shared data tables are held in data spaces that are owned by a CICS region and can span up to 100 data spaces for each data table owning region. Having larger memory available can support much larger shared data tables or many more shared data tables of a more moderate size. The use of shared data tables can reduce I/O access to the underlying Virtual Storage Access Method (VSAM) data set and avoid function shipping overheads.

- ▶ CICS internal trace table

As of CICS TS V4.2, the CICS internal trace table uses 64-bit storage in the CICS address space. This configuration can allow for much larger trace tables to aid problem determination.

- ▶ Channels and containers

Applications can use CICS channels and containers to pass data between programs and the data is held internally by CICS in 64-bit storage. Configuring more real memory can allow larger amounts of data to be passed in this way. CICS ensures that any individual task cannot over-allocate storage from the 64-bit virtual storage MEMLIMIT for the region.

- ▶ DB2 or VSAM data access

CICS applications that access data in DB2 or VSAM record-level sharing (RLS) data sets can benefit from larger DB2 buffer pools or larger VSAM RLS buffer pools.

- ▶ Java heap storage

Although CICS Java applications do not typically use large amounts of Java heap storage, benefits can be realized from having more memory available for those applications that do use such amounts of storage.

- 64-bit CICS APIs

As of CICS TS V5.1, applications that are written in non IBM Language Environment® assembler can use new CICS 64-bit APIs (EXEC CICS GETMAIN64 and EXEC CICS PUT64 CONTAINER), which acquire 64-bit storage. Availability of more real memory can be of benefit to such applications.

4.1.1 Considerations for the use of large memory with CICS

CICS uses real memory to provide backing storage for virtual storage use within the CICS address space. Some of the most significant uses of virtual storage in CICS that can require significant amounts of real memory as backing storage were described in 4.1, “Using larger memory with CICS” on page 24. These significant uses of virtual storage might be described as “storage-hungry” facilities.

In recent releases, CICS moved more of its storage areas into above-the-bar (64-bit) storage. This change allows much more virtual memory to be used for these storage areas, and hence real storage rather than virtual storage can become the limiting factor.

CICS also provides access to data stores, such as DB2 and VSAM. These access methods use buffers and storage pools, which also require real memory to provide backing storage. Consider the following points:

- For VSAM record-level sharing (RLS) access, the storage buffers are in 64-bit storage, whereas 31-bit storage is used for non-RLS LSR files.
- DB2 makes extensive use of buffer pools, as described in the section on DB2.

In general, you should minimize paging for CICS regions. Making greater amounts of real storage available can allow increased use of the “storage-hungry” CICS facilities without incurring the overheads of paging.

4.1.2 Real memory challenges for CICS

Paging results if there is insufficient real storage available to provide backing storage for the CICS address space. This issue can have a dramatic effect on response times.

The potential effect of paging is shown in Figure 4-1.

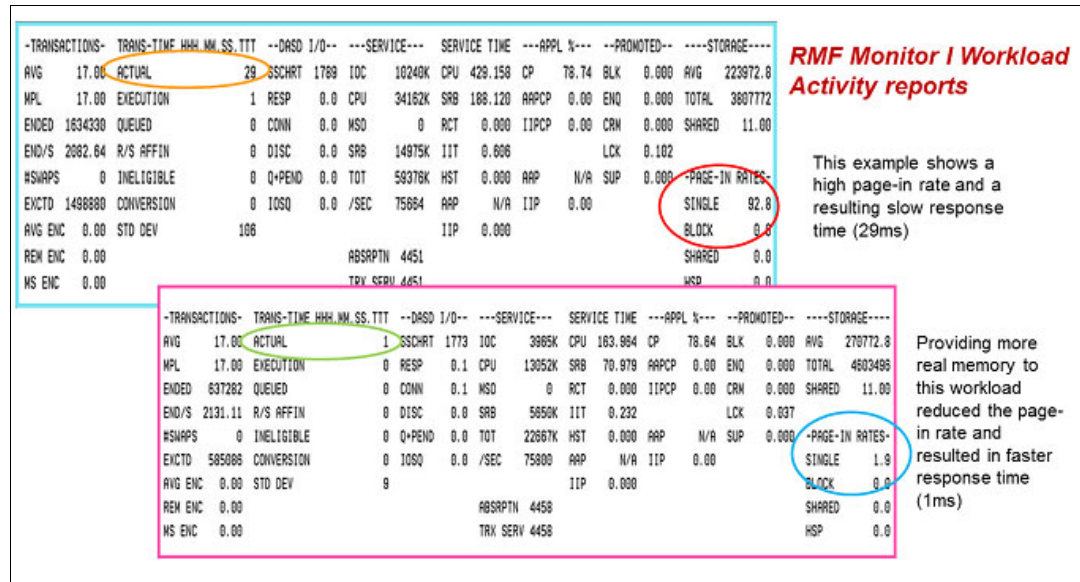


Figure 4-1 Effect of paging in CICS on response time

In Figure 4-1, the area in the blue square shows an RMF Monitor I Workload Activity report for a set of CICS regions that are experiencing a high page-in rate of 92.8 per second (circled in red). The average response time for transactions that are running in these regions is also high (29 milliseconds, which is circled in orange).

By contrast, the area that is in the pink square in Figure 4-1 is the equivalent RMF report when more real storage was made available. The page-in rate dropped to 1.9 per second (circled in blue), and the response time is now 1 ms (circled in green).

4.1.3 Measuring memory use by CICS

The best way to determine whether CICS has sufficient real memory is to check that the CICS address spaces are not paging or are experiencing only small amounts of paging.

There are no hard and fast rules for what is an acceptable amount of paging. It is best to strive for no paging at all, and a paging rate in excess of about 100 per second must be investigated.

You can check the following places for paging in CICS, which are described next:

- ▶ RMF Monitor I (shown in Figure 4-1)
- ▶ RMF Monitor III
- ▶ IBM OMEGAMON®
- ▶ CICS statistics (quasi-reentrant TCB usage can be an indicator)
- ▶ Paging in SDSF

CICS does not provide any indications of paging, but the CICS address space paging can be reviewed by using RMF reports, SDSF, or OMEGAMON (or other Monitoring products).

CICS statistics do not include any explicit information about paging, but a hint that paging might be an issue can be derived from the information that is provided about the use of the QR.

Paging in RMF Monitor I workload activity reports

As was shown in Figure 4-1 on page 26, the RMF Monitor I workload activity report can show page-in rates for CICS workloads. However, it is possible to miss examples of paging when aggregations of CICS regions are reviewed.

The reports that are shown in Figure 4-2 are RMF Monitor I workload activity reports for a collection of service classes that make up a workload.

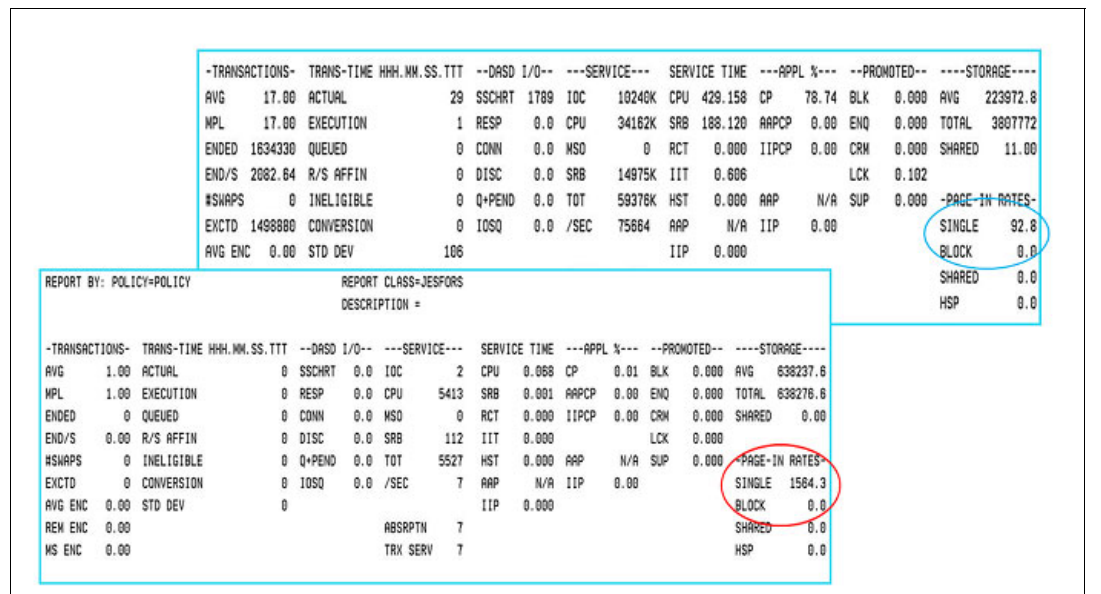


Figure 4-2 Page-In Rates across service classes

In the report that is at the top of Figure 4-2, the workload includes a combination of 17 CICS regions that are servicing an application. This workload involved four terminal-owning regions (TORs), 11 application-owning regions (AORs), a IBM CICSplex® SM management address space, and a file-owning region (FOR). The page-in rate for this workload is 92.8 (circled in blue) which, although not ideal, might not be an immediate cause for concern.

However, in the bottom report that is shown in Figure 4-2, the file-owning region is separated out. The page-in rate for this aspect of the workload (circled in red) is considerably higher, and is cause for some concern.

Therefore, if some amount of paging is seen across a collection of CICS regions, it is worth reviewing further to see whether one of those regions is suffering excessive paging.

