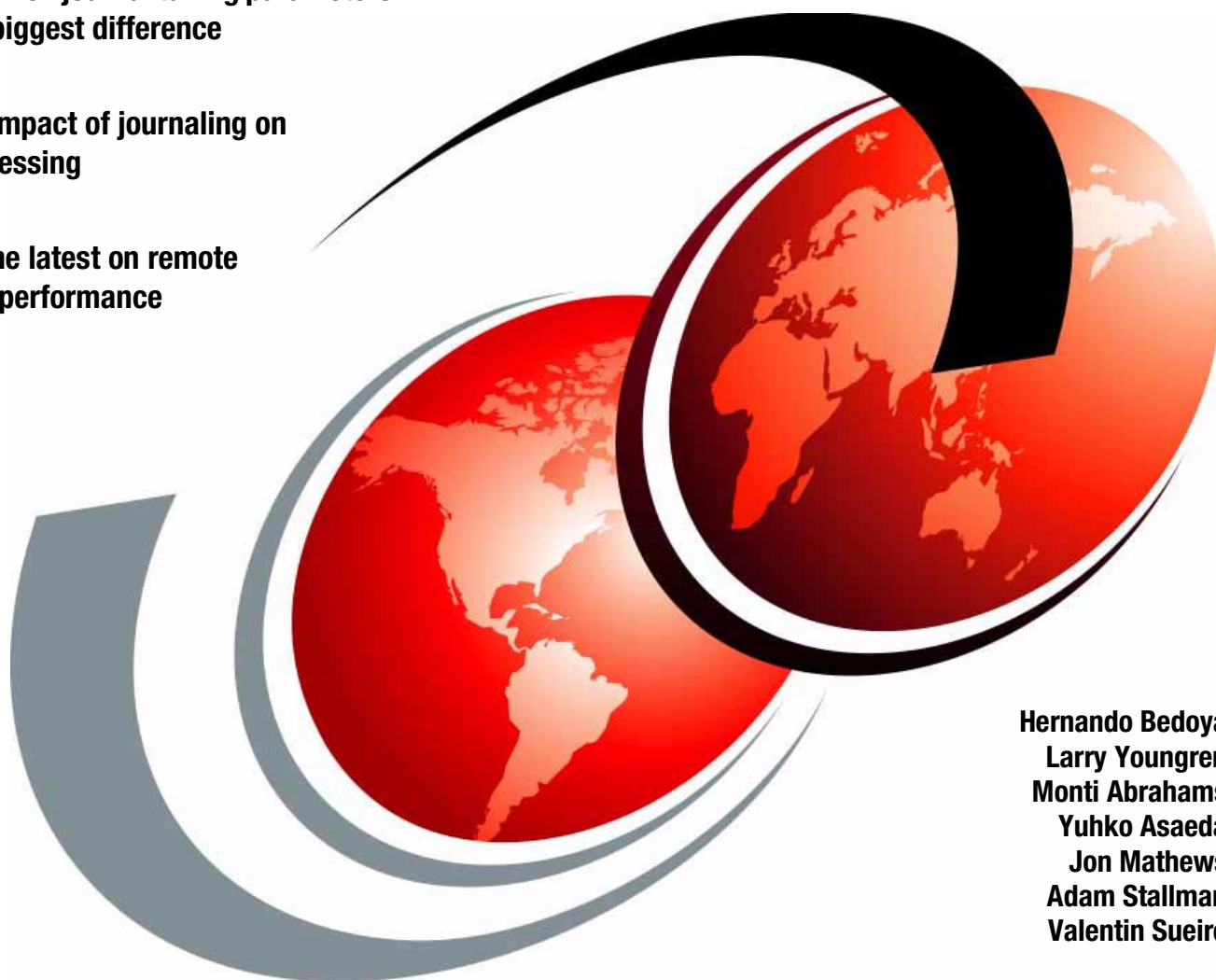


Striving for Optimal Journal Performance on DB2 Universal Database for iSeries

Discover which journal tuning parameters
make the biggest difference

Learn the impact of journaling on
batch processing

Discover the latest on remote
journaling performance



Hernando Bedoya
Larry Youngren
Monti Abrahams
Yuhko Asaeda
Jon Mathews
Adam Stallman
Valentin Sueiro

Redbooks



International Technical Support Organization

**Striving for Optimal Journal Performance
on DB2 Universal Database for iSeries**

May 2002

Take Note! Before using this information and the product it supports, be sure to read the general information in “Notices” on page xi.

First Edition (May 2002)

This edition applies to Version 5 Release 1 of OS/400, Program Number 5722-SS1.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2002. All rights reserved.

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Contents	iii
Figures	vii
Tables	ix
Notices	xi
Trademarks	xii
Preface	xiii
The team that wrote this redbook	xiii
Special notice	xv
Comments welcome	xv
Part 1. Background	1
Chapter 1. An introduction to journaling performance	3
1.1 A customer call to a journaling expert: Interview	4
1.2 The journal environment: Background	14
1.3 What's new in V5R1	18
1.3.1 A little history	18
1.3.2 Minimize the Entry-Specific Data parameter	18
1.3.3 Journaling for non-database objects	19
1.3.4 Journal PRPQs (5799-BJC, 5799-AJC)	21
1.3.5 Summary of V5R1 enhancements	26
Chapter 2. The testing environment	27
2.1 The hardware	28
2.1.1 Teraplex Center	29
2.1.2 The "Cluster Lab"	41
2.1.3 The ITSO laboratory	50
2.2 Application environments	51
2.2.1 Banesco's end-of-day batch process	52
2.2.2 Retail store application	53
Part 2. Performance considerations	55
Chapter 3. Hardware choices	57
3.1 Factors to be investigated	58
3.2 The testing scenarios	58
3.3 Test results and findings	65
3.4 Conclusions	72
Chapter 4. Journaling tuning	81
4.1 System Managed Access Path Protection (SMAPP)	82
4.2 Journal parameters	85
4.2.1 CRTJRN and CHGJRN parameters	85
4.2.2 Journal recovery ratio (JORECRA)	87
4.2.3 Changing and managing journal receivers	87
4.3 Recovering at a hot site - APYJRNCHG	89

4.4 The testing environment	89
4.5 The testing scenarios	90
4.6 Test results and findings	93
4.6.1 Recovering from a hot site	104
4.6.2 Throughput impact (MINENTDTA with BLOBs)	105
4.7 Focus Program tests	106
4.7.1 Focus Program results	107
Chapter 5. Application and software environment tuning	109
5.1 Application tuning	110
5.2 Analyzing application performance	110
5.3 Increasing performance	111
5.3.1 Batch job parallelism	111
5.3.2 Database indexes	112
5.3.3 Minimize and optimize database table extensions	114
5.3.4 Managing your open data path buffer	115
5.3.5 Symmetric Multiprocessing (SMP)	115
5.3.6 Enable Concurrent Writes (ECW)	116
5.3.7 Be journal friendly	118
5.3.8 Commitment control	118
5.4 Performance test scenario	119
5.5 Results and conclusions	120
Chapter 6. Remote journaling performance	123
6.1 Introduction	124
6.2 Remote journal basics	124
6.2.1 Asynchronously maintained remote journals	125
6.2.2 Synchronously maintained remote journals	126
6.2.3 The potential communications bottleneck	127
6.2.4 High Availability Business Partners (HA BP) solutions	128
6.3 Test environment	129
6.4 Performance benchmark scenario	130
6.5 Conclusions and findings	132
6.5.1 Communication infrastructure comparison	132
6.5.2 Comparison between synchronous and asynchronous remote journal	133
6.5.3 Comparison regarding the (5799-BJC) PRPQ	134
6.5.4 Comparison synchronous RJ: send volume versus acknowledge volume	134
6.5.5 Comparison among various journal receiver size option parameters	135
6.5.6 Comparison between remote journal asynchronous and HA solution	136
6.5.7 Unsent journal entries comparing old and new HA BP approach	138
6.5.8 Conclusions for HA BP solution	140
6.5.9 Catch-up mode tests	141
Part 3. Conclusions and recommendations	147
Chapter 7. Conclusion	149
7.1 General performance considerations	150
Appendix A. How to calculate the fast write percentage	155
Instructions	156
Related publications	157
IBM Redbooks	157
Other resources	157

Referenced Web sites	157
How to get IBM Redbooks	158
IBM Redbooks collections.	158
Index	159

Figures

1-1	The journal environment	15
1-2	Journal bundling.	16
1-3	Start journaling for data area objects.	19
1-4	Start journaling for IFS objects.	20
1-5	Journal Caching PRPQ	22
2-1	Hardware overview	28
2-2	iSeries Operations Navigator.	30
2-3	iSeries hardware at the Teraplex Center	31
2-4	LPAR overview	32
2-5	Primary partition configuration	33
2-6	Partition 2 (LPAR2) configuration	34
2-7	Partition 3 (LPAR3) configuration	35
2-8	Partition 2 (LPAR2) disk configuration.	36
2-9	Partition 3 (LPAR3) disk configuration.	36
2-10	User ASP2 in partition 2 (LPAR2)	37
2-11	User ASP4 in partition 2 (LPAR2)	38
2-12	User ASP2 in partition 3 (LPAR3)	39
2-13	User ASP3 in partition 3 (LPAR3)	40
2-14	User ASP4 in Partition 3 (LPAR3)	41
2-15	Cluster Lab hardware overview	42
2-16	Malmo hardware resources	43
2-17	Atlanta hardware resources.	44
2-18	Malmo primary partition	45
2-19	Secondary logical partition CL2 on MALMO(A)	46
2-20	Atlanta(B) primary partition	47
2-21	Secondary logical partition RCHASCL1 on System B	48
2-22	Secondary logical partition RCHASCL3 on System B	49
2-23	ITSO iSeries configuration.	51
2-24	Application overview	52
3-1	Enabling Edit Recovery for Access Paths (EDTRCYAP).	59
3-2	Symmetric Multiprocessing (SMP).	60
3-3	Disabling Edit Recovery for Access Paths (EDTRCYAP)	61
3-4	System ASP configuration	62
3-5	RAID-5 protected user ASP configuration	63
3-6	Configuration of user ASP mirrored at the disk level	64
3-7	User ASP with slower disk units	65
3-8	Elapsed runtime.	66
3-9	Runtime percentage variance	66
3-10	CPU consumption	68
3-11	Disk write statistics.	69
3-12	Logical database writes and I/O waits	70
3-13	Journal entries produced per second.	71
3-14	Average kilobytes per second written	72
3-15	Journal receivers sharing the same ASP.	73
3-16	Disk write cache overview	75
3-17	Write cache memory sizes.	76
3-18	No journaling with and without write cache	77
3-19	Journaling penalty on smaller system even with write cache	78

3-20	Journaling with no write cache.	79
4-1	Edit Recovery Access Paths (EDTRCYAP).	83
4-2	Remove internal entries (*RMVINTENT).	86
4-3	Changing journal receivers	88
4-4	Recovery at a hot site - APYJRNCHG.	89
4-5	Elapsed runtime	94
4-6	Runtime percentage variance	96
4-7	CPU consumption	97
4-8	Disk write statistics.	98
4-9	Database writes and I/O wait.	99
4-10	Total MB written.	100
4-11	Journal entries per second.	101
4-12	Total writes and I/O wait	102
4-13	Runtime summary	103
4-14	APYJRNCHG on a hot site	105
4-15	Throughput impact: MINENTDTA with BLOBs	106
4-16	Elapsed time to execute Focus Program (old version)	107
4-17	Elapsed time to execute Focus Program (new version)	108
5-1	MAX1TB access path size.	113
5-2	Pre-allocating space for your tables.	114
5-3	Concurrent Holey adds	117
5-4	Evolution of the Focus Program.	120
6-1	Remote journaling: asynchronous mode	126
6-2	Remote journaling: synchronous mode	127
6-3	HA BP data harvesting concept without remote journaling	131
6-4	Asynchronous remote journal for HA environment	132
6-5	Average sustained KB sent per minute by each communication infrastructure.	133
6-6	Source journal entries sent per minute without the PRPQ.	133
6-7	Number of journal entries generated per minute - PRPQ benefit	134
6-8	Synchronous RJ: send volume versus acknowledge volume	135
6-9	Number of transactions generated and sent per minute	136
6-10	Journal entries generated per minute	136
6-11	HA solution sending environment	138
6-12	Original non-remote journal HA BP solution without remote journaling.	138
6-13	Unsent journal entries backed up on the source system	139
6-14	HA BP versus remote journal conclusions.	140
6-15	Remote journal catch-up mode	142
6-16	Catch-up speeds and utilization.	142
6-17	How many disks per IOA	143
6-18	Catch-up speeds during catch-up mode	144
6-19	ESS arms on the target system	145

Tables

1-1	Suggested system resources.	22
1-2	Batch PRPQ benefits example 1	23
1-3	Batch PRPQ benefits example 2	23
2-1	Communication methods and protocols.	49
3-1	Disabling write cache.	77
4-1	The testing scenarios.	93
4-2	CPU overhead of journaling.	104
4-3	Focus Program test scenario.	106
5-1	Focus Program test scenarios.	119
5-2	The results	120
6-1	Benchmark test systems	129
6-2	Benchmark communications links	130
6-3	The test scenarios	130

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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



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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates 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. 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 IBM's application programming interfaces.

Trademarks

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

AS/400®	Enterprise Storage	Perform™	SP2®
DB2®	Server™	pSeries™	System/38™
DB2 Universal	IBM ®	Redbooks(logo)™ 	xSeries™
Database™	iSeries™	Sequent®	zSeries™
e (logo)® 	OS/400®	SP™	

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

Notes®

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Preface

Journaling is an inherent feature of OS/400 on the iSeries, a feature that has been around since the early releases of the System/38. The system employs journals to ensure data integrity and recoverability and this support can further be extended to reduce the duration of an abnormal system end. Journaling also serves as the cornerstone upon which many data replication and data warehouse refresh schemes are built.

The aim of this IBM Redbook is to provide you with an understanding of the factors which influence journal performance on the iSeries and to help you identify which hardware options, and software settings you can employ to minimize this impact. We will also share with you some ideas on how to minimize the impact of journaling on your iSeries from an application point of view.

The remote journal function on the OS/400 offers you a reliable and fast method to transfer journal entries to a remote iSeries server. In this redbook we have also conducted some tests concerning remote journal performance and tested this function in different environments.

If you want a big picture view of the information in this book, just read the interview in the first chapter and the conclusions in the last chapter. If you want the details and basis for these conclusions read the remaining chapters.

Throughout the redbook we will have some shaded boxes called *rules of thumb*. In these boxes we will describe the most important conclusions of the book. At the same time in Chapter 7, "Conclusion" on page 149 we summarize all the conclusions of our tests concerning journaling performance.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Rochester Center.

Hernando Bedoya is an IT Specialist at the International Technical Support Organization (ITSO) in Rochester, Minnesota. He writes extensively and teaches IBM classes worldwide in all areas of DB2 UDB for iSeries. Before joining the ITSO more than one year ago, he worked at IBM Colombia as an AS/400 IT Specialist doing presales support for the Andean countries. He has 16 years experience in the computing field and has taught database classes in Colombian universities. He holds a master's degree in Computer Science from EAFIT, Colombia with emphasis in database technology. His areas of expertise are database technology, application development, and data warehousing.

Larry Youngren serves as a Microcode Designer for the lower layers of the OS/400 operating system and the Leader of the iSeries Journal Development Team. Since joining IBM in 1976 he has worked exclusively with the microcode, first for the S/38 and now for the iSeries. He has led development teams responsible for database, journal, commit, checksum, Save_While_Active, and SMAPP. His current assignment involves future performance and recovery improvements affecting journaling. He has 26 years of experience with database and recovery software produced in Rochester, Minnesota. He holds a master's degree in Computer Science with an emphasis in operating systems.

Monti Abrahams is an iSeries IT Specialist and a Certified SAP R/3 Basis Consultant at IBM in South Africa. He has 10 years of experience in AS/400 systems development, two years of experience in configuring and implementing SAP R/3 production planning (PP-module) and five years SAP R/3 basis experience on AS/400. Monti currently supports SAP R/3 iSeries installations in South Africa and Namibia and also provides general iSeries technical support.

Yuhko Asaeda is an IT Specialist at the IBM Global Services Japan East Solutions Company in Tokyo. She supports customers especially those from the financial industry, such as Japanese Major Bank and Life Insurance Company directly as an AS/400 IT Specialist. She has 12 years experience in the computing field. Her areas of expertise are AS/400 system infrastructure, and disaster system recovery using Lakeview Technologies solutions.

Jon Mathews is a student at Iowa State University. He is working towards a bachelor's degree in Computer Engineering with an emphasis on networks and security. Currently, he is on a second cooperative education assignment at the IBM Laboratory in Rochester, Minnesota, where he has worked with the Journal Development Team for almost a year.