RMF Monitor III

Figure 4-3 on page 28 shows an RMF Monitor III report that was obtained for a CICS environment where real storage was limited and paging was occurring. This report is created by using the ISPF interactive RMF Monitor III application. It is the STORJ report in the Job Delays section. As shown in Figure 4-3 on page 28, the report provides advice about the occurrence of paging as a concern.

```

RMF V2R1      Job Delays
Line 1 of 1
Command ==>
Scroll ==> CSR

Samples: 5      System: MV2A  Date: 10/08/15  Time: 13.24.30  Range: 10  Sec

Job: CICS001      Primary delay: Paging of cross-memory storage.

Probable cause:  Paging in other address space.
Help panels contain more possible causes.

----- Job Storage Usage Data -----
Average Frames:  100K      Working Set:  100K      Fixed Frames:  1319
Active Frames:   100K      Aux Slots:   285K      DIV Frames:    199
Idle Frames:      0      Page In Rate:  698      ES Move Rate:  N/A

----- Job Performance Summary -----
Service      WFL -Using%- DLY IDL UKN ---- % Delayed for ---- Primary
CX ASID Class  P Cr  %  PRC DEV  %  %  %  PRC DEV STR SUB OPR ENQ Reason
B0 0271 CICSBTCH 1      0    0  0  20    0  80    0    0  20    0    0    0 XMEM

```

Figure 4-3 RMF Monitor III Storage Display

Seeing CICS paging by using Tivoli OMEGAMON

IBM Tivoli® OMEGAMON XE for CICS on z/OS in the Region Overview workspace shows the Page Rate that is listed under the z/OS Address Space tab, as shown in Figure 4-4.

----- File Edit View Tools Navigate Help 03/01/2016 14:39:19 -----		Auto Update : Off	
Command ==>		CICSplex : OMEGAPLEX	
KCPRGNOZ		Region : CICSTIV1	
CICS Region		z/OS Address Space	
Data Sources			
CICSTIV1 z/OS Address Space			
Largest Contiguous LSOA...		1244K	Largest Contiguous OSCOR.....
Page Rate.....		0.0/s	I/O Rate.....
Working Set Size.....		3592K	Region Status.....
z/OS Address Space Details for CICSTIV1 0x0059			
Type.....			
JESJOBID.....			
Step Name.....			
Proc Step.....			
CPU Percent.....			
IFA Percent.....			
SRB Percent.....			
TCB Percent.....			
ZIIP Percent.....			
CPU Percent Excluding Home SRB Time.....			
IFA Percent With Enclave Home SRB Time.....			
ZIIP Percent With Enclave Home SRB Time.....			
IFA on CP Percent.....			
ZIIP on CP Percent.....			
Job CPU Percent.....			
Job SRB Percent.....			
Job TCB Percent.....			
Job CPU Time.....			
Job SRB Time.....			
Job TCB Time.....			
Start Up Monitored.....			
Job Preemptable Home SRB Service Time.....			
Job Preemptable Home SRB Service Percent.....			
Job Additional SRB Service Time.....			
Job Additional SRB Service Percent.....			

Figure 4-4 IBM Tivoli OMEGAMON XE for CICS for z/OS Region Overview workspace

For more information about paging in OMEGAMON for CICS Classic, see this website:

<https://ibm.biz/BdsbQc>

OMEGAMON on z/OS shows paging information for any address. For more information, see this website:

<https://ibm.biz/BdsbQR>

CICS statistics

CICS TCB statistics can indicate that excessive paging is occurring. There might be a paging problem if the QR TCB, which is the CICS quasi-reentrant TCB that processes any non-thread safe work in CICS, is using CPU (and therefore doing productive work) for only a small proportion of the time that it is dispatched. A low ratio means that the QR TCB is involuntarily giving up the use of the CPU for periods as a result of an interrupt rather than issuing a Wait. This issue can be caused by CPU contention, where higher priority address spaces are taking the CPU or paging.

The top of Figure 4-5 shows the Dispatcher TCB Modes report for a system where considerable paging was occurring, and as shown in Figure 4-5, the row for the QR TCB features a large Dispatch time but a small Time/TCB. The bottom of Figure 4-5 was taken on a system with less paging. In this case, the Accumulated time is more than half the dispatch time.

TCB Mode	< TCBs Attached >		TCB	Attach	MVS	Accumulated	Accumulated	Accumulated
	Current	Peak	Attaches	Failures	Waits	Time in MVS wait	Time Dispatched	Time / TCB
QR	1	1	0	0	143277	00:10:24.771944	00:02:40.437340	00:00:28.560502
RO	1	1	0	0	0	00:00:00.000000	00:00:00.000000	00:00:00.000000
CO	0	0	0	0	0	00:00:00.000000	00:00:00.000000	00:00:00.000000
SZ	0	0	0	0	0	00:00:00.000000	00:00:00.000000	00:00:00.000000

QR TCB's CPU to Dispatch ratio is low (17.8%)

TCB Mode	< TCBs Attached >		TCB	Attach	MVS	Accumulated	Accumulated	Accumulated
	Current	Peak	Attaches	Failures	Waits	Time in MVS wait	Time Dispatched	Time / TCB
QR	1	1	0	0	76624	00:04:39.298034	00:00:20.673721	00:00:10.769302
RO	1	1	0	0	0	00:00:00.000000	00:00:00.000000	00:00:00.000000
CO	0	0	0	0	0	00:00:00.000000	00:00:00.000000	00:00:00.000000
SZ	0	0	0	0	0	00:00:00.000000	00:00:00.000000	00:00:00.000000

QR TCB's CPU to Dispatch ratio is better (52.1%)

Figure 4-5 CICS TCB Statistics Reports, where the top shows a likely cause for concern

A CPU to dispatch ratio of less than 65% often is cause for further investigation. However, the ratio to be low for other reasons (in addition to paging).

This information can also be provided by CICS Performance Analyzer.

SDSF

Figure 4-6 shows an example of an SDSF Display Active view from which it can be seen (in the Paging column) that several of the CICS regions are seeing significant amounts of paging, with some of the most affected examples highlighted in circles.

SDSF DA MV2A	MV2A	PAG **	CPU/L	75/ 79	LINE 1-18 (39)									
COMMAND INPUT ==>					SCROLL ==> CSR									
NP	JOBNAME	StepName	ProcStep	JobID	Owner	C	Pos	DP	Real	Paging	SIO	CPU%	ASID	ASIDX
	CFDTSERA	CFSERVER		JOB19415	JOHNB	0	NS	C1	309	0.00	0.00	0.00	284	011C
	CICSA001	IYCUA001		JOB19423	JOHNB	0	NS	E2	267T	154.42	142.98	0.29	271	010F
	CICSA002	IYCUA002		JOB19424	JOHNB	0	NS	E2	266T	168.00	333.86	0.44	252	00FC
	CICSA003	IYCUA003		JOB19425	JOHNB	0	NS	E2	265T	0.71	965.12	1.18	229	00E5
	CICSA004	IYCUA004		JOB19426	JOHNB	0	NS	E2	266T	234.49	145.84	0.15	254	00FE
	CICSA005	IYCUA005		JOB19427	JOHNB	0	NS	E2	266T	15.01	2011.7	1.62	226	00E2
	CICSA006	IYCUA006		JOB19428	JOHNB	0	NS	E2	253T	67.92	1118.1	1.25	228	00E4
	CICSA007	IYCUA007		JOB19429	JOHNB	0	NS	E2	266T	0.71	864.32	0.96	211	00D3
	CICSA008	IYCUA008		JOB19430	JOHNB	0	NS	E2	267T	91.51	593.37	0.70	176	00B0

Figure 4-6 SDSF Display Active example

4.1.4 Using more real memory to open opportunities in CICS

Thus far in this chapter, we described how to identify situations where paging is occurring in CICS that can be addressed by providing more real storage. This section considers the opportunities that more real storage can make available. If paging is not an issue and more real storage is available, how might CICS benefit?

The availability of greater amounts of real memory can allow the following CICS facilities that use significant amounts of virtual storage to grow:

- ▶ CICS Main temporary storage (in 64-bit storage in the CICS address space).
- ▶ Large or many Shared Data Tables (in data spaces).
- ▶ CICS internal trace table (in 64-bit storage in the CICS address space).
- ▶ Applications that use CICS channels and containers to pass data between programs (internally held by CICS in 64-bit storage).
- ▶ Java heap storage, for CICS Java applications that require significant heap storage.
- ▶ CICS applications that access VSAM RLS (by using RLS buffer pools), or DB2 (by using DB2 buffer pools).
- ▶ Applications that use the CICS non-LE assembler 64-bit APIs (GETMAIN64 and PUT64 CONTAINER), which get 64-bit storage.

An important point to consider is if any of these facilities are deliberately under-used to avoid paging in CICS. If so, that decision can be revisited with the larger amounts of real memory that is available on IBM z13 (z13) and IBM z13s™ (z13s).

For more information, see the following resources:

- ▶ The CICS Performance Series Redpaper on Effective monitoring for CICS performance benchmarks:

<http://www.redbooks.ibm.com/abstracts/redp5170.html>

- ▶ *IBM CICS Performance Series: CICS TS for z/OS V5 Performance Report*, SG24-8298:
<http://www.redbooks.ibm.com/abstracts/sg248298.html>
- ▶ The CICS Performance Guide, and other CICS information, in IBM Knowledge Center:
<https://www.ibm.com/support/knowledgecenter/SSGMGV/welcome.html>

4.2 IBM MQ v8 64-bit buffer pools

In this section, we describe the benefits that can be achieved by using IBM MQ for z/OS workloads and operation when significantly more memory is made available.

Before IBM MQ for z/OS V8, the buffer pools were constrained to the 2 GB private region of the queue manager MSTR address space. Buffer pools act as a memory cache for messages between the application performing a *put* or *get* of messages, and the long-term queue storage, which uses disk in the form of pagesets for private local queues.

In normal operations, messages flow smoothly between putting and getting applications and queue depths stay low. However, an important reason for the use of IBM MQ allows for asynchronous communication between applications. The putters can continue to function even if the getters are going slowly, or the link to a remote system is inoperative. In these circumstances, messages are staged on queues so queue depths rise.

When the amount of message data that is held on a queue exceeds the size of the buffer pool, data is “cast out” to the pageset on a least recently used basis and must be read in again when needed for get processing. Performance is most optimal when message data is held in memory and read/write I/O to the pageset is at a minimum.