Adam Stallman is a member of the SLIC Journal Development Team in Rochester, Minnesota. He has been working with the Journal Team to provide testing of journal functionality and to participate in the development of new journal features. Adam recently graduated from Iowa State University with a bachelor's degree in Computer Science. While attending school, he was able to gain experience and knowledge of a wide variety of software development practices by participating in internships with three different companies.

Valentin Sueiro is the technical support manager at the Banesco Universal Bank in Caracas, Venezuela. Before joining Banesco more than a year ago, he worked as an AS/400 Technical Consultant in Caracas, Venezuela. He has been involved in two extensive benchmarks at the IBM Rochester Benchmark Center in the last two years, stressing journaling performance and addressing application issues. He has 14 years experience in the computing field. He holds a degree in Computer Science from Universitarie College of Caracas.

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

Special thanks to the Teraplex Center Team in IBM Rochester:

Michael Cain
Tom Moskalik

Special thanks to the Cluster Lab Team in IBM Rochester:

Ron Peterson

Special thanks to the following Consultants that helped us with their vision and expertise:

Paula Stewart from Lakeview Technology
Jason Leisch from Lakeview Technology
Rick Turner consultant on Performance topics

In addition, we thank the following people, who are experts in their respective fields, and their employers for allowing them to give of their time to this project. Without them, the quantity and depth of the experiments in this book would not have been possible:

Charly Jones, Research Scientist of Vision Solutions, who configured and enabled a vast array of experiments using a rich variety of different communications gear. He produced the software used to kick off these tests, automate them, and analyze and reduce the resulting data collected. Charly's participation was critical and essential.

Craig Johnson, Manager of Product Development of Lakeview Technology Inc., who made it possible to obtain results for a wide variety of tests to assist customers in making performance decisions. He shared his insight and experience regarding real-world performance factors, and helped us pull out and analyze the measured results across a wide variety of scenarios. The resulting redbook gives a much broader view of factors affecting typical HA customers as a result of Craig's insights.

Thanks to the following people for their invaluable contributions to this project from IBM Rochester:

Dale Weber
Raymond Morse
Don Zimmerman
Bob Gintowt
Jim Bonalumi
Peg Levering
Brian Jongekryg
Gottfried Schimunek
Bruce Hansel
Bruce Jawer
Bob Galbraith
Kent Milligan
Jarek Miszczyk
Sue Baker
Dave Owen
Lou Antonioli

Special notice

This publication is intended to help database administrators, application designers, and system programmers to understand the best performance choices concerning journaling. The information in this publication is not intended as the specification of any programming interfaces that are provided by OS/400. See the PUBLICATIONS section of the IBM Programming Announcement for OS/400 for more information about what publications are considered to be product documentation.

Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review redbook form found at:
ibm.com/redbooks
- ▶ Send your comments in an Internet note to:
redbook@us.ibm.com
- ▶ Mail your comments to the address on page ii.



Part 1

Background

In this part we:

- ▶ Discuss why we are writing a book on journaling performance
- ▶ Describe the most common questions most customers have concerning journaling
- ▶ Describe the new V5R1 journaling enhancements
- ▶ Describe the testing environment, hardware, software and applications that we used for this project



An introduction to journaling performance

Journaling is an inherent feature of OS/400 on the iSeries, a feature that has been around since the early releases of the System/38. The system employs journals to ensure data integrity and recoverability and this support can further be extended to reduce the duration of an abnormal system end. Journaling also serves as the cornerstone upon which many data replication and data warehouse refresh schemes are built.

This support afforded by journaling can also provide a mechanism to minimize the possibility of the loss of critical enterprise data. You can achieve this by journaling not merely your user database tables but also other supported objects, such as those housing critical data including data areas, data queues, or contents of the IFS file system. At the same time, there is a constant demand for faster processing speeds, as well as a continual shortening of the available time frames to process massive amounts of data.

Journaling has an impact on system performance.

Many customers are in the process of implementing a high availability solution. This requires the use of journaling. These customers want to know:

- ▶ How will journaling affect their batch runs?
- ▶ How can they reduce the impact of journaling on their systems?
- ▶ What options are available to reduce the impact of journaling on performance of both their interactive jobs as well as nightly batch runs?

These are only some of the questions our customers ask and we hope to answer these and most of their other performance related questions in this redbook.

This chapter describes the most common performance-related questions that most customers have concerning journaling. It also provides an introduction to the journal environment and it illustrates the enhancements present in journaling for V5R1.

1.1 A customer call to a journaling expert: Interview

To better understand the concerns that many customers have in regard to journaling, we invited Banesco, an actual iSeries customer, to be part of this project. Banesco is one of the leading commercial banks in Venezuela. Banesco has been an AS/400 and iSeries customer for many years. In the last three years they have grown by more than 400% due to a large merger.

To ensure ultra high availability for their banking customers, journaling has become a critical requirement for their environment and, therefore, we invited Banesco's IT technical support manager to join us for this project. To illustrate some of the most common questions that our customers pose concerning journaling, we conducted an interview with Banesco's IT technical support manager and one of the leading technical experts in this field. This interview consisted of:

- ▶ **Valentin Sueiro:** Technical Support Manager at the Banesco Universal Bank (Banesco) in Caracas, Venezuela, an iSeries and AS/400 customer. This interview reflects his company's needs for journaling.
- ▶ **Larry Youngren:** Leader of the IBM Journaling Development Team for IBM Rochester that designs and implements new journal function at the microcode level of the iSeries operating system. His group also advises those in IBM's Global Services organization who consult with customers and IBM's business partners regarding journal performance and tuning issues.
- ▶ **Hernando Bedoya:** IT Specialist for the IBM ITSO in Rochester, MN, and project leader for this redbook. He served as the moderator of this interview.

And so our interview begins...

Hernando: Good day, Valentin. First of all, thank you for accepting our invitation to be part of this project. We believe that it is very important to have a customer participate in this project. This gives us the opportunity to investigate and measure performance issues using a real customer application, in this case your banking application. Let me introduce Larry Youngren from the IBM Journaling Development Team.

Valentin: I'm pleased to meet you, Larry. Thank you for inviting me to be part of this interesting redbook. This project is very important to us because it will help us investigate and understand many of the questions that we have regarding journaling performance.

As you know, we are a large commercial bank in Venezuela and we have been using AS/400 and iSeries for many years. Back some years ago, we operated our shop without the use of journaling and while we were relatively comfortable at the time with our backup and recovery strategy, we now realize that we weren't achieving the highest possible protection. Later, we implemented a true high availability solution but, due to some performance concerns, started journaling only our most critical tables.

Today even that is not enough. We now realize that we need to journal all of our tables to be able to fully implement our high availability solution. My batch window is becoming increasingly shorter as we are now offering Internet services as well and I cannot afford to substantially increase my batch window to accommodate journaling. My first question therefore is:

If we decide to start journaling all our database tables, how is my batch processing going to be affected? How much longer will it take?

Larry:

The journal support on the iSeries employs a software protocol known as *write-ahead journaling*. This means that each time you make a change to a journaled database table, a corresponding journal entry is produced and written to disk.

Writing to disk incurs noticeable and measurable overhead. Your challenge in a situation like this is to minimize that overhead. The good news is that there are both hardware and software assists that can substantially reduce this overhead. Our efforts during the production of this redbook were aimed at helping to identify those approaches that provide the largest performance benefit and to quantify that benefit for a particular application.

The bottom line is that it's possible, with sufficient software tuning matched with appropriate hardware, to limit this journal performance overhead to no more than 3% to 5% of the original elapsed time for your batch job.

The two most helpful approaches in most shops are:

1. Ensure you have sufficient write cache in your I/O adapters servicing your disks.
2. Install and enable the Batch Journal Caching PRPQ, 5799-BJC.

Valentin:

For our total high availability strategy, journaling merely database tables and access paths is not enough. We have critical data residing elsewhere.

Are there any plans to journal other types of objects, such as IFS objects?

Larry:

Yes indeed! In fact, V5R1 of OS/400 already provides the support to allow you to begin to capitalize on journaling protection for stream files within IFS directories, data areas, and data queues. Many of the same tuning principles that help speed up database journaling similarly are helpful in reducing the overhead of journaling non-database objects. Once you can journal such objects, it follows that you can replicate them to a second iSeries server in real time by utilizing a High Availability Business Partner's product that recognizes these new journal entry flavors.

Some folks have asked whether there's any need to separate the non-database journaled objects from traditional database objects and set up two separate journals. The answer is a resounding "*No!*". A single journal can easily service a mixture of object types. In addition, once journaled, objects such as data queues are far less likely to end up flagged as damaged objects in the event of a machine crash.

Therefore, not only can you journal objects well beyond the bounds of database, but you certainly should – especially if these non-database objects house critical data that is essential for the continued functioning of your business. In fact, that's probably one of the ways in which the iSeries server distinguishes itself from most other platforms; journaling on our platform isn't only for databases any more.

Valentin:

I understand that journaling has an impact on my batch runs because there could be a huge amount of journaling activity that has to be written to disk.

Are there any hardware configuration options that I could explore to reduce the disk impact of journaling?

Larry:

Faster spinning disks, private user ASPs to house those disks, and your choice of disk protection mechanism (RAID versus mirroring) all have an impact on your resulting journal overhead because they each affect the efficiency with which disk writes can be performed and journaling is an especially aggressive writer of disk sectors. The efforts of our project, that led to this redbook, attempted to investigate and calibrate each of these kinds of choices and the overwhelming result was that the quantity of write cache in the I/O adapters stood out as the most significant hardware choice by far.

If you make no other hardware choice to improve journal performance, make sure you've configured sufficient IOA write cache. In some instances, this may mean restricting yourself to only 10 disk drives per IOA, even though you are allowed to physically attach up to 15. In effect, such discipline on your part helps to assure that each disk drive is serviced by a larger quantity of write cache and that's a wise choice.

Given sufficient write cache in the I/O adapters, the influence of the other hardware factors remains largely hidden and almost inconsequential until your application overrides the capacity of the write cache, as might happen during an especially aggressive burst of database activity such as a CPYF operation or a remote journal catch-up phase. During these brief bursts, even a substantial write cache can feel overwhelmed.

If you tend to have a lot of these disk write intensive operations, you'll want to look beyond merely sizing your write cache properly and will also want to consider installing disk drives that spin faster as well as a disk protection scheme that inherently produces less total disk traffic.

In some of our earlier tests, prior to the residency, we worked with some customers who were able to see up to a 30% performance benefit during such periods of intense disk write activity by employing the faster 10K rpm disk drives rather than the slower 7.2K rpm drives. Others have found performance relief by selecting the disk drives configured into their user ASP wisely – assuring that each of the drives selected are serviced by a different IOA (rather than having all disk arms in the user ASP attached to the same IOA) and thereby maximizing the quantity of write cache at the disposal of the journal.

Valentin:

The current IOPs provide write cache in a variety of sizes. I could choose 4 MB or 26 MB.

How do I know what size of write cache is appropriate for me?

Larry:

In our past experience when working with other customers, we generally discovered that 4 MB of write cache is sufficient only if your journaling traffic is modest. If you elect to use the older vintage IOPs whose IOAs have only 4 MB of write cache, you'll probably similarly want to limit yourself to asking each IOA to service no more than 4 disk actuators. This is especially important if you tend to fill 2GB of journal receivers in less than an hour.

With the newer vintage 26 MB or even 104 MB write cache IOAs, we've yet to see an actual customer environment in which the write cache was unable to adequately service the journal traffic, provided you have configured enough disks in the user ASP that is housing your journal receiver. In our

particular tests conducted during this redbook project, 26 MB was almost always sufficient until we attempted so-called *remote journal catch-up processing* in which extremely high volumes of journal entries flow between a source iSeries and a target iSeries machine at a torrid pace. In that particular case, we needed even more IOA write cache configured in order to achieve optimal performance.

As machines become increasingly faster and workloads continue to climb in the future, the day may come when even 26 MB may not be enough to handle truly aggressive journal traffic. But for most applications, we don't seem to be there yet. I'm glad that the hardware designers are ahead of the curve on this one and have already introduced the 104 MB write cache offering so that we'll be able to handle even higher journal volumes in the future.

Valentin: We have been advised in the past to use user auxiliary storage pools to house our journal receivers.

Are there any specific guidelines I should follow concerning User Auxiliary Storage Pools?

Larry: The disk protection technique you employ for your user ASP can make a performance difference. The number of disk actuators configured within the user ASP also can influence performance. The underlying journaling support in the operating system attempts to spread the journal traffic rather evenly across all the disk units present in the user ASP. The more such units there are, the more parallelism is possible and hence the wider the effective bandwidth. These factors are especially important on machines that have little if any write cache in the IOAs – as might be true on some of the smaller and older models of iSeries machines.

Configuring sufficient write cache may be sufficient on its own; however, in many instances to mitigate the extra benefits of your disk protection choice and quantity of disk units configured, our tests suggest that RAID protection is generally faster than mirrored disk protection (provided you aren't overrunning the write cache of the IOA). And, similarly that while there were a few instances in which a user ASP housing only one singular disk unit was insufficient to handle the journal volume, two disk units in such an ASP were nearly always sufficient even in the largest configurations we tested.

If you are comfortable configuring sufficient disk arms and IOA write cache to service the journal via a user ASP approach, fine. It's probably the best choice. However, if you're reluctant to set aside that many disk arms for use exclusively by the journal, it may be acceptable instead to allow the journal receiver to reside within your system ASP in which case you should definitely do two things:

1. Enable the RCVSIZOPT(*MAXOPT1) feature for your journal, thereby allowing it to spread itself across up to 100 disk arms (the default without this setting is to limit itself to 10 arms).
2. Choose a sufficiently high journal threshold if you can afford the space. The higher the threshold, the more disk arms we write to in parallel and you want as much parallelism as possible if your journal volume is high. Hence, if you've been using a journal threshold smaller than 2GB, the time has probably come to re-examine your choice. The algorithm employed to decide how many disk arms to spread the journal receiver across is roughly:

Arms = Journal_Threshold_setting / 64 Meg

For any journal receivers placed in the system ASP, rather than a user ASP, you'll want to maximize the number of disk arms across which your journal receiver is spread to help make up for the fact that any journal writes to the system ASP must compete for use of these shared disk arms. Hence, you'll want to select a sufficiently high threshold to achieve this result.

Valentin: **The hardware options sound interesting, but what if I am a customer with budget constraints? Are there any options that do not require hardware or software purchases?**

Larry: My hunch is that your testing will reveal that the two most consistently helpful factors will be:

- ▶ Sufficient write cache in the IOA
- ▶ Use of the Journal Caching PRPQ

While both of these cost money, there are some additional no-cost options that have helped some customers to achieve improved journal performance. One of the most attractive is the V5R1 journal parameter MINENTDTA on the CHGJRN command. This setting informs the journal that it does not need to capture all the bytes of every database row that gets updated. Instead, it can simply capture only the changed bytes within each updated row.

This can be particularly attractive if you have applications which generally modify only a small number of columns within each row. Imagine, for example, a banking application that only changes your lingering balance within a master record and obviously doesn't change your account number or name or mailing address. Such an application would seem to be a particularly good fit for this software option. So too would be a real estate application that keeps track of, say, the asking price, address, and a photo of a house you had for sale. The address and photo are probably not likely to change but the asking price may get modified. When that occurs, it can be a big space savings and sometimes a nice CPU savings as well to record in the journal only the changed bytes. This is particularly true if the photo is represented as a large database BLOB field. Some customers we've worked with report up to a 30% size savings from this option alone.

One of the other popular journal tuning parameters you might want to consider is the MINFIXLEN setting on the CHGJRN command. Use of this option instructs the journal that he can save you both CPU overhead and space by reducing the quantity of audit information he's obliged to collect and record associated with each journal entry that gets produced.

Determining the program name, job name, and user profile associated with each new journal entry produced can burn CPU cycles. Using this option saves those cycles. According to some customers who have tried this option, the savings here are probably in the 5% CPU range.

Valentin: **Should we manage the journal receivers ourselves or leave it up to the system? Is there any benefit if the system manages the journal receivers?**

Larry: There used to be a very definite performance advantage by allowing the operating system to manage your journal receivers for you and you'd do so by specifying MNGRCV(*SYSTEM) on the CHGJRN command. In recent releases, we've removed the former substantial performance penalty which a user-managed CHGJRN incurred.

The former approach slowed down the CHGJRN request as the operating system purged main memory of all changed pages that were housing database row images associated with journaled tables. In fact, the CHGJRN waited until this massive purge completed.