Because the size of all the buffer pools for the queue manager must add up to less than the 2 GB region size, there is little benefit in carving it up into many different pools. In most cases, 4 - 8 buffer pools are used.

The queues are associated with pagesets, which provide their backing storage. We expect to see dozens of pagesets in use to provide adequate worst case storage capacity. The net of this configuration is that IBM MQ administrators spend a considerable amount of effort matching pagesets to buffer pools and juggling limited storage between those pools according to performance requirements.

IBM MQ for z/OS V8 opens the queue manager to use vastly more memory than older releases. First, buffer pools can be located “above the bar” in 64-bit addressable memory, which allows them to be much larger. Second, because memory is no longer constrained, IBM MQ implementation constraints on the number of buffer pools is removed. You can now define a buffer pool per pageset.

The combination of more and larger buffer pools means that you can add more memory (rather than people) at the performance tuning problem, which is always the least expensive option. IBM MQ rewrites the recommendations on pageset to buffer pool assignment and sizing recommendations. For more information, see this website:

<https://ibm.biz/BdsbkD>

A recent IBM z13 announcement is available at the following website:

<https://ibm.biz/BdsbkD>

IBM z13 highlights increased memory over previous generations of machines. On the z13, “mega-memory” allows up to 10 TB of real storage in a machine.

IBM MQ for z/OS V8 provides an option to assign buffer pools to page-fixed memory. Essentially, this option dedicates memory to performance of IBM MQ queues because it allows IBM MQ code to bypass the code that is needed to page fix and unfix around read and write I/O calls between buffer pools and pagesets.

Page-fixed buffer pools can be used on earlier generations of hardware, but the IBM z13 brings a change in the price and amount of memory available, which makes the argument for trading memory to drive CPU cost savings compelling. The use of large buffer pools that are fixed in memory allows for the use of some of the z13 “mega-memory” to increase the throughput of your applications while reducing CPU costs.

Performance analysis

Consider a QRep workload as an example, in which a move from the current IBM MQ V7.1 configuration today is compared with a future configuration with IBM MQ V8 that uses large 64-bit buffer pools that are fixed in real storage. Although the measurements that are shown in Figure 4-7 on page 33 and Figure 4-8 on page 33 were made on a zEC12, z13 technology “mega-memory” makes it more feasible to dedicate this amount of memory to an IBM MQ workload.

As described in *WebSphere MQ v8.0 for z/OS Performance Report*¹, the measured workload moved a fixed quantity of data (12 GB) between a *putter* on one z/OS queue manager by using IBM MQ channels to a *getter* on a second z/OS queue manager. Such a workload is often seen in IIDER (QRep) deployments and cross-site or inter-enterprise message feeds.

Measurements were made when the data was sent as 10 KB messages (see Figure 4-7 on page 33) and 1 MB messages (see Figure 4-8 on page 33). We compared an IBM MQ V7.1 configuration that used 800 MB buffer pool (200,000 4k buffers) in 31-bit storage with a V8.0 configuration that used 4 GB buffer pool (1 million 4-k buffers) fixed in 64-bit storage. The same configuration is applicable in the queue manager at both ends of the channel, but the results presented Figure 4-7 on page 33 and Figure 4-8 on page 33 are from the sending side.

The following use cases were considered:

- ▶ Real time (the pair of bars on the right of Figure 4-7 on page 33 and Figure 4-8 on page 33): The channel is active and can move messages when the putting workload starts, which is the normal case. Queue depths stay low.
- ▶ Backed-up (the pair of bars on the left of Figure 4-7 on page 33 and Figure 4-8 on page 33): The channel is inactive when the workload starts; 2 GB of data is allowed to accumulate on the transmission queue before the channel is started. This configuration corresponds with the state after an outage of the channel or receiving system.

In Figure 4-7 on page 33, we compare a workload where the data is moved as 10 KB messages. The CPU cost per message is broken out by putter, queue manager, and getter (in this case, an IBM MQ channel) for each case. Cost savings per message are greater for the “backed up” case. The queue manager must do less work in V8 because more data is kept in memory.

¹ Found at: <http://www.ibm.com/support/docview.wss?uid=swg24038347>

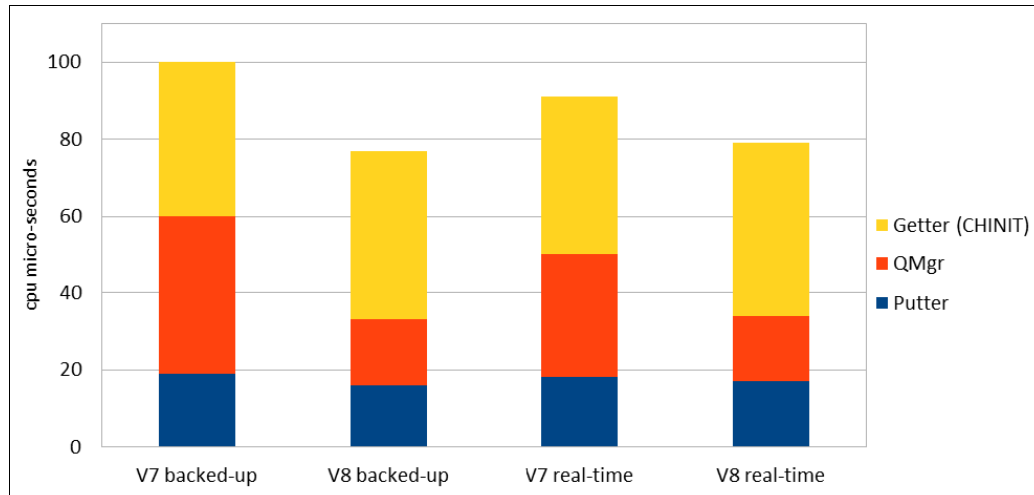


Figure 4-7 CPU cost per message: 10 KB messages

We might expect the real-time case to show little difference between V7 and V8 because the queue depths stay low and all messages fit in memory in both cases. In fact, because the messages are persistent, buffer pool contents are periodically written to the pageset, which provides a checkpoint to speed pageset recovery in case of a failure. That “write” I/O operation that is performed by the queue manager is reduced in cost when a page-fixed buffer pool is used.

Figure 4-8 shows the same overall pattern as the pattern that is shown in Figure 4-7, even though each message is 100x the size. Although the messages are 100x larger, the CPU cost per message increased only by a factor of 50x. It is more efficient to move data as a larger message.

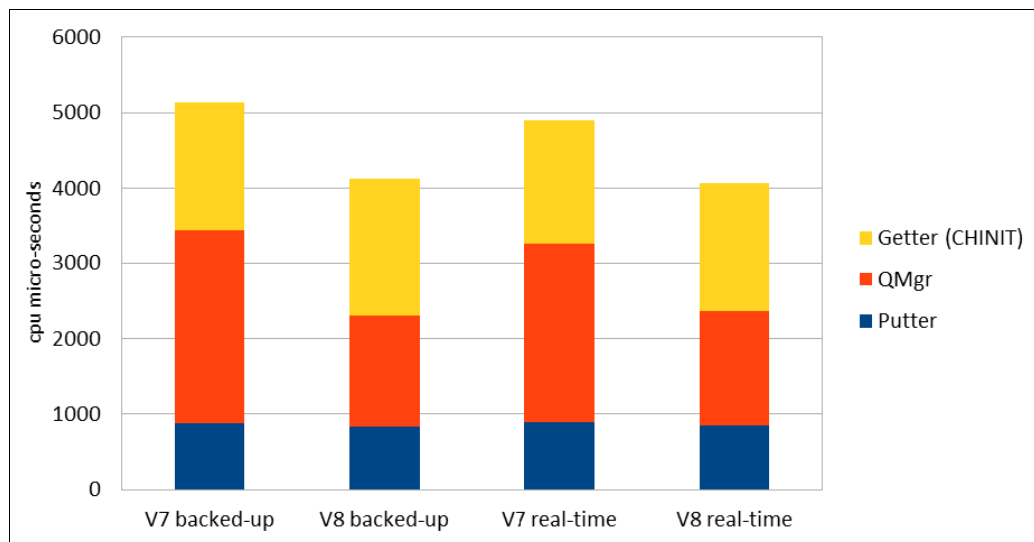


Figure 4-8 CPU cost per message: 1 MB message

IBM MQ V8 with large fixed buffer pools moves 12 GB of data significantly faster than V7.1, with total CPU cost of moving messages reduced by 10 - 20%. Data movement rates at least 20% faster (that is, reduced elapsed time), and in some cases, with deep queues (“backed-up” in Figure 4-7 and Figure 4-8), up to 120% faster were measured.

More information

For more information, see the following resources:

- ▶ *IBM MQ SupportPac MP1J: WebSphere MQ v8.0 for z/OS Performance Report:*
<http://www.ibm.com/support/docview.wss?uid=swg24038347>
- ▶ *IBM MQ SupportPac MP16: WebSphere MQ for z/OS Capacity planning & tuning:*
<http://www.ibm.com/support/docview.wss?uid=swg24007421>
- ▶ IBM Knowledge Center:
<https://ibm.biz/Bdsbkd>

4.2.1 How to right-size IBM MQ buffer pools

In the following sections, we describe the stages of an iterative process to follow to evaluate your system, identify how memory can be applied to buffer pools, make changes, and measure resulting improvements.

How much memory is the queue manager is using?

The IBM MQ buffer pools are in the IBM MQ MSTR address space. Before IBM MQ V8 and by default, these pools are allocated in 31-bit storage. The total storage for all buffer pools plus other queue manager allocations must be less than the 2 GB address space limit. In this step, we evaluate whether there is free space in the 2 GB region.

Evaluating whether there is free space can be accomplished by using one of the following methods:

- ▶ In an active system, IBM MQ periodically writes information about storage usage to the job log in a CSQY220I message, as shown in the following example output:

```
10.48.00 STC30242 CSQY220I @VTS1 CSQSCTL Queue manager storage usage: 299
299 local storage: used 992MB, free 527MB: above bar: used 197MB, free
299 >10GB
```

In this example, we see 527 MB is available.