That's no longer the case. Instead, this purge is delayed until the corresponding journal receiver is deleted. The hope is that by the time you delete your journal receiver, the matching database rows will have aged out of main memory naturally and gradually. Having said that, however, while it probably isn't a noticeable performance benefit any more to allow the system to manage the change of journal receivers for you, it may still be a wise choice on other grounds. In particular, it takes a burdensome housekeeping chore and turns it over to the operating system who never slumbers or calls in sick. Just be sure to set your journal threshold appropriately so that the operating system has some breathing room and sufficient time to react once you cross this threshold.

Valentin:

How often should I change journal receivers?

Larry:

That, I suppose, depends on what you're trying to accomplish. For some folks, especially those not employing a second iSeries server as a high availability site, but concerned about being able to recover as much of their data as possible from the journal, the answer is probably "as often as practical". Their objective is to safely save off the journal receiver contents to tape frequently so that in the event of an extended machine outage they can gather up their tapes housing the journal receivers and move to a hot site with as little loss as possible.

For others, especially those employing the remote journal support, the objective is not so much to rapidly detach a former journal receiver and save its contents to tape but rather to minimize the performance impact inherent in the very act of changing journal receivers. For them, the marching orders are probably "no more often than necessary".

Both needs can be addressed by not only employing the system managed support of journaling but also by selecting both your journal threshold wisely and by giving thought to your maximum journal receiver capacity. The maximum journal receiver capacity is influenced by the setting of the RCVSIZOPT parameter on the CHGJRN command. The default cap on maximum journal receiver size is approximately 2 GB. You can increase that cap by requesting RCVSIZOPT(*MAXOPT1), which has the effect of raising the upper size limit to approximately 1 Terabyte. Increasingly we're seeing more and more customers make this change, especially for shops in which the journal receiver grows by more than 2 GB per hour.

Valentin:

My journal receivers seem to grow rapidly in size.

Are there any options that will help me reduce the rate of growth of the journal receivers?

Larry:

Yes indeed! Some, like the STRJRNPF parameter for suppressing journal entries that identify each time a database file is Opened or Closed; OMTJRNE(*OPNCLO) have been around for years. They are good choices, especially if you have inherited applications that tend to open and close the same file frequently. Others, like the opportunity to capture selectively only the changed bytes of an updated database record via the MINENTDTA attribute on the CHGJRN command, are new with V5R1.

Anything you can do to reduce the number of bytes flowing into your journal receiver and ultimately flowing from main memory to the disk surface is a step in the right direction. This is especially true if you happen to also be sending these same bytes across a communication path to a distant remote journal on another iSeries server. Depending on your particular application's needs, you may also be able to sacrifice the collection of so-called journal "before" images.

Valentin:

Are there any additional journaling parameters that we could explore?

Larry:

We've already mentioned many of my favorites, but let me suggest a few more:

- *Consider the journal settings parameter:*

`RCVSIPOPT(*RMVINTENT)`

This, too, is found on the CHGJRN command. Shops that use SQL tables, have lots of access paths, or that elect to specify the REUSEDLT attribute for their database tables, ought to seriously consider this. This setting advises the underlying journal microcode that it has your permission to store the purely internal operating-system induced hidden journal entries on a disk surface separate from the ordinary journal entries that are explicitly generated by actions you've coded into your applications (like updating database rows). This is known as *removing internal journal entries*. I suppose it should more properly be called *separately writing internal journal entries*.

These so-called internal journal entries are produced automatically by the operating system when it senses that some journaled object has incurred some critical structural change that needs to be properly replayed at IPL time in the event that your machine happens to go down abnormally without successfully saving the contents of main memory. These entries help reduce the likelihood of object damage. They normally flow into your regular journal receiver, take up space, and are written to the same disk surfaces as the rest of your journal entries.

Such internal entries keep track of the number of deleted rows within your SQL tables and the location of each such deleted row. Others keep track of the internal structure of the binary tree housing your SQL Indexes and native Keyed Logical files.

Their presence is transitory in nature. They're needed only until the object they're intended to protect reaches a safe state on disk. As such, it seems a shame to tie up precious disk space by long-term, intermingling them among your ordinary journal entries; yet that's what happens unless you specify this journal attribute. Specifying the *RMVINTENT attribute seems especially helpful if your shop regularly saves journal receivers to tape and/or employs remote journaling support so as to route a duplicate copy of your journal entries to a second iSeries server. In both of these instances, using *RMVINTENT is probably an especially attractive choice since it reduces the number of journal entries that must be replicated. That's because, without this attribute the default behavior is to both write these extra journal entries to tape during save operations as well as to ship them down the communication wire to your backup machine when remote journaling is being used.

Yet, they're not used when the objects on the tape are restored; nor are they used by a backup machine in a High Availability environment. Therefore, both actions seem wasteful and you can eliminate these wasteful overheads by overriding the default setting.

- *Identify your largest and most critical Access Paths and elect to explicitly enable journaling support for them:* This is one of my other favorites. This reduces the amount of background work performed by the SMAPP (System Managed Access Path Protection) microcode tasks.

SMAPP is a facility running in the background on most machines whose mission in life is to help keep the duration of abnormal IPL processing modest in size. It accomplishes this by dynamically deciding which access paths to implicitly start journaling.

If you have large, popular, frequently modified tables or physical files, chances are that each of the access paths defined over them are repeatedly selected day after day, for this implicit journaling protection. Having the microcode tasks make this decision for you and then monitor their decision second by second costs you something.

If you open and close your files frequently, this overhead can be especially onerous. You can minimize this overhead by making the decision overtly yourself, once and for all, by identifying such access paths and then issuing the STRJRNAP command for each. Doing so helps reduce the cost of the background SLIC journaling tasks without sacrificing the shorter IPL durations we were seeking.

- *Use the API, QJOSPEED, better known as the Journal Batch Caching PRPQ:* I consider this the most powerful and successful journal tuning parameter of all. This is not a standard feature of the OS/400 operating system as shipped, but instead is available as a separately installable PRPQ (5799-BJC).

It's primarily intended for those environments, like batch jobs, where the rate of arrival of new journal entries is especially rapid and where the timeliness of writing the corresponding journal entries to disk is not quite as critical. It achieves impressive speed in such environments by caching the resulting journal entries in main memory just long enough to assemble an optimal quantity; then it writes them to disk in unison in a single disk revolution thereby generally achieving quite impressive performance gains.

Experiments using the journal batch caching PRPQ during this residency demonstrated time after time that this optional feature can substantially improve the performance of most journaling environments. In fact, it works so well that many shops elect to install one instance on their production machine and another instance on their target machine. That way neither the production machine nor the high availability target machine tend to fall behind. Both are able to sustain impressive journal performance even when faced with high rates of journal traffic. If you get nothing else from this redbook, resolve to take a good long hard look at this impressive PRPQ.

Valentin:

From an application point of view, are there any options that we could explore?

Larry:

I'm glad you asked that question. Many folks concentrate only on the hardware configuration choices or purely on the journal tuning parameters and fail to take into account the fact that they can often make simple changes in their applications that can lead to substantial differences in both journal and database performance. There are dozens of application changes one could entertain that lead to improved journaling and/or database performance in a journal-intensive environment. Among my favorites (because I've seen them lead to measurable and impressive performance gains in shops like yours) are:

- ▶ *Find and retrieve any lingering uses of FRCACCPH(*YES):*
Some years ago it was a popular practice to try to reduce the duration of abnormal IPL options by regularly writing new keys to disk as rapidly as they arrived in keyed logical files. Once the opportunity to journal files via STRJRNAP came along it made far more sense to abandon the FRCACCPH attribute for access paths. Surprisingly, I still find shops that haven't abandoned the older approach. They're giving up a lot of performance and would be wise to set FRCACCPH(*NO) or CHGLF.
- ▶ *Breaking your long-running batch jobs into multiple parallel threads:*
This is a clear winner and something that I know your shop in particular has already capitalized on and has seen an impressive return on your investment. If your application doesn't have a strict dependency that you update the first row in a particular table before you update the final row in this table, then it certainly makes sense to entertain the idea of breaking that phase of your long-running batch job into two parallel streams:
 - One updates all rows in the first half of the table.
 - The other updates all rows in the second half of the same table.

What had been one job during this phase becomes two jobs, and these jobs are run in parallel. Unless the original job was already consuming all the CPU cycles on your machine, the result is likely to be that the execution time for your run is cut nearly in half. Better yet, from a journaling perspective, because we have two players instead of only one trying to make journal deposits at the same time, we get the opportunity to achieve better journal bundling. This usually means that we actually issue fewer disk writes, all of which leads to more efficient journal usage.

But don't stop at only two such parallel threads. If you can break certain phases of the application into two parallel actions, you may be able to just as easily break it into four or eight or even sixteen such parallel jobs. In fact, some of the customers who've had the most success in utilizing this technique have as many as 32 such parallel threads all executing at the same time. The general rule-of-thumb seems to be: "Keep splitting your original job into more and more parallel threads until you've reached nearly 95% CPU saturation."

- ▶ *Use a simple database override command, which can be accomplished with OVRDBF along with SEQONLY(*YES) and NBRRCDs:* This is a traditional favorite approach that helps speed up both journaling and database processing. The secret here is to make sure the database files you select for this override have a *private* view that is used *only* to perform database Adds (not updates or deletions). You'll want to select a NBRRCDs value that maximizes use of your ODP buffer and matches it

against the underlying best of breed of your disk technology.

Here's what you need to know in order to accomplish that. The operating system limits itself to writing no more than 128k bytes to the disk on behalf of the database or journal per disk revolution. Knowing that, and the fact that each database row has an extra leading status byte added by the operating system, it's a simple calculation to set your optimal NBRRCDS value:

$$\text{Optimal NBRRCDS} = 128K / (\text{Database_Row_width} + 1)$$

Valentin:

Now that you mention parallel streams or parallel tasks we do have a nice example of this approach. In our batch end-of-day process, we had a program that used to take multiple hours and we decided to break it into several parallel tasks. In our tests during this project, we actually brought it down from 83 minutes elapsed time to 23 minutes.

We have heard about remote journaling.

How can this help us, since we have both a source and a target system?

Larry:

The fact that you have two iSeries servers, your primary and a separate failover target system, means that you can attempt to achieve high availability by sending your journal entries in real time from the production server to the backup server. Some folks do this by acquiring a High Availability Business Partner's software and letting them orchestrate the flow of journaled database record images from one machine to the next.

Some years ago, the only way to accomplish this was to read the journal entries on the production machine, place the images on the communication wire, catch them on the far end, place them into a holding object often called a "log file", and then eventually replay those database images to a replica of your database residing on the backup machine.

The act of harvesting the journal images on the production system puts some drain on the CPU of that system and also introduces some delay into the trip from production machine to backup machine. Both of those observations led us to provide a newer and perhaps better alternative which some folks have recently started using – *remote journaling*. Some of the tests we conducted during this residency were aimed at measuring the remote journaling performance characteristics. (There's also a former redbook, *AS/400 Remote Journaling Function for High Availability and Data Replication*, SG24-5189, which addresses this same topic in detail).

When you elect to enable the remote journaling attribute, the local journal on your production system ends up with an identical twin journal on the backup system. Entries that flow into the local journal, also end up in real time in the remote instance of this journal. This is known as *sync-flavored remote journal support*. It affords a new opportunity for even higher levels of availability since it removes the risk associated with the inherent latency of the prior approach.

Once you elect to employ remote journaling, the quantity of CPU cycles consumed on the source machine is reduced since the harvesting step no longer occurs on your production machine but instead is transferred to your backup machine. In addition, with remote journaling the transport

technology moves from actions performed at an application level down to the microcode level, which affords an opportunity for additional efficiency.

Most high availability business partners and software providers who are serious about performance have versions of their product available that incorporate this efficient remote journal transport technology.

Part of our efforts during preparation of this redbook were aimed at measuring the CPU overhead that can be off-loaded from the production machine to the backup machine by employing the remote journal technology instead of the older vintage approach. For the particular applications we measured, this harvesting overhead consumed about 7% of the available CPU capacity, so the use of remote journal technology is likely to reduce your production machine's CPU load by a corresponding amount.

Hernando: Gentlemen, allow me to interrupt. These are all very valid questions and really interesting answers. But, instead of simply talking about these issues, why don't we put Larry's advice to the test. We have the hardware in place, so let's install the Banesco applications and database and implement the concepts that Larry so eloquently explained. We can then measure and experience how the impact of journaling can be reduced.

Based on this interview and similar issues raised by other customers, we embarked on a series of tests to investigate how you can reduce the impact of journaling on the iSeries. Many of these tests were conducted using the actual Banesco database and application programs and were run on an iSeries with the same processing capabilities and hardware configuration as the Banesco production iSeries server. Thereafter, to help supplement our investigation, additional tests were conducted using an SQL environment that simulates a retail sporting goods store environment. This application is modeled on an actual retail store which operates in a city in Northern Minnesota, USA.

1.2 The journal environment: Background

The journal support on the iSeries creates a new journal entry each time you modify a journaled object. In the case of a database table, for example, adding of a new row or updating an existing row produces a new journal entry which contains a copy of the contents of the row, as well as some descriptive information about the transaction that caused the change. This new journal entry is then deposited in the journal receiver and written to disk in order to ensure survivability. In fact, journal entries are always written to disk before their corresponding database rows are allowed to leave main memory. With that much disk activity, you need to capitalize on every possible opportunity to make this process as efficient as possible.

There are several approaches which you can follow to help make this process more efficient. These include:

- ▶ Hardware selection and configuration choices
- ▶ Journal specific tuning parameters
- ▶ Application design techniques

The tests described in this redbook will attempt to measure the effectiveness of these various approaches. You will notice that certain techniques show more advantages than others. However, certain concepts that we have identified as having only a small impact on our banking application, might by contrast have a much greater impact on your application, therefore they all deserve consideration.

Figure 1-1 summarizes the concept of journaling on the iSeries.

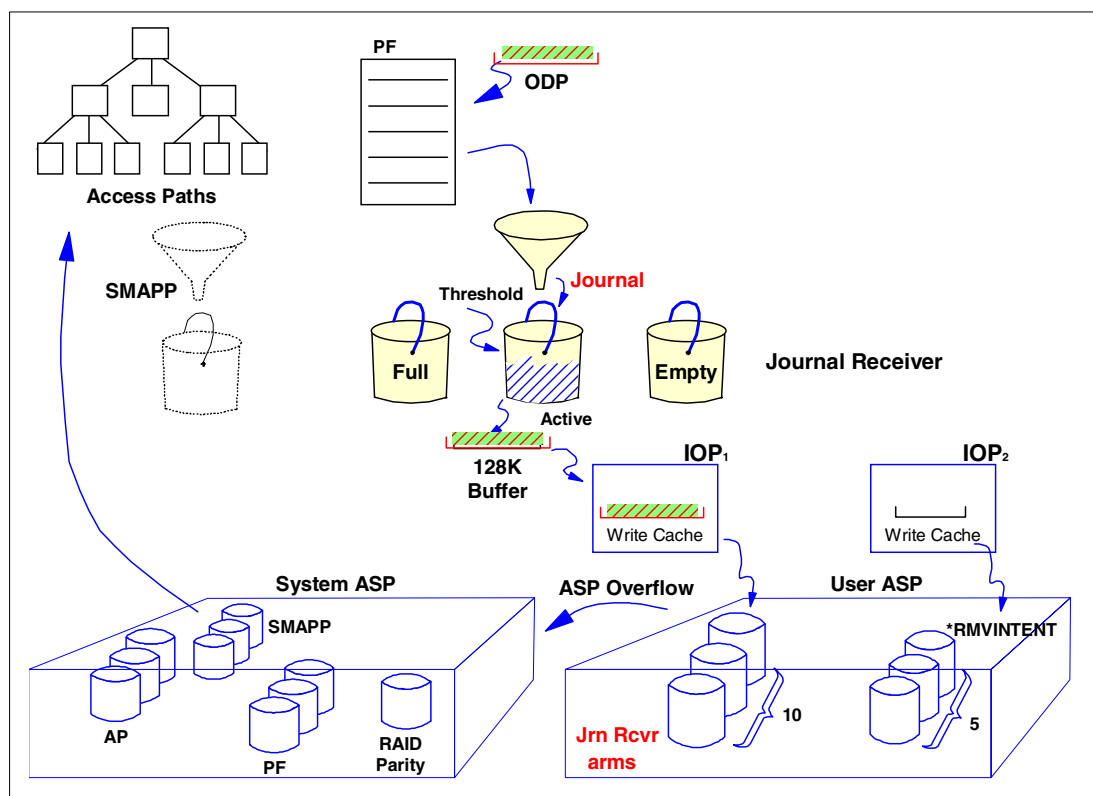


Figure 1-1 The journal environment

Let us trace the path of a journal entry as illustrated in Figure 1-1. Throughout the book we will refer to the concepts demonstrated by this chart.

Bundling

A journaled object is modified such as the physical file or table shown on Figure 1-2 and a corresponding journal entry is constructed in main memory. At the same time other user jobs modify other objects that are also journaled to the same journal. These journal entries are also placed in main memory. If the subsequent journal entries arrive rapidly, the journal manager generally keeps adding entries to the same area in main memory until the memory buffer is filled, usually as close to 128 kilobytes as possible. This process is referred to as bundling. The more journal entries you can string together into the same bundle, the fewer total trips to disk you must make and hence the more efficiently you are using your disk subsystem, main storage, IOP and IOA.

When the memory buffer is full, a storage management job is triggered. This job issues a single write operation which writes the contents of the entire memory buffer to disk. Therefore, bigger bundles means less overhead which generally translates into better performance. Figure 1-2 illustrates this concept.

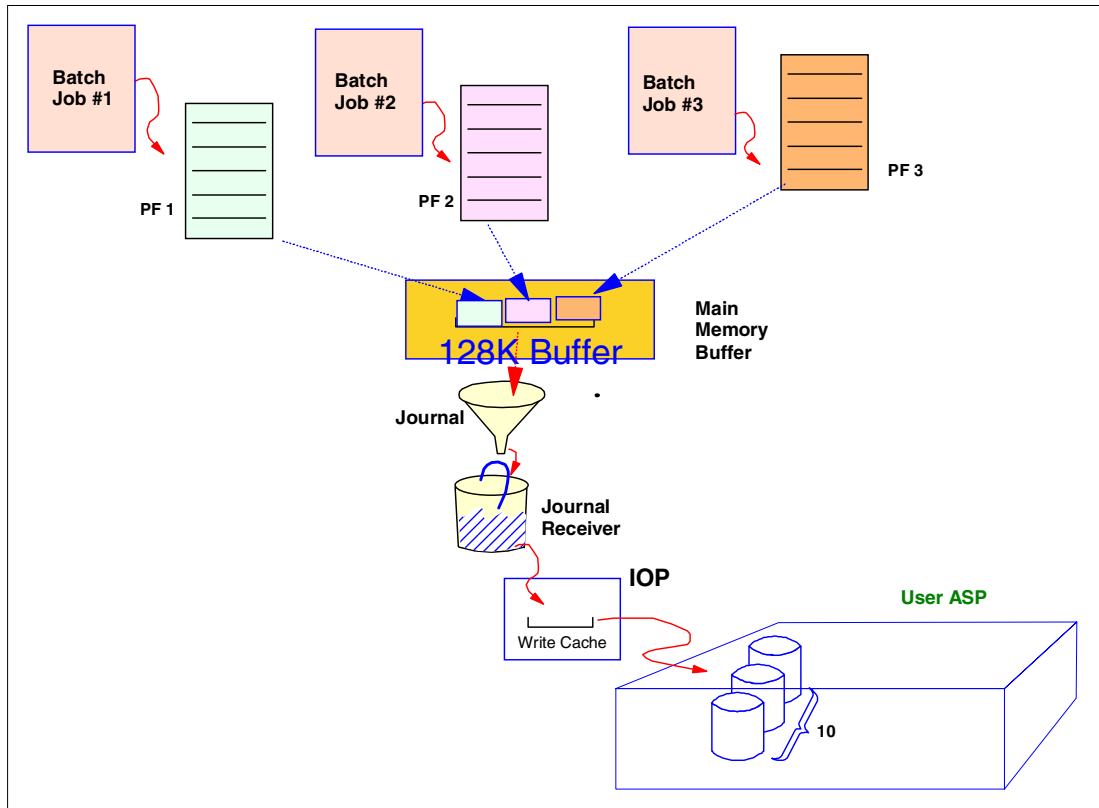


Figure 1-2 Journal bundling

Such bundling brings about improved journal performance.

Write cache

If your system has an I/O adapter (IOA) that supports write cache, the journal bundle is transferred from main memory to the write cache when the write operation is issued. Once the data is received into the write cache, it is considered successfully written. The IOA is equipped with its own independent power source that will ensure the data is written to the physical disk unit, even in the event of a power failure of the main system. Configuring your system such that most journal writes can be satisfied by the write cache of an IOA instead of waiting for a physical disk rotation is a clear performance winner since many IOA write cache transfers can be satisfied in 0.5 milliseconds or less while trips all the way out to the disk surface can take 10 to 20 times longer.

Coalescence of write cache contents

If the rate of filling the journal was rather timid, the journal data could remain in the write cache up to a few minutes. Most journal traffic however leaves the IOA's write cache rapidly. The IOA is equipped with a processor that monitors the flow of data into the write cache. This processor uses algorithms to determine how long to retain the data in the write cache, depending on the type and frequency of subsequent data flow into the write cache. The objective is to string together data that resides in the same area of the disk or to replace data already in the write cache with more up-to-date data. This process is referred to as coalescence. The advantage of this approach is that larger, but fewer, physical write operations occur against the disk unit.

Both the bundling which takes place in main memory and the coalescence which takes place in the IOA's write cache are helpful in terms of reducing the total number of physical disk revolutions needed to service your journal disk write traffic. Fortunately, there are a few things you can do with some software and some hardware, which can influence this bundling behavior and make it more efficient. Let's take a look at some of these options.

Hardware choices

On the hardware side, you can elect to purchase and configure sufficient write cache to service the level of disk write traffic being produced by your journal activity. Our experiments were aimed at attempting to determine how much write cache in the IOA is enough. This is important because write cache comes in multiple sizes. Some smaller machines have none while larger systems with modern IOPs often have an ample supply. Having enough to support your peak anticipated journal traffic is critically important if you are striving to achieve optimal journal performance. So critical, in fact, that systems with an insufficient amount can incur a journal bundle write overhead as high as 100 ms per bundle when the quantity of write cache is woefully insufficient while systems with a well tuned and properly configured set of IOA write cache can see journal overhead per bundle written as low as 0.5 milliseconds. That's a 200-fold difference.

Software choices: Journal parameters

There are several journal tuning parameters that you can employ in an attempt to reduce the impact on your I/O subsystem. Some of these parameters help reduce the number of bytes required to represent a journal entry. These are particularly useful when you are concerned about sending journal entries over a communication line to a backup server, as would be true if you employ a High Availability Business Partner software solution.

Other options will enable you to construct wider strings of bundled journal entries in main memory, thereby employing main memory as an efficient staging area. One such caching option is the Batch Journal Caching PRPQ (5799-BJC). This highly effective caching option is a separate operating system feature that has to be specifically ordered and installed on your iSeries. Our tests revealed that the Batch Journal Caching PRPQ provides major performance advantages. This operating system feature capitalizes on the concept of caching and exploits it even further. In brief, the Batch Journal Caching PRPQ reserves two caching buffers in main memory for each disk arm used for journal receivers. Therefore, the more disk arms you employ for journaling, the more cache you have at your disposal. This translates into much fewer physical disk I/O operations.

Each main memory journal caching buffer can expand up to a maximum size of 128K bytes. Thus the best journal performance is achieved when the average journal bundle size approaches 128K since it means you've topped out each buffer and thereby have used its full caching potential. The journal caching PRPQ helps you strive for this objective. It accomplishes this by not merely stringing together journal entries contributed by separate jobs (which is the tamer normal journal behavior) but rather by also stringing together subsequent journal entries from the same job. This is especially important when you execute long running batch jobs. By allowing the Journal Caching PRPQ to string together the consecutive journal entries produced by a series of back-to-back database row update operations all within the same job, the batch job executes in substantially less time because each separate row update no longer needs to wait for a separate disk write. Instead, the whole set of row updates are bundled together and incur the cost of only a single journal-related disk write.

Some of the graphs you will see later in this redbook will illustrate just how powerful this caching behavior can be.

1.3 What's new in V5R1

Before introducing the new journal enhancements in OS/400 V5R1, let us briefly look at the history of the journaling support.

1.3.1 A little history

Originally, journaling was designed to protect only your database tables and its contents. A journal entry was written to disk for each database add, update or delete operation. In addition, disks did not rotate as fast as they do these days and the I/O Processor did not have any write cache. Disk arm contention frequently occurred due to the inter-play of regular database operations and journal operations. Journal entries are deposited on disk through a synchronous write operation, while database information often is written asynchronously.

Over time the concept of an auxiliary storage pool (ASP) was introduced, in part, as a means to address the problem of disk arm contention. As databases grew in size, the need for better performing journals became evident.

These requirements were met by introducing faster disk units, IOAs with large write cache capability, the Batch Journal Caching PRPQ and significant tuning knobs for the journal function.

The following sections will discuss some of the newest journaling features and functions in detail.

1.3.2 Minimize the Entry-Specific Data parameter

A new parameter MINENTDTA (Minimize the Entry-Specific Data) on the *CRTJRN* command and *CHGJRN* command allows you to specify that only the changed bytes within updated database rows need to be recorded on the journal, resulting in smaller journal receiver sizes. The benefits of having smaller journal receivers are:

- ▶ Less frequent journal housekeeping
- ▶ Less bytes written to disk
- ▶ Less bytes sent via remote journal support to a second iSeries, which means less taxing use of the communication bandwidth
- ▶ More journal entries can be packed into the same bundle

Note: You cannot save or restore a journal that has the parameter MINENTDTA specified to any release prior to V5R1M0. Also, a journal that has the parameter MINENTDTA specified cannot be replicated to a remote journal that resides on a system that has a release prior to V5R1M0. This obviously makes sense since releases prior to V5R1, wouldn't be prepared to handle these minimized entries.

This feature is supported for both database tables and data areas. The possible options are:

- | | |
|----------------|--|
| *NONE | Journal entries for all journaled objects will be deposited in the journal with complete entry-specific data. None will be minimized |
| *FILE | Journaled database tables may have journaled entries deposited with minimized entry-specific data. |
| *DTAARA | Journaled data areas may have journal entries deposited with minimized entry-specific data. |

For more information on MINENTDTA, refer to *iSeries Backup and Recovery*, SC41-5304, or the following Web site:

<http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/books/c4153045.pdf>

1.3.3 Journaling for non-database objects

One of the many purposes of journal management is to enable you to recover the changes to an object that have occurred since the object was last saved. You use a journal to define what objects you want to protect with journal management. With the release of OS/400 V5R1, it's now possible to journal objects other than merely database tables and indexes. The presence of the new support for non-database objects opens up a whole new opportunity for creative recovery schemes, including critical application data which doesn't reside in a traditional database. The new objects eligible for journaling include:

- ▶ Data areas
- ▶ Data queues
- ▶ IFS objects
 - Stream Files(*STMF)
 - Directories(*DIR)
 - Symbolic Links(*SYMLNK)

Starting and ending journaling for non-database objects

- ▶ Start journaling

You can start journaling for non-database objects using the following commands:

- STRJRN (Start Journaling for objects found by path name such as IFS objects)
- STRJRNOBJ (Start Journaling for objects found by library name such as data areas and data queues), see Figure 1-3 and Figure 1-4.

Start Journal Object (STRJRNOBJ)

Type choices, press Enter.

Object	> TESTDTAARA	Name
Library	*LIBL	Name, *LIBL, *CURLIB
+ for more values		
	*LIBL	
Object type	> *DTAARA	*DTAARA, *DTAQ
Journal	> DTAARAJRN	Name
Library	*LIBL	Name, *LIBL, *CURLIB
Images	> *BOTH	*AFTER, *BOTH