- ▶ The same information can be found in IBM MQ's SMF 115 statistics records. The field that we want is QSRSAVAL. An example of SMF 115 data that is formatted by the MP1B ² is shown in the following example:

```
MVAA,VTS1,2015/12/08,11:16:31,VRM:710, Region >16MB size , 1520 MB
MVAA,VTS1,2015/12/08,11:16:31,VRM:710, >16MB Allocated user , 00eb0000, 14 MB
MVAA,VTS1,2015/12/08,11:16:31,VRM:710, >16MB Allocated system, 3d1a7000, 97MB
MVAA,VTS1,2015/12/08,11:16:31,VRM:710, >16MB Max limit ,5f000000, 1520 MB
MVAA,VTS1,2015/12/08,11:16:31,VRM:710, >16MB Start of region ,21000000,
MVAA,VTS1,2015/12/08,11:16:31,VRM:710, >16MB Current High ,21eb1000,
MVAA,VTS1,2015/12/08,11:16:31,VRM:710, >16MB not used , 527 MB
```

In this example, we see 527 MB is available in 31-bit storage.

² For more information, see this website:
<http://www.ibm.com/support/docview.wss?uid=swg24005907>

How much memory does the buffer pools use?

Next, we want to determine how much memory the buffer pools are using. We consider the following online and offline methods:

- In a running IBM MQ, use the **DISPLAY USAGE TYPE(PAGESET)** command. This command can be entered at a console or by using IBM MQ ISPF panels or the CSQUTIL batch utility program. The output shows information about the pagesets that were used and the buffer pools that were used to cache those pages, as shown in the following example:

```
CSQI065I !MQ51 Buffer pool attributes ...
  Buffer  Available  Stealable  Stealable  Page  Location
   pool   buffers   buffers   percentage class
-    0      2001      1976           98 4KB    BELOW
-    1      2001      1999           99 4KB    BELOW
-    2      1051      1050           99 4KB    BELOW
-    3      1051      1050           99 4KB    BELOW
-    4     25000         0            0 4KB    BELOW
End of buffer pool attributes
```

Each buffer is 4 KB. Therefore, the total buffer pool space in use is the sum of the buffers; in this example, 31,104 buffers or 121 MB.

- The same information can be found in SMF 115 statistics records. We want the QPSTNBUF field for each of the buffer pools. The following example shows output from MP1B formatter:

```
> 01 Buffs 200000 Low 9664 Now 10004 Getp 4604118 Getn 776616
   01 Rio 5034 STW 795724 TPW 554594 WIO 164024 IMW 33834
   01 DWT 17 DMC 229 STL 1226479 STLA 12 SOS 0
   01 Location - below bar
```

Which buffer pools are over- or under-sized?

IBM MQ buffer pools act as a cache for data that is stored on IBM MQ pagesets. When buffers that are assigned to the buffer pool can never hold data, the buffer pool is over-sized. We can safely reduce the size of the buffer pool, with no performance effect and return memory to the system for use elsewhere. A buffer pool is under-sized when it cannot hold the working set of messages that are active on a queue.

Finding oversized buffer pools

A buffer pool is over sized when it is larger than the data that the backing pagesets can hold. We suggest 120% as the threshold for being too large. This limit allows some room for growth (pageset expansion) and a use case to eliminate pageset write at checkpoint for non-persistent messaging.

The output of the IBM MQ command **DISPLAY USAGE TYPE(PAGESET)** can be used, as shown in the following example:

```
Page  Buffer  Total  Unused  Persistent  NonPersist  Expansion
set   pool   pages  pages  data pages  data pages  count
-    0     0    5038    4980         58         0 USER    0
-    1     1    1078    1061         17         0 USER    0
-    2     2    1078    1074          4         0 USER    0
-    3     3    1078    1075          0         3 USER    0
-    4     4   71634   71621         13         0 USER    0
-    5     5    1078    1078          0         0 USER    0
End of page set report
```

```
CSQP001I !MQ21 Buffer pool 0 has 1000 buffers, 991 (99%) stealable
```

```
CSQP001I !MQ21 Buffer pool 1 has 1000 buffers, 995 (99%) stealable
```

```

CSQP001I !MQ21 Buffer pool 2 has 5000 buffers, 4999 (99%) stealable
CSQP001I !MQ21 Buffer pool 3 has 5000 buffers, 4995 (99%) stealable
CSQP001I !MQ21 Buffer pool 4 has 5000 buffers, 4994 (99%) stealable
CSQP001I !MQ21 Buffer pool 5 has 1000 buffers, 999 (99%) stealable

```

The first part of the output shows the mapping of pagesets to buffer pools and the total number of pages that are contained in the page set. Buffer pool 2 is used for pageset 2 only. Pageset 2 also has a maximum size of 1078 pages. Applying our 120% rule, we calculate that buffer pool 2 can use only $1078 * 1.2 = 1294$ buffers. In the second part of the output, we can see that buffer pool contains 5000 buffers. It is over sized by $5000 - 1294 = 3706$ pages, which is memory that can be better used elsewhere.

Finding undersized buffer pools

A buffer pool is under sized when buffer manager statistics indicate that peak usage is greater than 85% of the pool (which might be tolerable for some batch-like applications; for example, loading a queue with messages for periodic processing), or 95% (all use cases). After the 95% full threshold is reached, IBM MQ APIs are delayed. Externally, you might notice a performance effect on applications, and IBM MQ writes the following message to the job log:

```
CSQP020E @VTS1 CSQP1RSW Buffer pool 1 is too small
```

To find under sized buffer pools, we must review IBM MQ buffer pool statistics, as shown in the following example from QPST:

```

> 01 Buffs  200000 Low   9664 Now    10004 Getp 4604118 Getn  776616
   01 Rio      5034 STW 795724 TPW   554594 WIO  164024 IMW   33834
   01 DWT      17 DMC   229 STL 1226479 STLA      12 SOS      0
   01 Location - below bar

```

We are interested in the following statistics:

► **QPSTNBUF**

Labeled “Buffs” in our example, which shows pool has 200,000 buffers.

► **QPSTCBSL**

Labeled “Low”, is low water mark of stealable (free) buffers (here, 9664). The low water mark is $9664 / 200000$, which is less than 5% of the buffer pool. The buffer pool was over 95% used and is undersized.

This buffer pool is under sized for the workload that is being requested of it. More indications of the affect this issue is having on applications can be seen by the statistics QPSTIMW, labeled IMW, and QPSTRIO, labeled Rio, which show that IBM MQ APIs were delayed for nearly 34000 synchronous write operations and 5000 read operations during the interval. These operations represent synchronous I/O.

Important: Synchronous I/O time causes API and application delays. Assuming an average 500 micro-seconds per I/O, we can estimate nearly 20 seconds $((33834 + 5034) * 500\text{us})$ of application delay in this SMF interval waiting for IBM MQ pageset I/O.

Changing a buffer pool

We gathered information about the system in the previous steps. The first step tells us how much memory is available for growing buffer pools. The second step tells us the total amount of memory that is allocated to buffer pools. The third step shows whether there are buffer pools that are oversized and we can release memory back to the system (add to the free area), or buffer pools that are undersized and so we want to make bigger.

Consider the following suggestions:

- Make only a single change at time and evaluate the effect on the system.
- Reduce the size of any oversized buffer pools before adding storage to other buffer pools.
- Increase the size of the buffer pool that is used by your most performance critical applications first.

Use the IBM MQ **ALTER BUFFPOOL(n) BUFFERS(xxxxxx)** command to change the size of a single buffer pool where xxxxxx is the target number of buffers. IBM MQ dynamically adds or removes buffers from the current buffer pool until the target is achieved. The changes to a buffer pool size are check-pointed and so are automatically reestablished after a queue manager restart.

For an oversized buffer pool, reduce the buffer pool to be 120% the sum of sizes of all the pagesets that use it.

For an undersized buffer pool, increase the buffer pool size in increments of 10% if there is sufficient storage. If you cannot use the **LOCATION(ABOVE)** command, do not let the total size of all buffer pools exceed 75 - 80% of the available 31-bit memory.

If you have IBM MQ Version 8 or higher, consider placing buffer pools above the bar. The same command is used to do perform this task, with the addition of a **LOCATION** attribute, as shown in the following example:

```
ALTER BUFFPOOL(n) BUFFERS(xxxxx) LOCATION(ABOVE)
```

Further performance benefits are available by requiring that fixed storage is used for buffer pools when they are above the 2 GB bar, as shown in the following example:

```
ALTER BUFFPOOL(n) BUFFERS(xxxxx) LOCATION(ABOVE) PAGECLAS(FIXED4KB)
```

Evaluating the improvement

Externally, did application performance improve or do the CSQP020E messages warning of small buffer pools still appear in the job log? Further insight to the affect of IBM MQ on application performance can be gained from IBM MQ's CLASS(3) Accounting trace that is written to SMF 116 records. Example 4-1 and Example 4-2 show two examples that are formatted with the MP1B supportpac utility, which demonstrates a buffer that is too small and then resized to be sufficiently large for the workload.

Example 4-1 Buffer pool too small

-MQ call-	N	ET	CT	Susp	LOGW	PSET
Get	: 949070	39	9	30	0	29
Put	: 949070	14	8	5 0		
Maximum depth encountered 10000						

Example 4-2 Buffer pool large enough

-MQ call-	N	ET	CT	Susp	LOGW	PSET
Get	: 1063808	10	9	0	0	0
Put	: 1063808	10	9	0	0	
Maximum depth encountered 10000						

The Susp field shows the average number of microseconds each API call is delayed for (here, 30 microseconds out of an average 39 microseconds per MQGET request). The cause of this delay is pageset.

Compare with the statistics that are shown after the buffer pool is increased, Susp drops to 0 and the average elapsed time per MQGET API improved by 75%.

Finally, review the buffer manager statistics records, particularly the QPSTIMW and QPSTRIO fields. For buffer pools that increased in size, the number of read and write operations to the pageset should decrease. This reduction might also be visible in channel path and volume utilization reports from RMF.