Bottom

F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
 F24=More keys

Figure 1-3 Start journaling for data area objects

```

                                Start Journal (STRJRN)

Type choices, press Enter.

Objects:
  Name . . . . . > '/dir1/dir2'

  Include or omit . . . . . *INCLUDE      *INCLUDE, *OMIT
                        + for more values
  File identifier . . . . .
                        + for more values
  Journal . . . . . > '/QSYS.LIB/TESTLIB.LIB/IFSJRN.JRN'

  Directory subtree . . . . . > *ALL          *ALL, *NONE


More...
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 1-4 Start journaling for IFS objects

► End journaling

You can stop journaling for non-database objects using following commands:

- ENDJRN (End Journaling for IFS objects)
- ENDJRNOBJ (End Journaling for Data Areas and Data Queues)

Considerations

There are several factors to consider if you plan to use non-database journaling. Some of these choices can improve journaling performance.

► IFS related:

- IFS object journaling can support various kind of objects such as *STMF, *DIR, *SYMLNK.
 - If you want to replicate new objects created after starting journaling, you can specify *Yes in INHERIT parameter in STRJRN command.
 - When you start journaling path name qualified objects such as stream files with the INHERIT parameter, then all objects created hereafter into that directory will automatically be journaled to the same journal.
 - If you specify SUBTREE(*ALL) on STRJRN when you start journaling, then subtrees and all objects therein are included.
- You can apply recent changes for these objects by using the APYJRNCHG command just like you've been able to do for years on behalf of database objects.
- You can omit journal entries representing open operations, close operations, and force operations against stream files by specifying *OPNCLOSYN in the OMTJRNE parameter of the STRJRN command. By doing this, you can reduce the number of journal entries in your journal receiver. However, it's only valid for *DIR and *STMF objects. (It obviously doesn't make sense for symbolic links, because they are only aliases.)

- ▶ All journaled objects, including data queue objects, must reside in the same ASP as the journal. (This helps assure that the linkage between the journal and the journaled object remains in tact.)
- ▶ If you elect to journal data areas, all modifications accomplished via CHGDTAARA command will show up in the journal.
- ▶ When you journal data queues, ONLY after images will be captured.
- ▶ IPL operations will recover all journaled objects from the journal, including data queues and data areas.

1.3.4 Journal PRPQs (5799-BJC, 5799-AJC)

Two new PRPQs provide an opportunity for you to achieve a significant performance improvement and easier hot site recovery. The Batch Journal Caching PRPQ(5799-BJC) is aimed at speeding up batch jobs and it helps High Availability solution providers to improve their processing speed when replaying replicated changes on a backup system. The other PRPQ (5799-AJC), available for use with V5R1, provides a new command, APYJRNCHGX, that will successfully apply or replay journal entries housing, even the DDL (Data Definition Language) operations at a hot site.

Batch Journal Caching PRPQ (5799-BJC)

The Batch Journal Caching PRPQ (5799-BJC) is especially helpful for shops who tend to execute lots of long-running overnight batch jobs which update database tables. Without this PRPQ it can often require a substantial amount of manual tuning in order to achieve optimal journal performance. With the PRPQ enabled, batch jobs in a High Availability environment often see their elapsed execution times improve significantly. The same PRPQ which speeds up batch job execution on the production machine can also be of significant value on the backup/target machine in a HA environment because it assists the HA BP software if installed executing on this backup machine in replaying the replicated database changes more rapidly, especially in those instances in which the replicated database files are journaled. In some instances, the replay time has improved up to four-fold.

This PRPQ can provide substantial improvement to applications which perform a large number of database add, update or delete operations against journaled tables. Without this PRPQ, batch jobs wait for each new journal entry to be individually written to disk. With the PRPQ, far fewer waits ensue, since multiple journal entries are combined into a single disk write. Thus, it can greatly reduce the impact of journaling on batch run time by reducing the delay in waiting for each journal entry to be written to disk. It should be noted, however, that applications using commitment control will likely see less improvement, because the commitment control support already performs some similar journal caching.

The following description and Figure 1-5 is an explanation of how this PRPQ works:

1. When you turn the PRPQ on, it allocates caching buffers in main memory to store your journal entries.
2. Subsequent journal entries remain in these buffers until an optimal size is achieved (128K). Also, the corresponding database rows similarly remain resident in main memory until the journal buffer is written to disk. The PRPQ buffers database operations including adds, updates, and deletes.
3. When your application issues a commit or does a CLOSE operation to the tables or when the 128K buffer becomes full, the PRPQ flushes the buffers and sends its contents to the write cache on the IOA. After that, database rows are written to the disk storage.

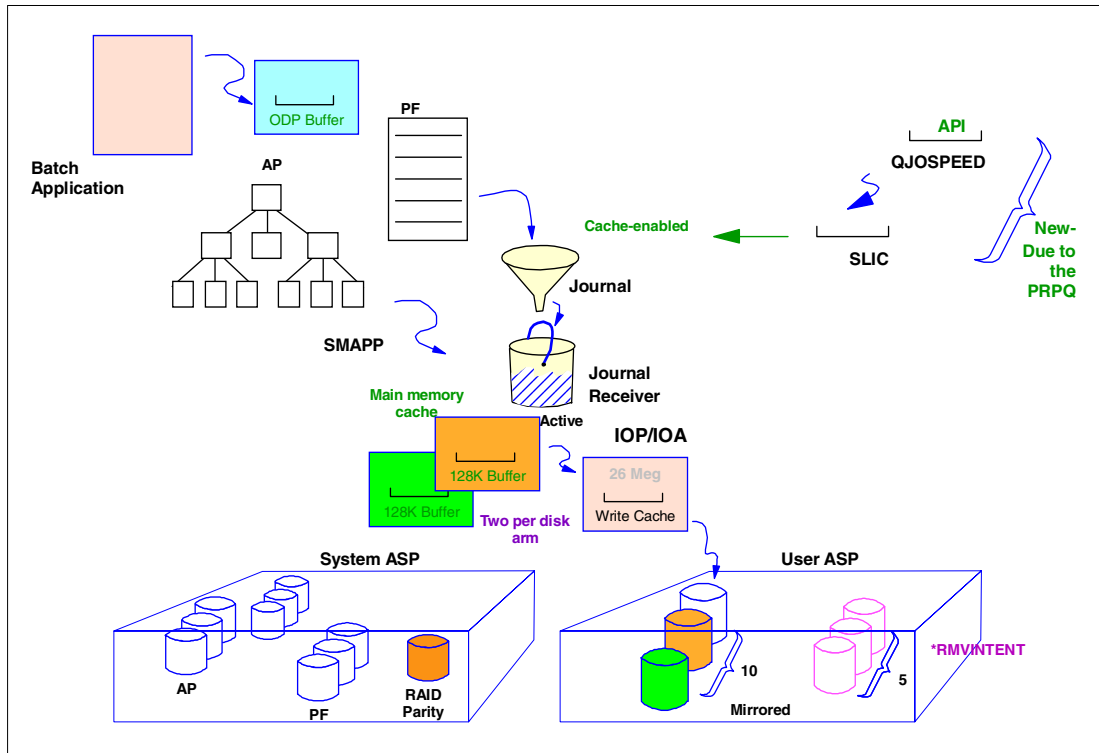


Figure 1-5 Journal Caching PRPQ

How to start caching

After installing the PRPQ, Batch Journal Caching must be turned on for each journal by issuing a call to QJOSPEED.

CALL QBJC/QJOSPEED PARM('Journal_Name' '*ON')

- Journal_Name is a 10 character field
- Second parameter can be:
 - *ON
 - *OFF
 - *DSP
- Optional third parameter:
 - Library Name

Considerations and recommendations

- If you use High Availability Business Partner Solution software and would like to help their apply/replay jobs keep from falling behind on the target machine when you've got a database intensive batch job running on the source machine, consider installing this PRPQ on the target machine.
- The quantity of system resources such as main memory and disk arms you will need depends on your journal rate. Please refer to the following table.

Table 1-1 Suggested system resources

Journal arrival rate	Extra main memory needed to house caching buffers	Minimum number of disk arms needed in the user ASP
200 MB/Min	1 MB	2
600 MB/Min	2 MB	4

Journal arrival rate	Extra main memory needed to house caching buffers	Minimum number of disk arms needed in the user ASP
1 Gig/Min	3 MB	6
1.8Gig/Min	5 MB	10

Note: Is the batch journal caching PRPQ appropriate for every environment? No, probably not. While it's an extremely effective performance enhancer for high volume batch environments, it is less effective in low volume interactive environments. In fact, some shops deliberately toggle it ON during nightly batch runs and OFF again for the interactive portion of their day. Since the PRPQ depends upon use of a main memory cache, and since this main memory cache doesn't get written to disk until it becomes full (128KB), low volume interactive environments might not only glean very little performance benefit from use of the PRPQ but also should consider the consequences of having both their journal entries and matching database changes memory-resident for a longer period of time.

Performance benefits of the 5799-BJC PRPQ

Later in this book you will see some graphs that illustrate the benefits of how powerful this caching behavior can be on the Banking environment that we tested. In the meantime the following are some tests results that have been reported by customers who have benchmarked this PRPQ.

The first test was done on a batch job that performs 5 million database operations (10% inserts and 90% updates). The batch job was provided by a customer who enlisted the assistance of the journaling team in speeding up their overnight batch runs. This job generates 9 million journal entries throughout the run. The batch job used to take 9773 seconds and with the PRPQ it was taken down to 1433 seconds. Table 1-2 illustrates the benefits this customer achieved using the PRPQ.

Table 1-2 Batch PRPQ benefits example 1

	Elapsed time	% Difference
Ordinary Journaling Enabled	9773 sec	
Using the 5799-BJC PRPQ	1433 sec	85% less elapsed time

While the Journal Caching PRPQ is an effective tool in many scenarios on a production system, the same journal caching behavior can be mighty helpful in most instances on a target system in an HA 24x7 environment. During the second test the journaling team worked with a customer who benchmarked the benefits of the journal caching PRPQ on a target system where HA BP software was used to replay the journal entries. The measurement used in this case was the number of transactions/hour that are being replayed on the target system.

Table 1-3 Batch PRPQ benefits example 2

	Database replay rate	% Difference
Ordinary journaling	600,000 transactions/hour	
Using the 5799-BJC PRPQ	2,400,000 transactions/hour	400% better

Rule of thumb: From Table 1-3 we can conclude that if you employ an HA BP solution and would like to help the HA BP “apply/replay” jobs keep up on the target system when you’ve got a journal intensive job running on the source system, consider installing the Journal Caching PRPQ on the target side as well.

Note: For those who'd like to ascertain the performance advantages in their own shop before making a purchase decision, there's a 60-day trial copy that can be requested as a set of PTFs.

For a **V4R4** 60-day trial copy install:

MF26771 (Slic changes)
SF65861 (XPF API changes)
SA93202 (Test Fix containing QJOSPEED)

For a **V4R5** 60-day trial copy install:

MF26772 (Slic changes)
SF65860 (XPF API changes)
SA93215 (Test Fix containing QJOSPEED)

For a **V5R1** 60-day trial copy install:

MF26785 (Slic changes)
SI01388 (XPF API changes)
SE01985 (Test Fix containing QJOSPEED)

The 60-day trial copy will show up in the QSYS library and will indeed cease providing performance advantages 60 days after being activated. The regular PRPQ, by contrast shows up in the QBJC library.

For additional information and considerations on this PRPQ refer to the following Web sites:

<http://www-1.ibm.com/servers/eserver/series/ha/hajournal.htm>
<http://www-912.ibm.com/supporthome.nsf/document/10000051>

Search for the word 5799BJC. It will bring up the readme file of the PRPQ.

Apply Journal Changes Extended PRPQ (5799-AJC)

There are a number of DDL database operations such as ALTER TABLE which can not automatically be applied by the APYJRNCHG command. In addition, commitment control boundaries during APYJRNCHG are not honored adequately if they spanned these DDL operations. These factors cause APYJRNCHG operations to be impractical and unsatisfying for some SQL customers especially those whose applications generate lots of DDL operations (such as Alter Table) or disruptive native commands (such as RGZPFM). Prior to V5R1 there was an additional concern in HA replication environments, and it stemmed from not having both DDL and DML(Data Manipulation Language) operations recorded on the same journal. Prior to V5R1, some critical object level journal entries only showed up in the audit journal rather than in the regular journal. This often forced shops to try to merge the contents of the two journals. In V5R1 the DDL operations are placed in the same journal as the regular DML database changes (Inserts, deletes and updates).

In V5R1, the Apply Journal Changes Extended PRPQ (5799-AJC) is helpful if your recovery plans include use of a hot site. This PRPQ enhances the hot site recovery capability of the journaling support such that DDL operations such as Alter Table and Create Table can be replayed effortlessly thereby extending the facilities formerly available for hot site recovery.

This should be especially attractive for customers who use applications which make a regular habit of creating new SQL tables during execution. It also will be attractive for shops who perform concurrent install of new software application upgrades since many of these upgrade operations add new columns to existing tables via the Alter Table statement. Some of the most popular ERP (Enterprise Resource Planning) packages in the marketplace fall into this category. This new support also makes it possible for the HA BPs to step up to modernizing their products since many of the object level changes they formerly harvested from the audit journal and then merged with the separate database journal are now all found in one place: the database journal. This should afford the opportunity for them to make their replay processing on the target machine more timely.

This no-charge PRPQ (5799-AJC), available for use with V5R1, will provide you a new command, APYJRNCHGX, that will successfully apply/replay even the DDL operations. The new DDL journal entries are deposited in V5R1 even if you have not installed the PRPQ. In other words the PRPQ is not required to get the new entries deposited. It is only required if you need to apply them. The new command APYJRNCHGX applies both the ordinary journal entries as well as the new DDL journal entries. The following list identifies the SQL statements that produce the new DDL journal entries:

- ▶ CREATE TABLE
- ▶ CREATE VIEW
- ▶ CREATE INDEX
- ▶ DROP TABLE
- ▶ DROP VIEW
- ▶ DROP INDEX
- ▶ ALTER TABLE ADD FIELD
- ▶ ALTER TABLE DROP FIELD
- ▶ ALTER TABLE ALTER FIELD
- ▶ ALTER TABLE ADD CONSTRAINT
- ▶ ALTER TABLE DROP CONSTRAINT
- ▶ RENAME
- ▶ CREATE TRIGGER
- ▶ DROP TRIGGER
- ▶ GRANT
- ▶ REVOKE
- ▶ CHANGE CONSTRAINT
- ▶ CHANGE TRIGGER

Also for the first time in V5R1, the following CL commands are journaled if they operate on journaled tables:

- ▶ ADDPFCST
- ▶ RVKOBJAUT
- ▶ CHGOWN
- ▶ RGZPFM
- ▶ CHGPFCST
- ▶ RMVM
- ▶ CHGPFPFM
- ▶ RMVPFCST
- ▶ CHGPFTRG
- ▶ RNMOBJ(for a physical file)
- ▶ DLTF
- ▶ GRTOBJAUT
- ▶ RNMM

Note: If your applications use any of these statements (and you would like them replayed at a hot site), you'll probably want to consider switching to the new APYJRNCHGX command rather than employing the former APYJRNCHG command.

Tuning tips and considerations

There are some considerations that have to be followed when you use the PRPQ. They are:

- ▶ Unlike the traditional APYJRNCHG command, the new APYJRNCHGX command does not allow you to customize your apply list such that you can selectively apply to only certain database tables within a library. Instead, it applies journal changes to ALL tables in that library.
- ▶ Some DDL operations, especially ALTER TABLE, are naturally long running commands. These long running commands must acquire locks and hold them for extended periods of time. In order to protect against such “hangs” during a long-running APYJRNCHGX replay operation a time-out mechanism can be specified by using the CL command shown below. It should be done before issuing the APYJRNCHGX command.