New applications and analytics

In this chapter, we describe how adding memory can facilitate the deployment of new applications by diminishing the need to restructure a workload environment to provide sufficient memory to accommodate the new work. Adding memory can alleviate some of the logistics associated with new application use.

5.1 Cognos Dynamic Cubes

IBM Cognos Dynamic Cubes adds an in-memory relational online analytical processing (OLAP) component to the dynamic query mode server to provide a multidimensional view of a relational data warehouse with accelerated performance. You can then perform OLAP analysis by using the Cognos Dynamic Cubes server.

Cognos Dynamic Cubes complements the Cognos query engine and enhances the ability of Cognos to enable faster decision making by keeping the required data in memory. When a cube is deployed and made available within a Cognos environment, it acts as the data source for dynamic cube applications. It manages all aspects of data retrieval and leverages memory to maximize responsiveness.

Large memory makes the IBM z13 an ideal host for Cognos Dynamic Cubes. Cubes can be deployed close to the data source. When more cubes are needed, they can be added without acquiring and configuring more host systems. Cognos cubes that are deployed on the IBM z13 also benefit from the z Systems Qualities of Service: Security, availability, reliability, while achieving the high performance and scalability that is required by business analytics applications.

The use of larger amounts of memory resulted in near linear scalability (that is, 95% scale efficiency) in internal measurements of Cognos Dynamic Cubes on z/OS. The number of Cognos instances, each containing two Dynamic Cubes and approximately 160 GB of memory, was increased from one instance (164 GB total memory) to three instances (511 GB total memory). This result achieved almost three times the transaction rate of the single Cognos instance, with little variation in response time (plus or minus 5%). the results are listed in Table 5-1.

Table 5-1 Cognos Dynamic Cubes results

Total # of Cognos Servers	Total # of Cubes	Total # of CPUs	Total memory used by Cognos (GB)	Ratio of Trans per CPU second	Ratio of response times
1	2	8	164	1	1
2	4	16	335	1.95	1.05
3	6	24	511	2.87	.96

These results are shown in Figure 5-1.

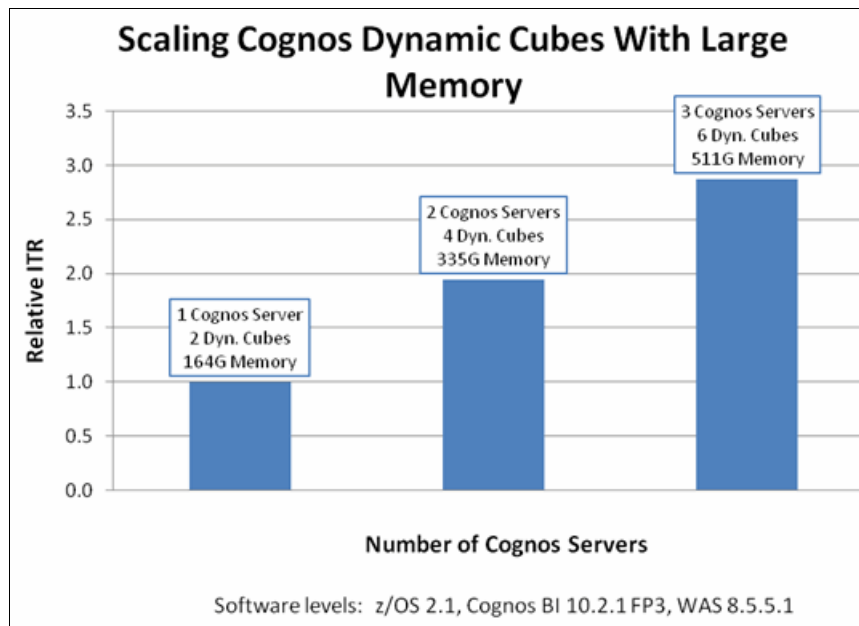


Figure 5-1 Cognos Dynamic Cubing results



Other benefits

In this chapter, we describe the use of DFSORT and its role in reducing work data set I/O and improving sort elapsed time. We also provide some test results for large memory performance by using the IBM z/OS Communications Server.

This chapter includes the following topics:

- ▶ 6.1, “DFSORT I/O reduction” on page 44
- ▶ 6.2, “z/OS Communications Server large memory performance results” on page 52

6.1 DFSORT I/O reduction

DFSORT is the IBM high-performance sort, merge, copy, analysis, and reporting product. DFSORT is an optional feature of z/OS. DFSORT adds the ability to perform faster and easier sorting, merging, copying, reporting, and analysis of your business information. It also provides versatile data handling at the record, fixed position or length, or variable position or length field, and bit level.

DFSORT optimizes the efficiency and speed with which operations are completed through synergy with processor, device, and system features (for example, memory objects, IBM Hiperspace™, data space, striping, compression, extended addressing, DASD and tape device architecture, processor memory, and processor cache) and other products (for example, the SAS System, COBOL, PL/I, and IDCAMS BLDINDEX).

DFSORT is an active user of memory for reducing I/O to intermediate work files. For many years, customers used DFSORT's data in memory techniques to reduce work data set I/O and improve sort elapsed times.

In recent releases, DFSORT improved its algorithms for evaluating and the use of available memory. In V1R12, DFSORT introduced the ability to use memory object work files, which function similar to Hiperspace work files.

A significant advantage of memory object over Hiperspace work files was an increase in the maximum usable by a single sort 32 - 64 GB. In V2R1, DFSORT increased the maximum memory object work file limit further to 1 TB. Other enhancements to V2R1 include better management of memory usage by concurrent sorts and changes to the algorithms to improve reliability and reduce the risk the effect to other workloads.

DFSORT performance is affected by the following factors, which makes it difficult to provide a single percentage improvement that is provided by data in memory:

- ▶ Sort input and output I/O speed. A sort can process only the input as fast as it can be read and write the sorted output as fast as it can be written. The speed at which DFSORT can read and write the data sets can be affected by the devices that are used for input, buffering options in effect, or contention in the I/O subsystem.
- ▶ For program-invoked sorts, there might be program logic that must be executed for each record that is passed to and from sort.
- ▶ CPU contention. Although DFSORT generally is not CPU intensive, delays for CPU can limit the ability of DFSORT to sort at optimal speeds in a constrained environment.
- ▶ Sort work disk I/O contention. Because DFSORT overlaps sort work I/O with input and output I/O, sort work I/O might not delay a sort unless the I/O contention drives response times high enough to cause DFSORT to wait.
- ▶ Current memory utilization. If the current memory configuration allows most sorts to complete in memory or a combination of memory and disk workspace, the benefit of extra memory is diminished.

The following example shows some of these factors and measurement data that can help assess the potential benefit of providing DFSORT with more memory resources.

In this example, we have a system with ample CPU resources and nominal CPU contention. A workload of 20 concurrent DFSORT test jobs is executed with varied degrees of memory resources. Total intermediate workspace requirement of the sorts is 300 GB and the workload is executed four times with varied amounts of available memory (0, 100, 200, and 800 GB).

Figure 6-1 and Figure 6-2 show the elapsed time comparison. The main chart is divided into two sets of 10 sort jobs.

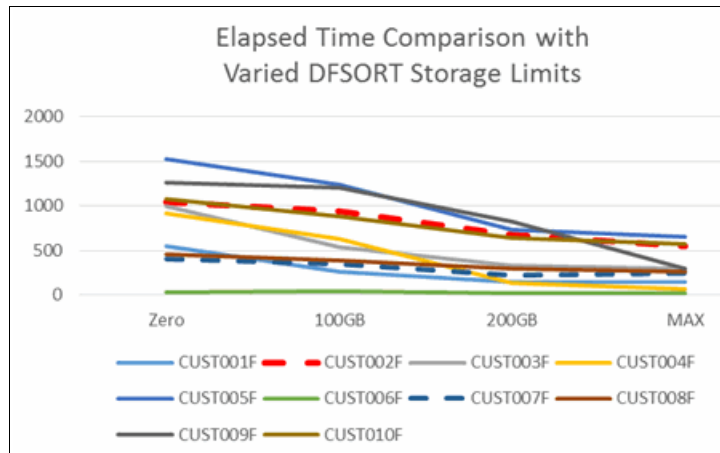


Figure 6-1 DFSORT elapsed time comparison

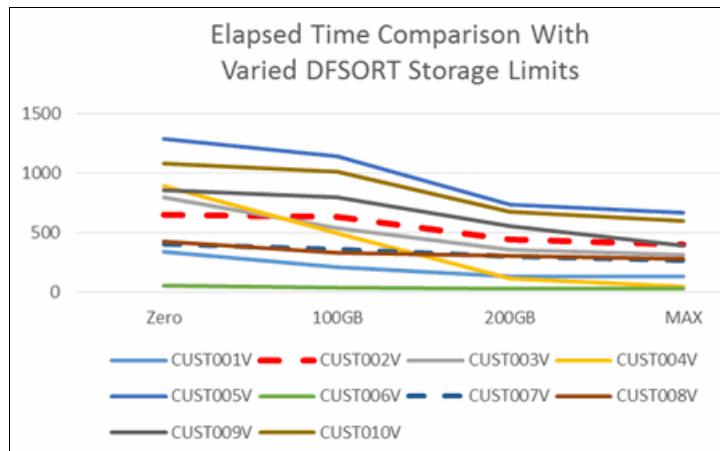


Figure 6-2 DFSORT elapsed time comparison

Elapsed time was improved from 0 - 100 GB by only 20% because of the sorts that used a combination of storage and disk workspace. With 200 GB, the improvement increased to 49% because most of the workspace can be contained in memory. With the maximum available, the improvement increased to 58% because all of the work data set I/O is eliminated.

Figure 6-3 and Figure 6-4 show the work I/O reduction.