```
ADDENVVAR QIBM_JO_APPLY_TIMEOUT 'nnnn'
```

The default time out value is 24hrs.

Note: You may want to select a substantially smaller value than the default if you are confident that none of the journaled DDL operations are likely to take that long. On the other hand if you know that you have an enormous sized SQL table referenced by lots of indexes and hence that even 24 hours isn't sufficient, you could use this CL command to select an even longer timeout value thereby helping to insure that the APYJRNCHGX CL command doesn't terminate prematurely by falsely concluding that your ALTER TABLE operation has hung.

1.3.5 Summary of V5R1 enhancements

The benefits derived from the V5R1 journal enhancements are:

- ▶ High availability customers can protect more object types than just database tables
- ▶ There can be a reduction of quantity of data transferred between systems
- ▶ Improved recovery from data loss
- ▶ Fewer damaged data queues
- ▶ Reduction in journaling performance overhead for large batch jobs
- ▶ Ability to apply DDL operations at a hot site
- ▶ New facilities to speed up HA BP replay operations on a target machine



The testing environment

In this chapter we describe the hardware and software environments that were employed to conduct the various tests described in this redbook.

2.1 The hardware

The Banesco database and applications encompass many terabytes of data and require an iSeries with substantial processing capabilities. In order to match the actual Banesco production environment, we conducted our tests at the IBM Teraplex Integration Center. We selected a representative subset of this application whose files consumed approximately 400 MB.

Figure 2-1 below shows a basic overview of the hardware we used to conduct the various tests described in this redbook.

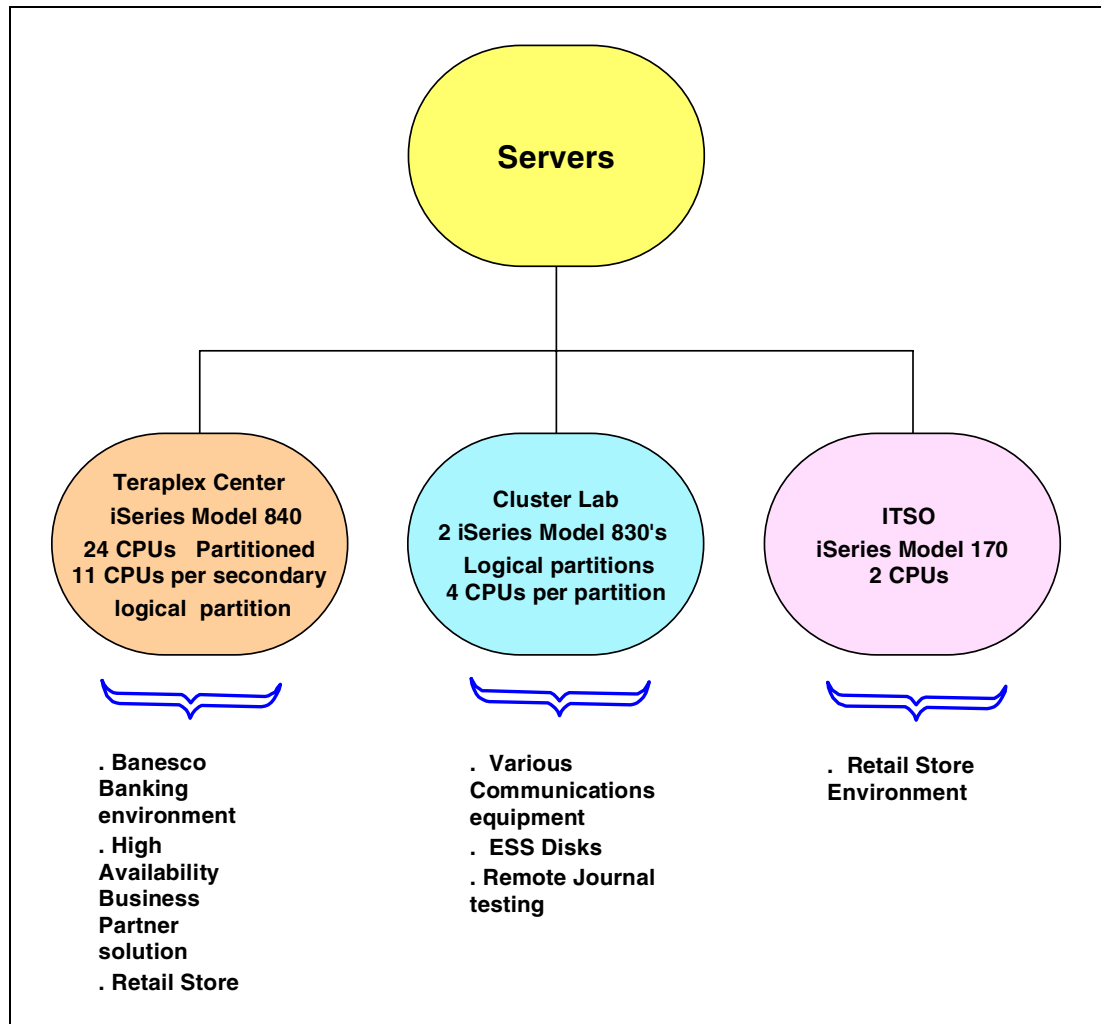


Figure 2-1 Hardware overview

As you can see from the above, we used four different iSeries servers spread across three labs to conduct our tests. Each one of these servers was chosen based on their specific hardware configuration in order to cater for each particular application environment. The specific hardware of each of these servers is discussed in greater detail later in this section.

2.1.1 Teraplex Center

The iSeries IBM Teraplex Center is a leader in business intelligence scalability testing. Located in Rochester, Minnesota, USA, this center is designed to investigate, research and demonstrate very large database (VLDB) business intelligence implementations on iSeries. Here, specialists perform scalability and stress testing on internal, business partner and customer data that can span over several terabytes. At the Teraplex Center, customers and business partners integrate and test IBM hardware with both IBM and business partner software.

The Teraplex Center mission is:

- ▶ To demonstrate scalability and performance (go big, then go break it), with the aim to find problems before customers encounter them in their production environments
- ▶ To prove to customers that their data warehouses will scale on-site
- ▶ To create and solve problems so that customers do not have to deal with them in their production environments

Since 1996, IBM has invested \$76 million (US) to create Teraplex Centers for the iSeries, xSeries, pSeries and zSeries. Each of these centers is equipped with several of the most powerful systems of their type and staffed with highly skilled business intelligence (BI) specialists.

For more information about the Teraplex Centers, refer to the following URLs:

<http://www.as400.ibm.com/developer/bi/teraplex/>

<http://www.ibm.com/solutions/businessintelligence/teraplex/>

The iSeries server

In order to replicate the actual Banesco production environment, we configured an iSeries server at the Teraplex Center with the hardware specifications as shown in Figure 2-3 to conduct our tests.

All figures in this section detailing the physical hardware layout of the iSeries server were created using the standard V5R1M0 iSeries Operations Navigator graphical interface.

Note: Operations Navigator is the windows-based interface used to configure, monitor, and manage the OS/400 environment. Operations Navigator was originally available with V4R3 with significant enhancements added in subsequent releases, primarily in the Database and Hardware - Disk configuration areas. While OS/400 continues to provide a powerful set of command level interfaces for managing your environment, Operations Navigator should be your first choice for managing jobs, printed output, disk storage, simple clustering configurations, system resource utilization, message responses, TCP/IP-based network configuration, file system content, database, security and system value management.

In this chapter we will use the Operations Navigator interface to describe the hardware configurations that we used in our tests. Figure 2-2 illustrates the initial window which appears when you click on the Operations Navigator icon on your desktop.