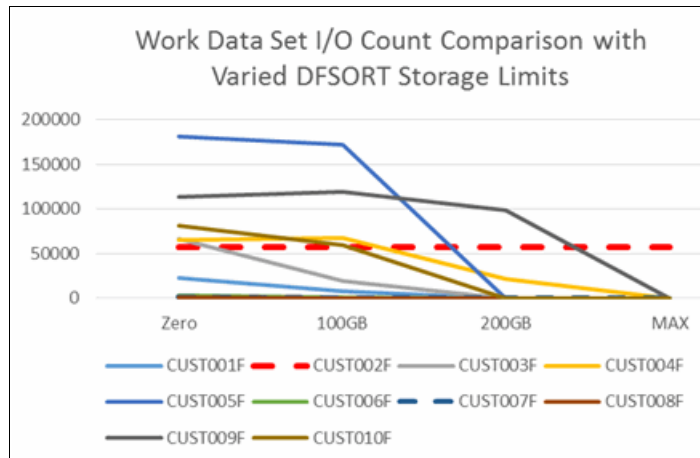


Figure 6-3 DFSORT I/O comparison

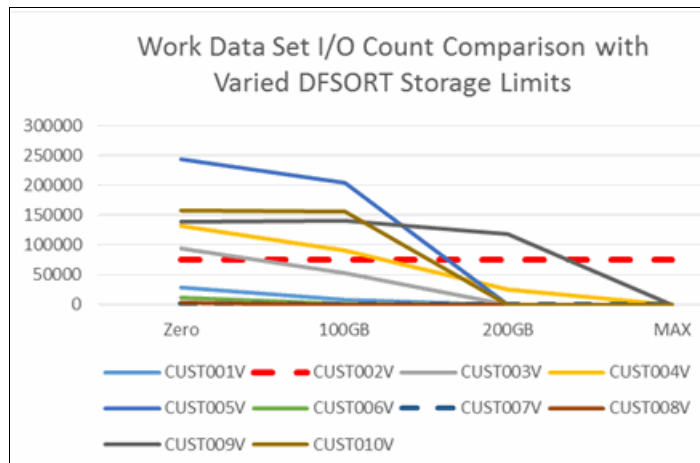


Figure 6-4 DFSORT I/O comparison

In Figure 6-3 and Figure 6-4, the dashed lines are flat for the CUST002x test cases and almost zero for the CUST007x test cases. This result is because the CUST002x test cases were executed with runtime options to prevent any use of data in memory and the CUST007x test cases use a DFSORT INCLUDE parameter that excludes most input records from being selected for sorting.

The significance of this result is that the performance of these sorts is almost dependent on the input and output I/O performance, yet they still experienced improved elapsed time when the DFSORT storage was increased. This increase was due to reduced contention in the I/O subsystem that was caused by the other sorts performing work data set I/O.

Although sorting in memory can improve elapsed times and reduce I/O contention, there is a CPU cost that is related to moving the pages to and from the memory objects or Hiperspaces, as shown in Figure 6-5 and Figure 6-6.

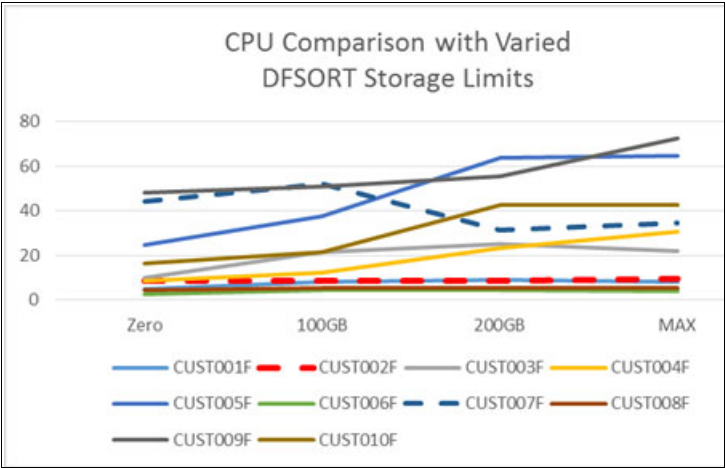


Figure 6-5 DFSORT CPU comparison

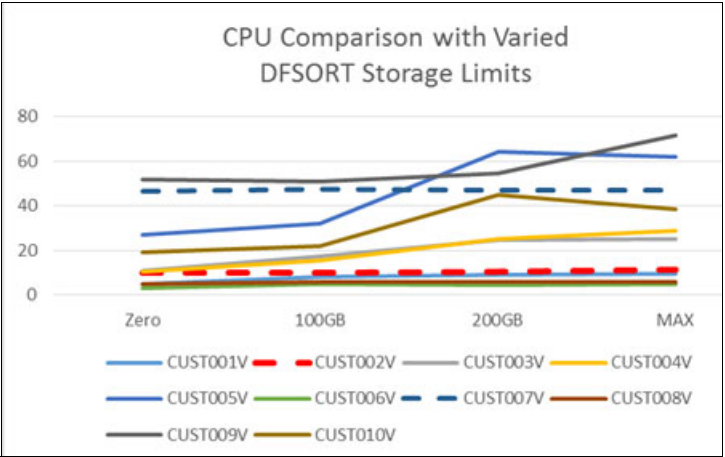


Figure 6-6 DFSORT CPU comparison

The CPU time from 0 - 100 GB increased by 21%. With 200 GB, the increase was 56% and the increase was 65% with the maximum available. The overall numbers are low; therefore, although a 65% increase might seem high, the CPU cost of executing the sorts remains less than 10% of the elapsed time.

The RMF Cache statistics also provide some insight into the potential for elapsed time improvements, as shown in Figure 6-7.

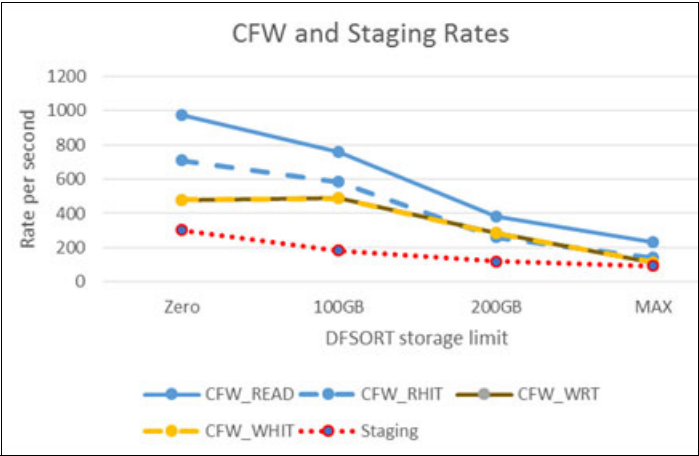


Figure 6-7 DFSORT storage statistics

When DFSORT memory was limited, the high sort work data set I/O stressed the disk cache. The Cache Fast Write read hit (CFW_RHIT) rate is lower than the total read rate and the Staging rate is much higher. This result is because the cache is constrained. DFSORT initially writes the work data sets by using Cache Fast Write. However, by the time it reads them back in, they were forced out of the cache and must be staged back in. As we increase the memory usage of DFSORT, the staging rate reduces and the CFW hit percentage improves because of less stress on the cache.

We also see the benefits in terms of I/O disconnect time, as shown in Figure 6-8 and Figure 6-9 on page 49.

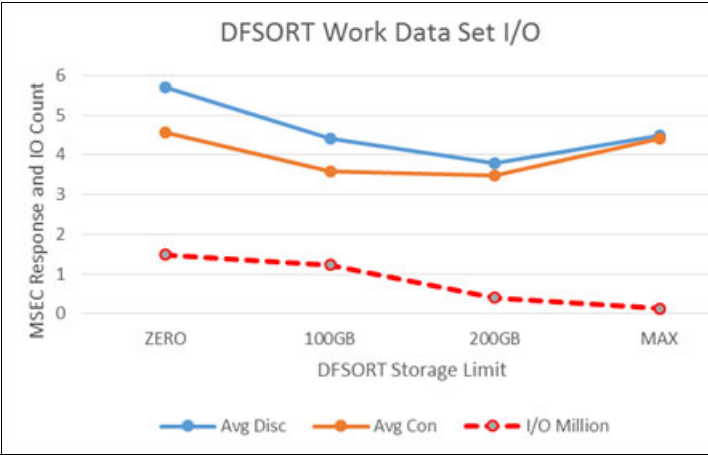


Figure 6-8 DFSORT I/O response time statistics

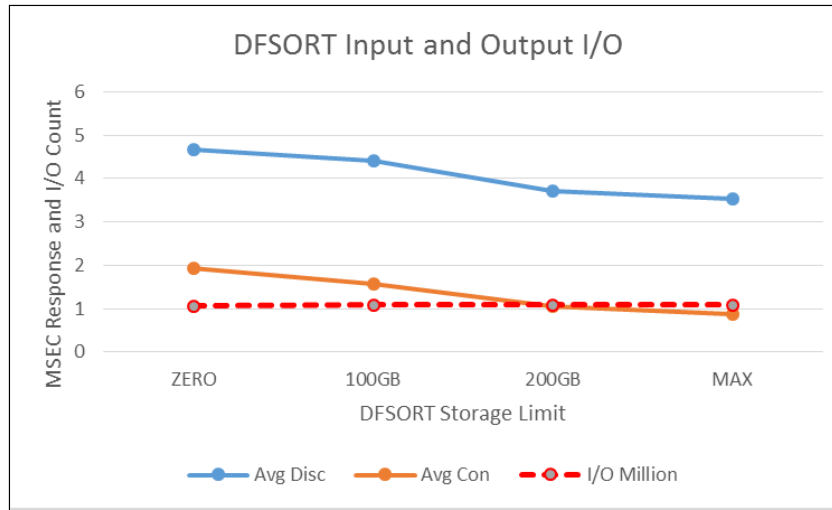


Figure 6-9 DFSORT I/O response time statistics

The number of I/Os to the input and output data sets is not affected by the use of more memory but the response time still improves as a result of the reduced contention in the cache subsystem when the work data set I/O is eliminated.