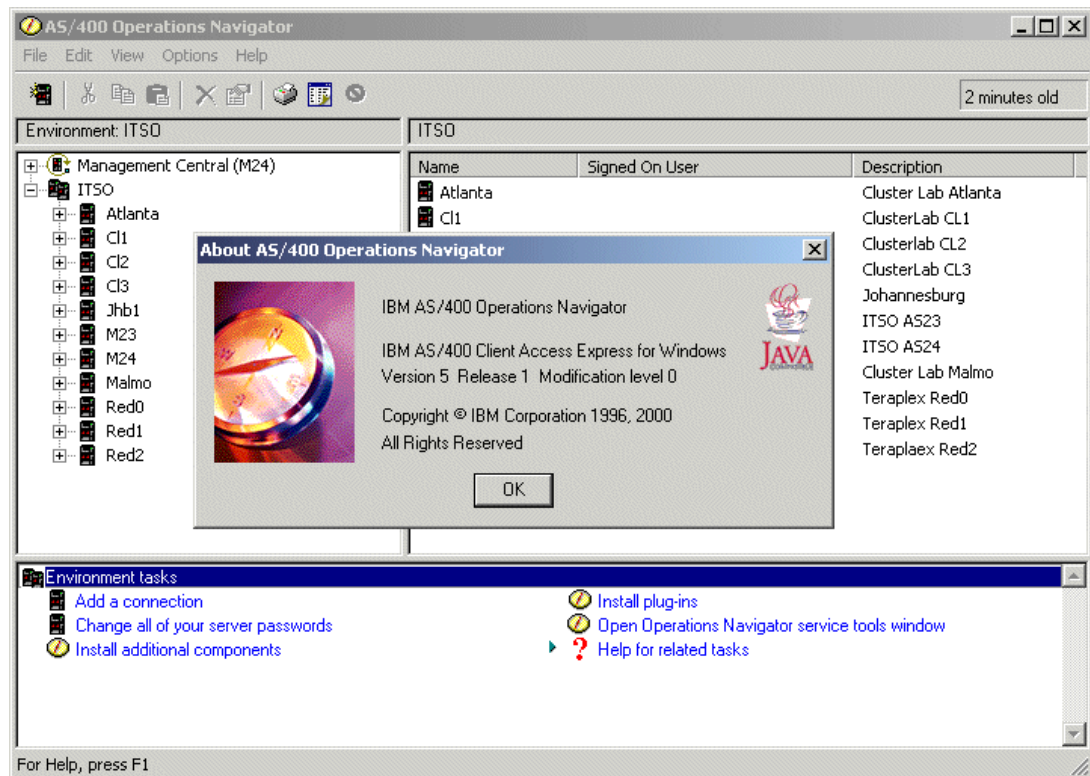


Figure 2-2 iSeries Operations Navigator

Once you have clicked on the corresponding server you can graphically view the hardware configuration of the server.

As you can see from Figure 2-3, we used an iSeries Type 9406, Model 840 with 24 dedicated processors. This high-end iSeries server had a total of 96GB of main memory and a total disk storage capacity of 1.9TB. All the Combined Function I/O Processors (CFIOP) were Type 2843 and all disk units were attached to Type 2748 Multiple Function I/O Adaptors (MFIOA), each providing 26MB of write cache.

Note: It's important to note the write cache size, because as we will see later, the quantity of write cache is one of the most influential hardware choices you can make if you want to optimize journal performance.

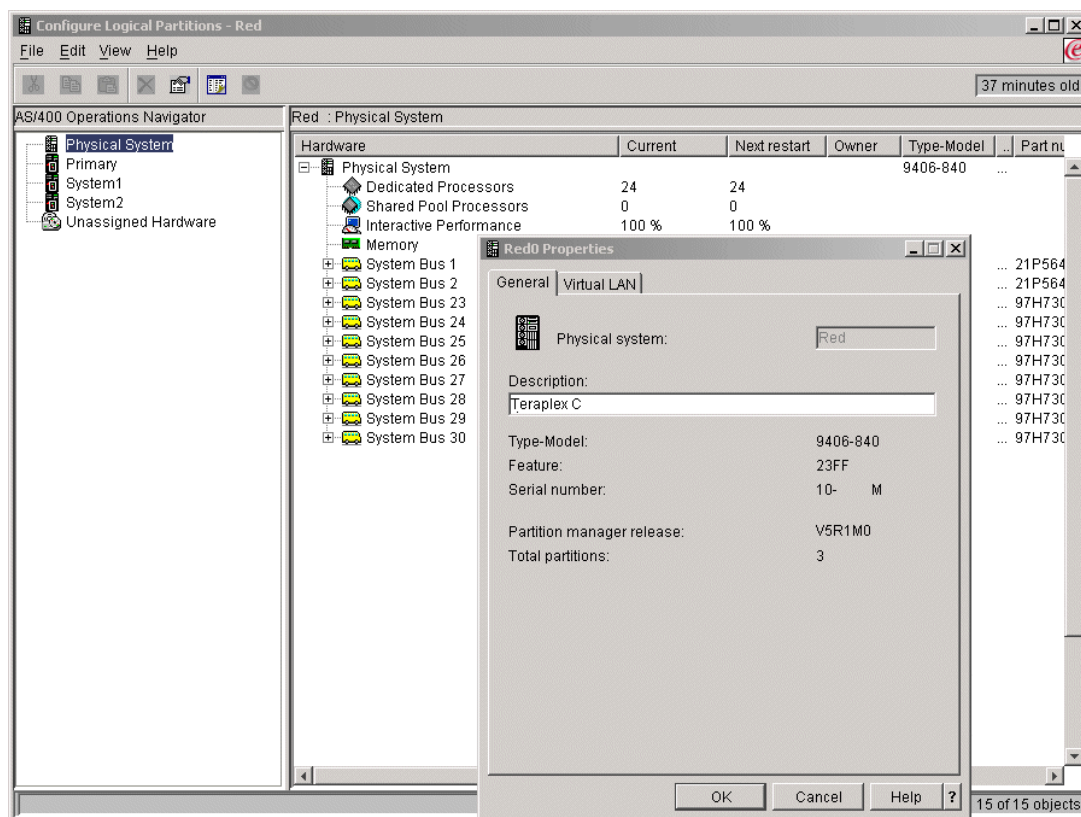


Figure 2-3 iSeries hardware at the Teraplex Center

We further configured this server into three logical partitions (LPAR).

Figure 2-4 shows an overview of the LPAR configuration used at the Teraplex Center. LPAR1 in Figure 2-4 depicts the primary or controlling partition. LPAR2 and LPAR3 are the secondary logical partitions. A virtual LAN provides communication between the logical partitions. We performed all our Teraplex Center testing in partitions 2 and 3.

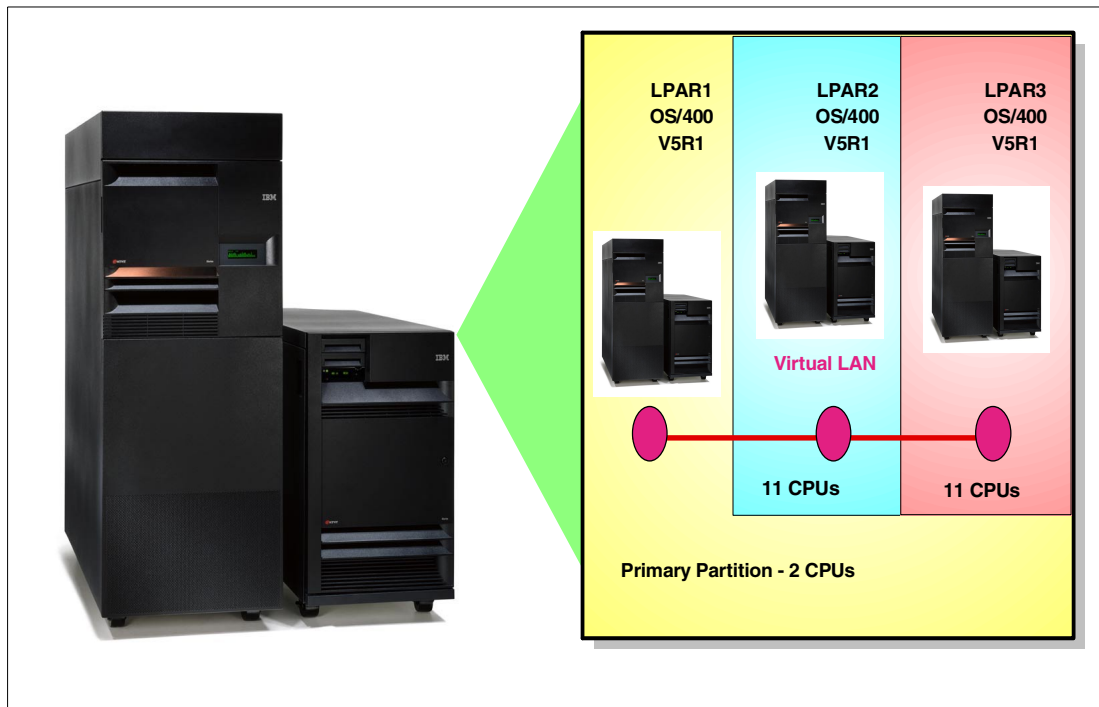


Figure 2-4 LPAR overview

Note: For more information on *Logical Partitioning* refer to *LPAR Configuration and Management Working with iSeries Logical Partitions*, SG24-6251.

The hardware configuration of the primary partition is shown in Figure 2-5 and the two secondary partitions are shown in Figure 2-6 and Figure 2-7.

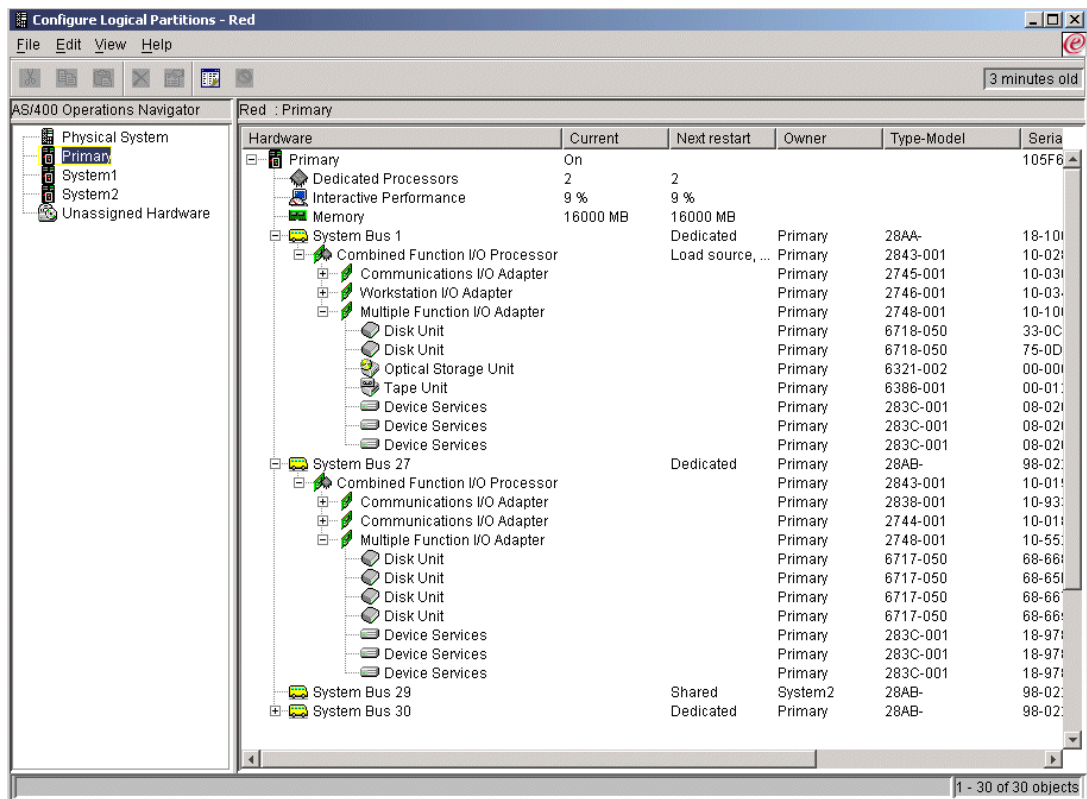


Figure 2-5 Primary partition configuration

We configured the primary partition with minimum resources, as no actual tests were conducted in this partition. Its sole function was to control the secondary partitions and to provide an environment to analyze the test results collected in the two secondary partitions.

The primary partition was assigned two main processors and it had 16GB of main memory. The system ASP hosted the operating system and a user ASP was created with four disk drives to hold the performance data we produced and a saved copy of journal receivers for their analysis after each run. This allowed us to conduct post-test analysis without impacting subsequent concurrent runs.

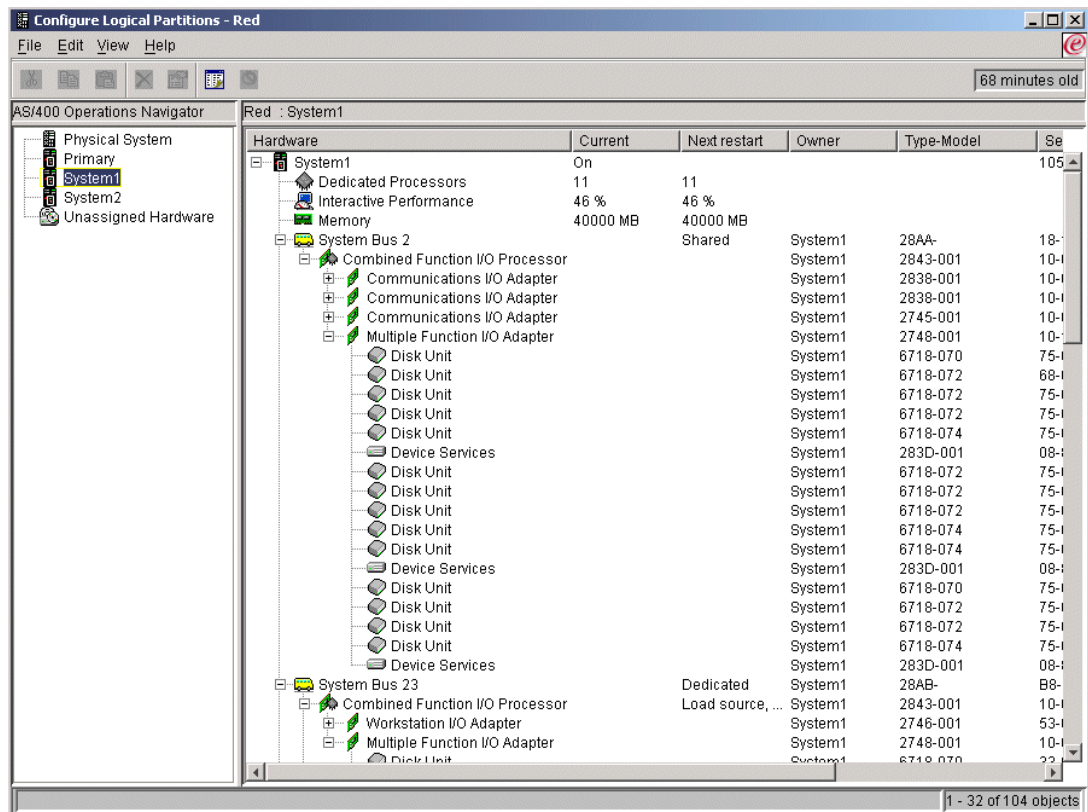


Figure 2-6 Partition 2 (LPAR2) configuration

As you can see from Figure 2-6 and Figure 2-7, the two secondary partitions have identical processors and memory pool sizes. This allowed us to conduct tests in both of these partitions simultaneously and enabled us to directly compare the test results collected from each of these logical partitions. Each partition had 11 main processors assigned to it and each had 40GB of main storage.

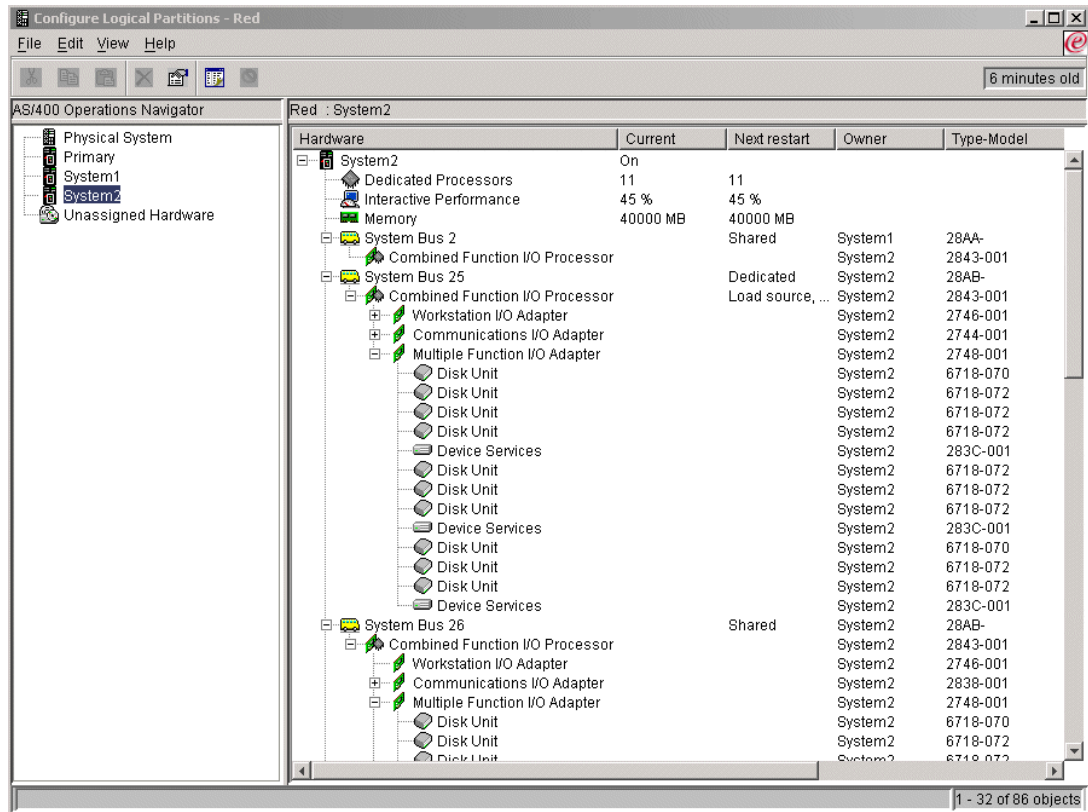


Figure 2-7 Partition 3 (LPAR3) configuration

The V5R1 version of Operations Navigator added new functions and enhanced previous ones which allow you to manage disk. In this chapter we will use the new added functions to describe the disk configurations that we used for our tests.

The disk configuration of the two secondary partitions are described in Figure 2-8 and Figure 2-9. The system auxiliary storage pools (ASPs) of both logical partitions have identical configurations. They both have 44 disk drives.

While the system ASP, used to house the Banesco application and tables, was deliberately configured to be identical for both partitions, the user ASPs, used to house the journal receivers, were intentionally *not* identical. This was a deliberate decision. It afforded us a richer variety of hardware and configuration test beds which we used to investigate the performance impact of user ASP choices and alternatives.

LPAR2 had two user ASPs as shown in Figure 2-8.

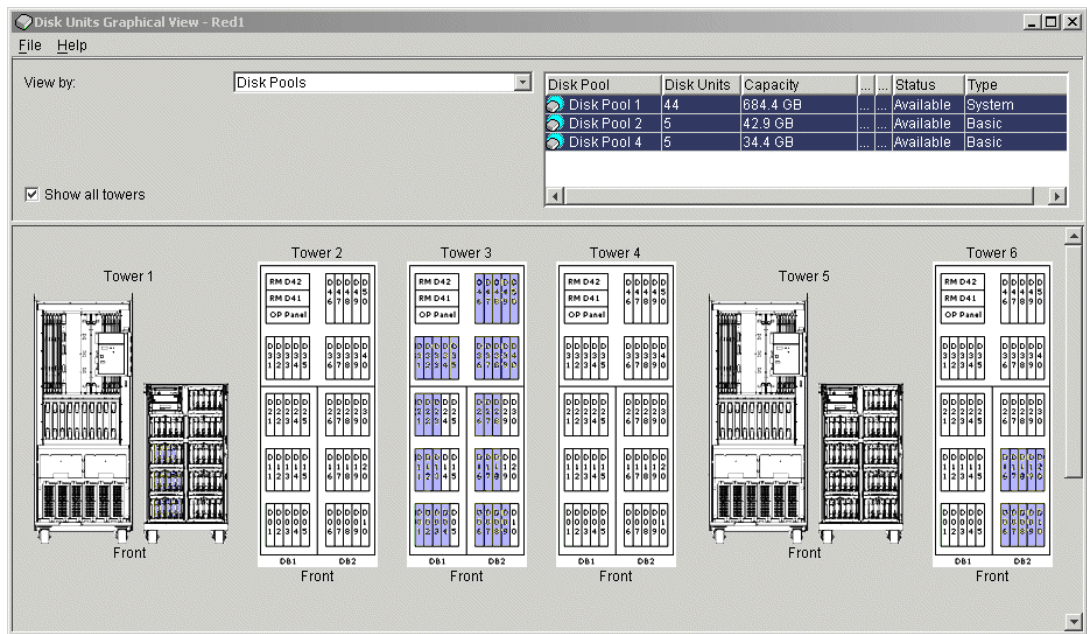


Figure 2-8 Partition 2 (LPAR2) disk configuration

LPAR3 had three user ASPs as shown in Figure 2-9.

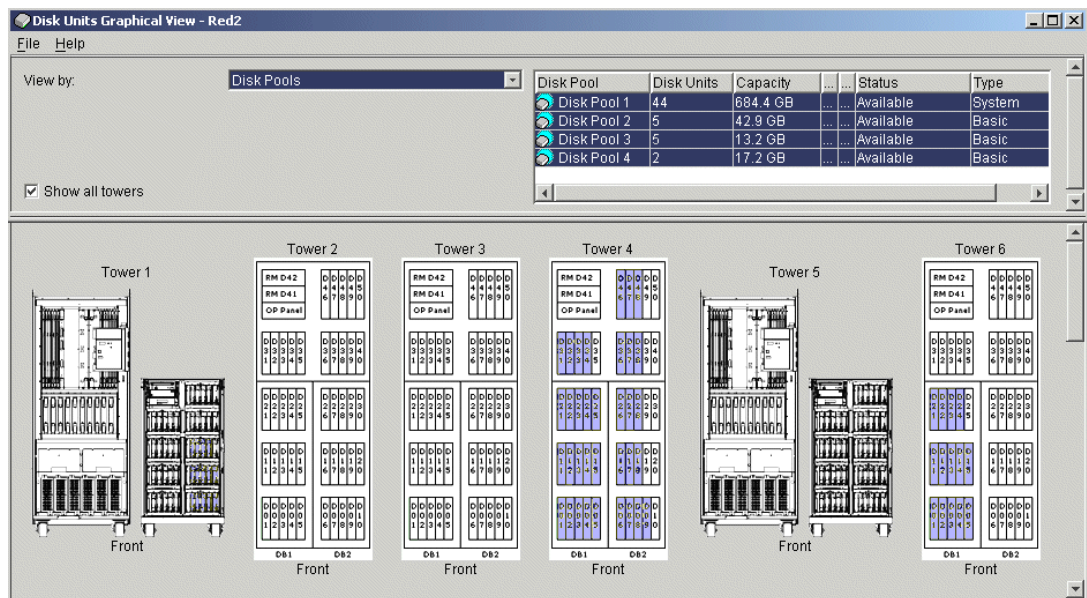


Figure 2-9 Partition 3 (LPAR3) disk configuration

Armed with different user ASPs we were able to compare different disk configurations such as mirrored user ASPs, RAIDed user ASPs, and slower disk drives versus faster disk drives. Figure 2-10 and Figure 2-11 below depict the configuration of the user ASPs for logical partition 2.

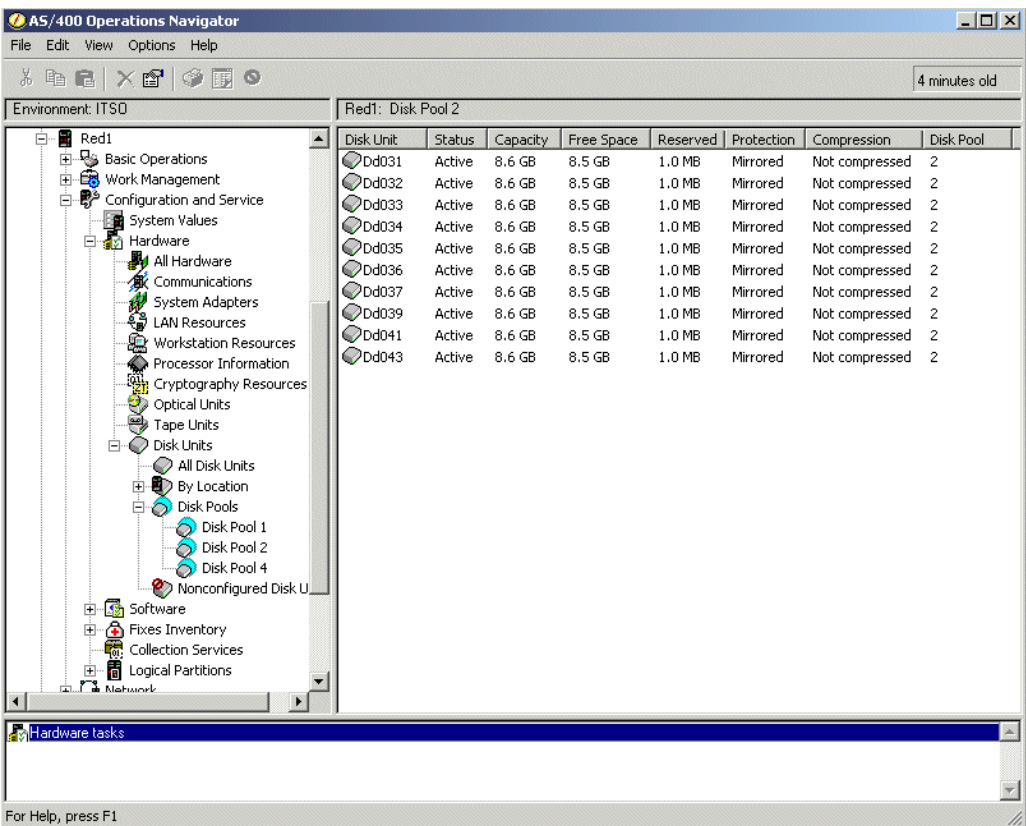


Figure 2-10 User ASP2 in partition 2 (LPAR2)

User ASP 2 in partition 2 consisted of ten mirrored disk drives as shown in Figure 2-10, with a storage capacity of 8.6GB each and with a rotational speed of 10,000 revolutions per minute (rpm).