When assessing the potential benefit of large memory on DFSORT in your environment, the following questions must be answered:

- ▶ How much memory is available to DFSORT?
 - We must understand if there is enough memory available for DFSORT to use.
 - Are DFSORT installation defaults tailored to fully use memory?
- ▶ How large are the sorts that are executing?
 - Are the sorts large enough to use the available memory?
 - Are they so large that the available memory does not make much difference?
- ▶ Are the sorts being delayed by I/O?
 - If the existing memory is enough to eliminate the work data set I/O, more memory is not going to have much effect.
 - If the I/O subsystem is performing well, the I/O might not be delaying the sorts.

RMF can be useful in answering the first and third questions. RMF overview reports provide a useful view of available storage resources and workload delays. An RMF overview report that shows the number of available frames on a system by interval along with low, medium, and high impact frames is shown in Figure 6-10 on page 50.

RMF OVERVIEW REPORT										
z/OS V1R13			SYSTEM ID AL18				START 07/24/2012			
			RPT VERSION V1R13 RMF				END 07/24/2012			
NUMBER OF INTERVALS 6			TOTAL LENGTH OF INTERVALS 01.30.00							
DATE	TIME	INT	AVGAVAIL	AVGLOW	AVGMED	AVGHIGH	CSTOR	DEMPAGE	PFAULTS	MAXH
MM/DD	HH.MM.SS	HH.MM.SS								
07/24	06.45.00	00.15.00	6769689	15471366	20831744	24630832	23592960	0.000	0.000	2038195
07/24	07.00.00	00.14.59	7580352	16492716	21108080	24621168	23592960	0.000	0.000	1322386
07/24	07.15.00	00.14.59	6811607	15447348	20853120	24636768	23592960	0.000	0.000	
07/24	07.30.00	00.15.00	7280219	16158817	21054560	24638608	23592960	0.000	0.000	2078631
07/24	07.45.00	00.14.59	8519231	16559088	21187536	24627536	23592960	0.000	0.000	1028866

Figure 6-10 RMF Overview Report

Other useful information includes the configured storage frames, the demand page rate, number of page faults and the maximum Hiperspace pages that are used during each RMF reporting interval. In this example, we can see the system consistently has over 6.5 million frames (24 GB) of available main storage with no paging activity.

In the RMF overview report shown in Figure 6-11, a batch service class is profiled to understand where the most time is spent. This service class has a high CPU delay with minimal I/O usage and delay, which indicates that sorts that are executed in this service class are not likely to benefit from reducing work I/O with memory.

RMF OVERVIEW REPORT										
z/OS V1R13			SYSPLX ID PLEX1			START 01/05/2016-00.00.00				
			RPT VERSION V2R2 RMF			END 01/05/2016-23.50.00				
NUMBER OF INTERVALS 143			TOTAL LENGTH OF INTERVALS 23.50.00							
DATE	TIME	INT	CPUUSGP	CPUDLYP	IOUSGP	IODLYP	SSCHRT			
MM/DD	HH.MM.SS	HH.MM.SS								
01/05	00.00.00	00.09.59	3.7	73.5	4.0	0.5	7209			
01/05	00.10.00	00.10.00	2.6	75.6	3.3	0.4	5473			
01/05	00.20.00	00.10.00	3.3	73.8	4.5	0.5	7424			
01/05	00.30.00	00.10.00	3.1	71.2	5.8	0.7	10167			
01/05	00.40.00	00.10.00	4.3	70.3	7.8	0.8	15075			
01/05	00.50.00	00.10.00	5.8	66.0	7.6	0.7	12912			

Figure 6-11 RMF overview report showing batch service class

The same report (as shown in Figure 6-12) for a different batch service class indicates a more significant I/O usage and delay with minimal CPU delay. Sorts that execute in this service class are more likely to benefit from the use of memory to reduce work data set I/O.

RMF OVERVIEW REPORT										
z/OS V1R13			SYSPLX ID PLEX1				START 01/05/2016-00.00.00		INTERVAL 00.10.00	
			RPT VERSION V2R2 RMF				END 01/05/2016-23.50.00			
NUMBER OF INTERVALS 143			TOTAL LENGTH OF INTERVALS 23.50.00							
DATE	TIME	INT	CPUUSGP	CPUDLYP	IOUSGP	IODLYP	SSCHRT			
MM/DD	HH.MM.SS	HH.MM.SS								
01/05	00.00.00	00.09.59	17.3	10.2	23.7	9.2	10288			
01/05	00.10.00	00.10.00	21.4	16.6	20.9	8.7	9463			
01/05	00.20.00	00.10.00	29.8	15.1	22.7	9.1	6839			
01/05	00.30.00	00.10.00	33.3	9.4	23.0	9.1	6768			
01/05	00.40.00	00.10.00	36.4	7.8	22.5	9.2	5291			
01/05	00.50.00	00.10.00	21.0	9.5	23.3	8.9	7655			

Figure 6-12 RMF overview report showing a different batch service class

DFSORT generates SMF records that provide useful information about the sort activity in your system. The provided information includes the total records and bytes sorted, DFSORT installation and runtime parameters that are in effect, the total storage currently being used by DFSORT, sort elapsed times, and sort CPU times.

All the fields in the DFSORT SMF records are documented in the DFSORT Initialization and Tuning Guide (V2R2 Publication SC26-7524-03, Appendix D. SMF Type-16 Record). Although DFSORT does not provide a tool for reporting on the SMF records, there are tools, such as IBM's Tivoli Decision Support for z/OS and Merrill Consultants' SAS/MXG.

Figure 6-13 shows a sample report that was generated from the DFSORT SMF type 16 data. It provides insight into the size of the largest sorts along with the amount of work data set space and I/Os that are associated with them. These sorts often benefit if available memory were significantly increased and DFSORT can use it to reduce I/O to the work data sets.

Day	Timestamp	Pgm	Elapse Secs	CPU Secs	GB Sorted	Hiperspace GB Used	Work Dataset Tracks Used	Work Dataset I/Os
5	01:00:32.95	SORT	7,917.56	1,458.31	804.00	0.00	16,607,700	3,335,113
5	05:28:35.67	FPECMAIN	8,872.90	1,790.92	131.71	4.56	2,604,645	691,464
5	05:08:07.83	FPECMAIN	7,647.54	1,185.19	131.71	5.14	2,592,870	688,375
5	04:54:27.24	FPECMAIN	6,829.91	870.57	131.71	3.81	2,620,005	695,568
5	05:53:25.85	SORT	2,878.45	409.10	98.03	19.60	1,755,435	344,779
5	01:30:25.85	SORT	5,697.02	207.89	96.87	15.70	1,688,205	448,709
4	22:36:18.94	SORT	4,251.13	356.59	96.84	0.00	3,684,840	1,551,771
4	22:54:28.36	SORT	3,393.15	194.10	96.84	6.75	1,861,425	496,169
4	21:47:06.76	SORT	1,798.95	227.23	96.84	12.94	1,733,760	464,844
4	21:57:53.83	SORT	1,645.68	208.14	96.84	0.00	1,881,945	505,577
4	08:51:02.98	SORT	1,608.13	564.34	91.91	27.42	1,498,380	310,963
4	19:05:23.70	SORT	433.59	138.63	89.99	3.06	0	6
4	18:58:02.43	SORT	1,104.03	242.76	89.99	22.14	1,496,235	338,890
4	19:45:53.41	SORT	2,391.63	275.27	87.35	16.02	1,580,355	318,978
4	23:46:05.14	SORT	573.64	72.78	84.23	0.13	0	9
4	13:42:15.95	SORT	1,644.30	335.71	82.55	17.19	1,476,195	290,105
4	23:27:46.88	SORT	4,341.38	174.93	81.36	0.00	1,703,655	333,879
4	20:45:29.76	SORT	2,580.74	175.29	68.74	0.00	1,420,935	278,811
5	01:47:21.18	SORT	1,555.15	162.26	64.95	0.00	34,395	4,549
5	03:00:34.94	SORT	1,175.07	122.38	60.47	10.05	1,109,670	225,080

Figure 6-13 DFSORT SMF type 16 data report

DFSORT can benefit from increased use of memory to eliminate work data set I/O. However, the potential benefit varies based on the current resource usage and contention. DFSORT workloads that feature high I/O contention and low CPU contention are more likely to experience high elapsed-time improvement.

6.1.1 Conclusion

This section highlighted some of the performance benefits that were observed while growing memory size for a number of z Systems software functions. Some of these gains can be substantial in the correct environment, as was noted in the DB2, MDM, and IBM MQ cases.

Although customer environments might experience various performance benefits as compared to some of the examples that are presented in this section, there seems to be enough evidence to suggest that configuring more memory can be a positive enhancement for many installations because of reduced I/O rates, improved transaction response time, and in some cases reduced CPU time.

For more information about DFSORT, see *DFSORT Tuning Guide*, SC23-6882. Consider the following points:

- ▶ In Chapter 4, the section entitled, “Hipersorting, memory object sorting and data space”, includes information and recommendations for setting DFSORT’s installation defaults to control DFSORT’s use of main storage resources.
- ▶ In Chapter 5, the following sections provide more information about each of DFSORT’s methods for using main storage:
 - Memory object sorting
 - Hipersorting
 - Sorting with data space

6.2 z/OS Communications Server large memory performance results

In this section, we review our test results for large memory performance by using the IBM z/OS Communications Server.

This first test shows the advantage of the use of large application and TCP send and receive buffers for streaming (bulk data) transfers. Our streaming workloads loop between the client sending 1 byte of data to the server and the server responding with 20 MB. This configuration tests the network performance with bulk data transfers. This test was done in our lab on zEC12 systems that were close together (low latency).

In our labs, the use of large application and TCP send and receive buffers for streaming workloads provides throughput, response time, and CPU utilization improvements. We measured a 12% throughput improvement, a z/OS CPU reduction on the sending side of 8.45%, and a z/OS CPU reduction on the receiving side of over 16%. These improvements are shown in Figure 6-14.

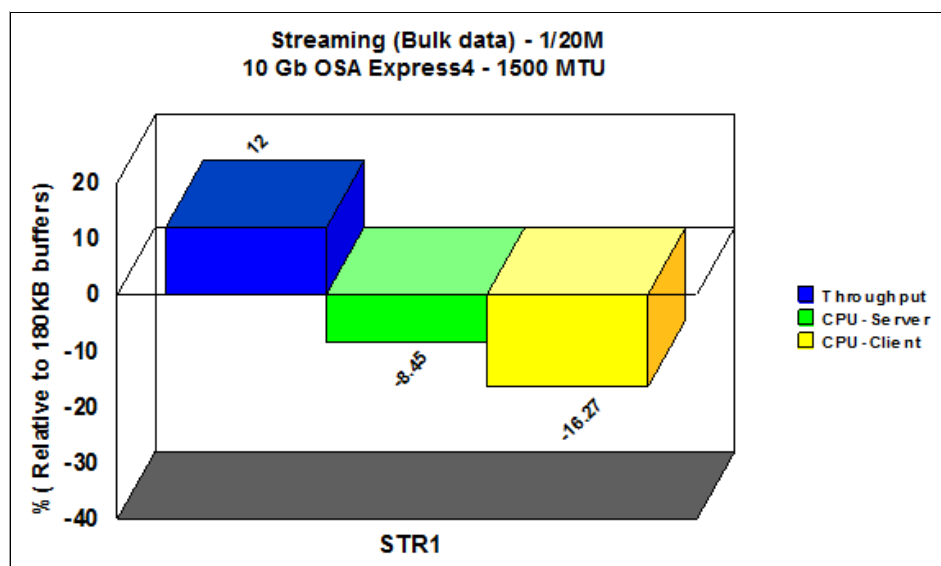


Figure 6-14 Results of a send and receive for bulk data transfers

In this test, we compared sending bulk data over two types of TCP connections. The first (base) connection used more traditional application and TCP send and receive buffer sizes. The application used 64 KB send and receive sizes. A buffer size of 180 KB was used for the TCP send and receive buffers. The second connection used larger application and TCP buffers. The application used 1 MB send and receive sizes and 2 MB was used for the TCP send and receive buffers. This size is the maximum size that is allowed by z/OS TCP.

Both connections used an OSA Express4S 10 Gb configured with a 1500 MTU and NOSEGMENTATIONOFFLoad. This configuration is the most typical OSA configuration.

In this test, we ran a streaming workload (1 TCP connection) between two z/OS systems. The client sends 1 byte of data to the server. The server responds by streaming 20 million bytes back to the client in a loop for several minutes. In both connections, the receiving application specified `msg_waitall` on the `recv()` API call. The `msg_waitall` flag on the `recv()` call instructs the TCP layer not to post completion of the socket `recv()` call until all of the requested data is present. This flag makes receive processing efficient because it minimizes the number of socket-receive API crossings.

The next test shows the advantage of the use of a large TCP receive buffer for bulk data transfers over large distances (high latency). In this test, we used FTP to repeatedly transfer a 2.8 GB file over a period. The transmitting node is an IBM AIX® machine in Boulder, Colorado. The receiving node is a z/OS system IBM z10 in Research Triangle Park, NC. The z/OS system (the receiver) uses a function that is named Dynamic Right Sizing (DRS), which self-tunes the TCP receive window to match the Round Trip Time of the connection. With Dynamic Right Sizing, the TCP receive buffer can grow up to 2 MB. This configuration allows the “pipe” to remain full, which leads to an increase in throughput.

In this test, we compare the throughput over 2.5 hours, transferring the 2.8 GB file in a loop to a z/OS system by using Dynamic Right Sizing versus a z/OS system that uses a fixed TCP receive buffer of 64 KB. As Figure 6-15 shows, we transferred 78.3 GB with the larger TCP receive buffer compared to 39.6 GB with the fixed, smaller TCP receive buffer.

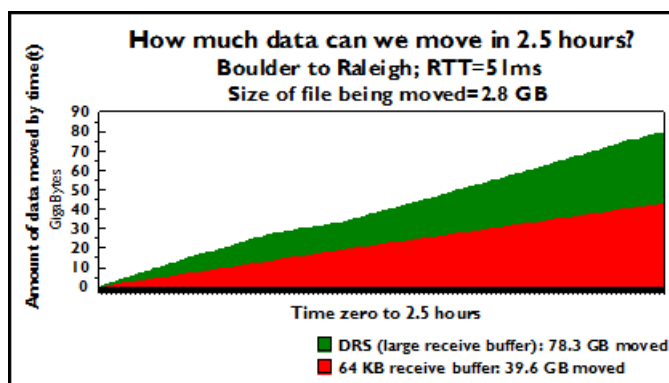


Figure 6-15 Large TCP receive buffer results

In summary, the use of large TCP buffers for sending and receiving data can improve throughput and overall networking-related CPU usage. This result is especially true for streaming type (bulk) workloads that traverse high latency networks. However, in many cases, you must configure the larger sizes. The default values are only 64 KB. This value can be too small for streaming connections. Beginning in z/OS V2R1, you can configure TCP send and receive buffers as large as 2 MB.

Figure 6-16 shows the output from a sample **NETSTAT,CONFIG** command. Also shown is the current TCP maximum send and receive buffer sizes and the default buffer sizes. In this example, the maximum receive and send buffers are only 256 KB. The default send and receive buffers sizes are only 64 KB.

D TCPIP,TCPCS1,N,CONFIG			
EZD0101I NETSTAT CS V2R3 TCPCS1 405			
TCP CONFIGURATION TABLE:			
DEFAULTRCVBUFSIZE:	00065536	DEFAULTSNDBUFSIZE:	00065536
DEFLTMAXRCVBUFSIZE:	00262144	SOMAXCONN:	0000001024
MAXRETRANSMITTIME:	120.000	MINRETRANSMITTIME:	0.500
ROUNDTRIPGAIN:	0.125	VARIANCEGAIN:	0.250
VARIANCEMULTIPLIER:	2.000	MAXSEGLIFETIME:	30.000
DEFAULTKEEPALIVE:	00000120	DELAYACK:	YES
RESTRICTLOWPORT:	YES	SENDGARBAGE:	NO
TCPTIMESTAMP:	YES	FINWAIT2TIME:	0600
TTLS:	NO	EPHEMERALPORTS:	01024-65535
SELECTIVEACK:	NO	TIMEWAITINTERVAL:	060
DEFLTMAXSNDBUFSIZE:	00262144	RETRANSMITATTEMPT:	015

Figure 6-16 Example **NETSTAT,CONFIG** command

You can also monitor individual connections to ensure that the correct buffer sizes are being used. For example, you might want a large TCP receive buffer for the receiving side of a streaming connection. The **NETSTAT,ALL** command (shown in Figure 6-17) can be used to display the values in use for a particular TCP connection. In this example, the TCP receive buffer for this connection is 2 MB. The TCP send buffer is the default value, 64 KB. This setting is appropriate for the receiving side of a streaming connection because it often does not send large message sizes.

D TCPIP,TCPCS1,N,ALL			
CLIENT NAME: USER33		CLIENT ID: 0000002F	
LOCAL SOCKET: 9.10.120.100..9999			
FOREIGN SOCKET: 9.10.240.100..1024			
BYTESIN:	00000000098473884485		
BYTESOUT:	00000000000000005093		
SEGMENTSIN:	00000000001549388483		
SEGMENTSOUT:	0000000000000000356		
...			
QOSPOLICY:	NO		
ROUTINGPOLICY:	NO		
RECEIVEBUFFERSIZE:	0002097152	SENDBUFFERSIZE:	0000065536
CONNECTIONSIN:	0000000001	CONNECTIONSDROPPED:	0000000000
MAXIMUMBACKLOG:	0000000010	CONNECTIONFLOOD:	NO

Figure 6-17 Monitoring to ensure that the correct buffer sizes are being used

The results that were obtained in other configurations or operating system environments might vary significantly, depending upon the environments that are used. Therefore, no guarantee can be given that other implementations achieve performance gains that are equivalent to those gains that are described in this chapter. Users should verify the applicable data for their specific environment.

For more information about the IBM z/OS Communications Server and bulk data, see this website:

<https://ibm.biz/BdspMd>



Conclusion

This paper highlighted some of the performance benefits that were observed while growing memory size for several IBM z Systems software functions. Some of these gains can be substantial in the right environment, as was noted in the DB2, MDM, and IBM MQ cases.

Though customer environments can experience various performance benefits (as compared to some of the examples presented in this IBM Redpaper publication), there seems to be enough evidence to suggest that configuring more memory can be a positive enhancement for many installations because of reduced I/O rates, improved transaction response time, and in some cases reduced CPU time.

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this paper.

IBM Redbooks

The following IBM Redbooks publication *Architect's Guide to IBM CICS on System z*, SG24-8067, provides more information about the topic in this document and is available at this website:

<http://www.redbooks.ibm.com/abstracts/sg248067.html>

You can search for, view, download, or order other Redbooks, Redpapers, Web Docs, draft, and other materials, at the following website:

ibm.com/redbooks

Online resources

The following websites also are relevant as further information sources:

- ▶ *IBM z Systems: Performance Report on Exploiting Large Memory for DB2 Buffer Pools with SAP:*

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102461>

- ▶ IBM MQ SupportPac MP1J: WebSphere MQ v8.0 for z/OS Performance Report

This report is the source material that was used for the performance measurements that are described in this paper:

<http://www.ibm.com/support/docview.wss?uid=swg24038347>

- ▶ IBM MQ SupportPac MP16: WebSphere MQ for z/OS Capacity planning & tuning

This publication is the document buffer pool statistics that were used for sizing purposes:

<http://www.ibm.com/support/docview.wss?uid=swg24007421>

- ▶ IBM Knowledge Center:

<https://ibm.biz/Bdsbkd>

- ▶ DFSORT Tuning Guide:

<http://publibz.boulder.ibm.com/epubs/pdf/ice2ct10.pdf>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



REDP-5238-01

ISBN 0738455962

Printed in U.S.A.

Get connected