User ASP 4 in partition 2 had RAID-5 protection and consists of five disk drives as shown in Figure 2-11. Each disk unit had a storage capacity of 8.6GB and a rotational speed of 10,000 rpm.

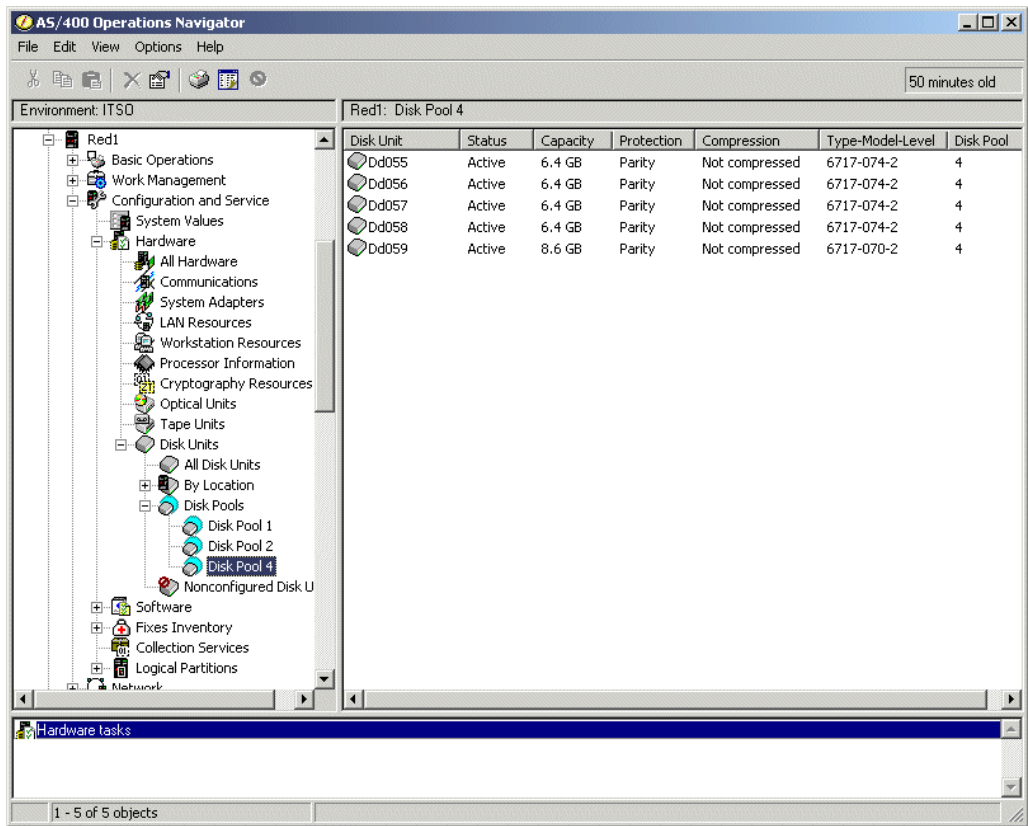


Figure 2-11 User ASP4 in partition 2 (LPAR2)

Partition 3 had 3 user ASPs. Figure 2-12 shows the detail of user ASP 2 in this partition.

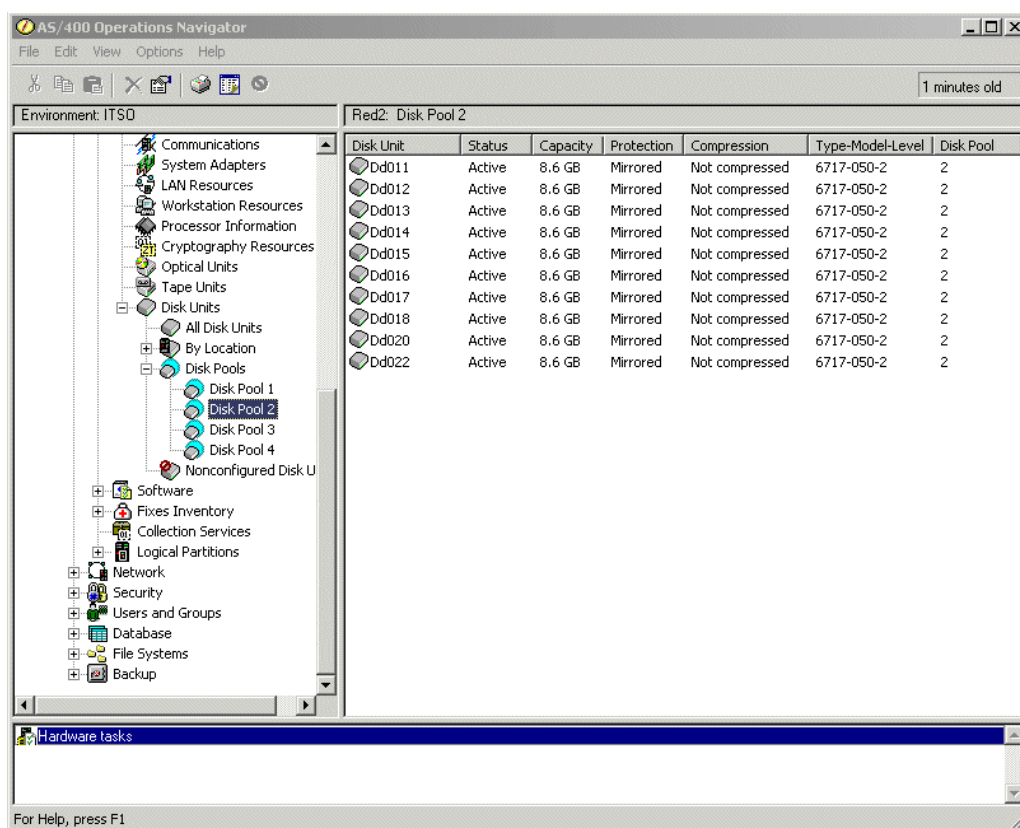


Figure 2-12 User ASP2 in partition 3 (LPAR3)

This user ASP consisted of ten mirrored disks. Each disk unit had a storage capacity of 8.6GB and a rotational speed of 10,000 rpm.

Figure 2-13 shows user ASP 3 of this partition.

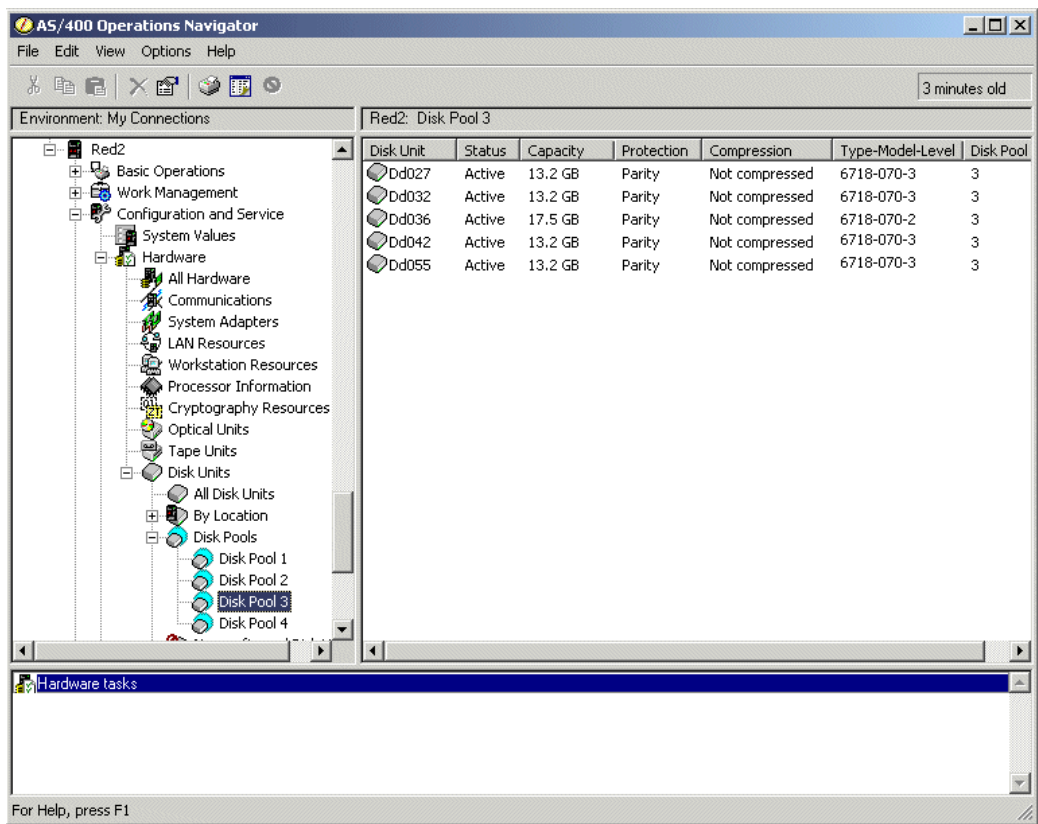


Figure 2-13 User ASP3 in partition 3 (LPAR3)

As shown in Figure 2-13, user ASP 3 consists of five disks with RAID-5 protection. Each disk unit has a storage capacity of 17GB and a rotational speed of 7200 rpm.

Note: Since the five disk drives are RAID protected, the available space is 80% of the total capacity. If you add up the available space it adds up to 70.3GB which is 80% of 87.5GB.

User ASP 4 consisted of four mirrored disks, each with a storage capacity of 8.6GB and a rotational speed of 10,000 rpm as shown in Figure 2-14.

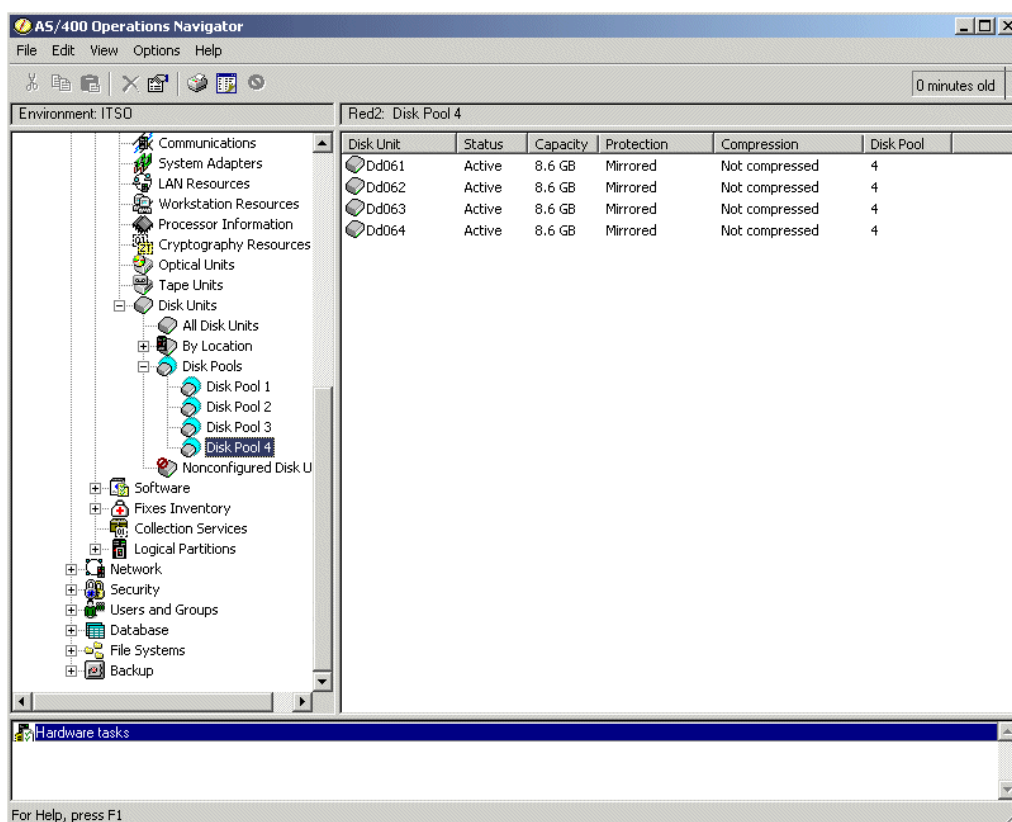


Figure 2-14 User ASP4 in Partition 3 (LPAR3)

2.1.2 The “Cluster Lab”

This IBM internal laboratory environment is equipped with high-end iSeries servers, providing facilities for testing and benchmarking large scale applications and solutions requiring clustered configurations of high availability software. It is ideally positioned to facilitate testing under various communication infrastructures, speeds and protocols. Its ability to provide iSeries servers in a clustered environment made it an ideal environment for conducting tests pertaining to remote journaling and High Availability Business Partner solutions. For more information on clustering, refer to the following URLs:

<http://www.redbooks.ibm.com/redpapers/pdfs/redp0111.pdf>
<http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245194.pdf>

Figure 2-15 provides a basic overview of the hardware layout of the iSeries servers used in the Cluster Lab environment.

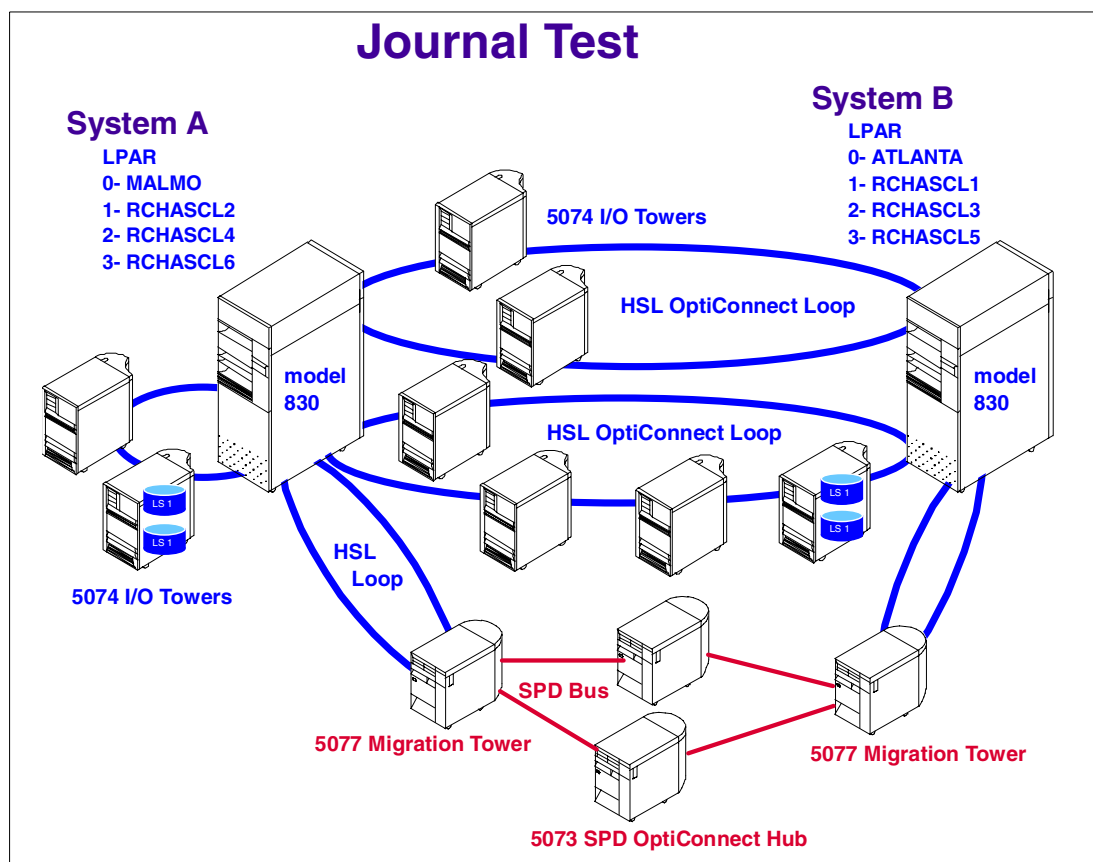


Figure 2-15 Cluster Lab hardware overview

This configuration consisted of two iSeries Model 830 servers, named Malmo and Atlanta, in a clustered environment, using four shared disk towers and four private disk towers to facilitate switch-over. Optical HSL cables connected these towers into HSL OptiConnect loops. Each server was configured with a primary partition and three secondary logical partitions (LPAR).

Figure 2-16 shows the physical hardware configuration of system Malmo(A) and the hardware configuration of system Atlanta(B) is shown in Figure 2-17.

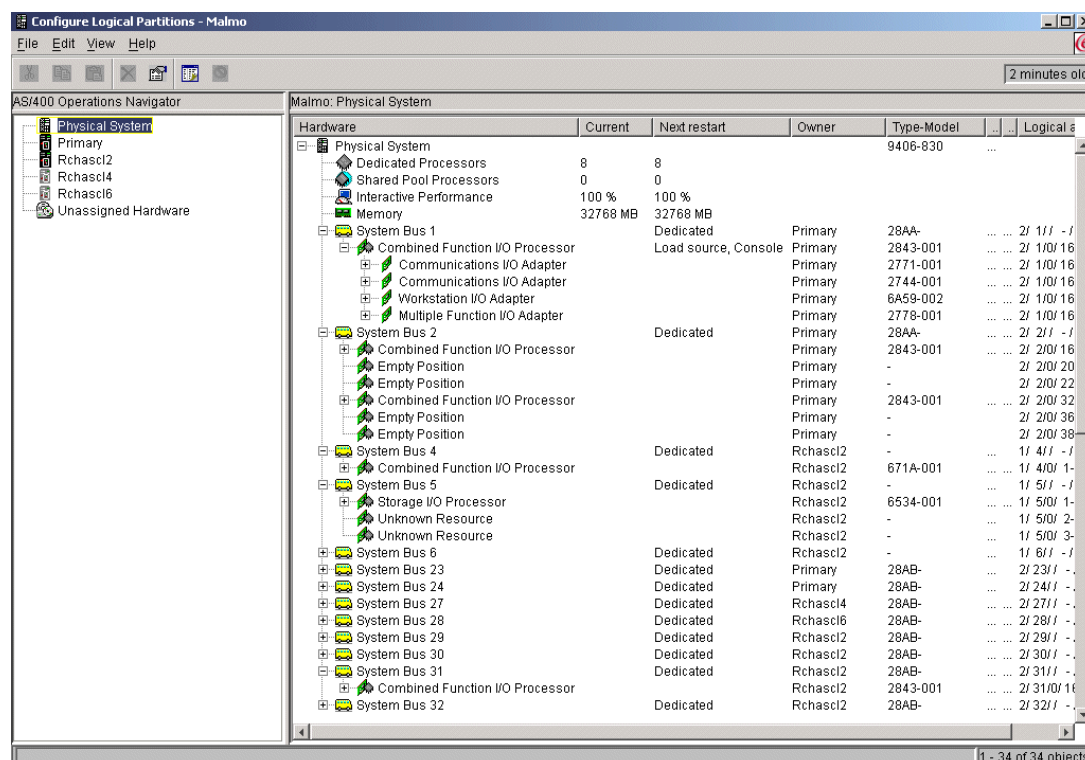


Figure 2-16 Malmo hardware resources

Figure 2-16 shows that this is an iSeries Model 830 server with eight dedicated processors and 32GB memory. This server is configured with a primary partition and three secondary logical partitions. The details regarding the primary partition are shown in Figure 2-18 and that of the secondary logical partition RCHASCL2, in Figure 2-19.

Configure Logical Partitions - Atlanta

File Edit View Help

2 minutes old

AS/400 Operations Navigator

Atlanta: Physical System

Physical System

Hardware	Current	Next restart	Owner	Type-Model	Logical address
Physical System				9406-830	...
Dedicated Processors	8	8			
Shared Pool Processors	0	0			
Interactive Performance	100 %	100 %			
Memory	32768 MB	32768 MB			
System Bus 1		Dedicated	Primary	28AA-	...
Combined Function I/O Processor		Load source, Console	Primary	2843-001	...
Communications I/O Adapter			Primary	2771-001	...
Communications I/O Adapter			Primary	2744-001	...
Workstation I/O Adapter			Primary	6A59-002	...
Multiple Function I/O Adapter			Primary	2778-001	...
Disk Unit			Primary	6717-074	...
Disk Unit			Primary	6717-074	...
Disk Unit			Primary	6717-072	...
Disk Unit			Primary	6717-072	...
Disk Unit			Primary	6717-072	...
Optical Storage Unit			Primary	6330-002	...
Tape Unit			Primary	6386-001	...
Device Services			Primary	283C-001	...
Disk Unit			Primary	6717-070	...
Disk Unit			Primary	6717-074	...
Disk Unit			Primary	6717-072	...
Disk Unit			Primary	6717-072	...
Disk Unit			Primary	6717-072	...
Device Services			Primary	283C-001	...
Disk Unit			Primary	6717-070	...
Disk Unit			Primary	6717-070	...
Disk Unit			Primary	6717-074	...
Disk Unit			Primary	6717-072	...
Disk Unit			Primary	6717-072	...
Device Services			Primary	283C-001	...
System Bus 2		Dedicated	Primary	28AA-	...
System Bus 4		Dedicated	Rchasc11	-	...
System Bus 5		Dedicated	Rchasc11	-	...
System Bus 23		Dedicated	Rchasc15	28AB-	...
System Bus 24		Dedicated	Rchasc15	28AB-	...
System Bus 25		Dedicated	Rchasc13	28AB-	...
System Bus 26		Dedicated	Rchasc13	28AB-	...
System Bus 27		Dedicated	Primary	28AB-	...

1 - 40 of 46 objects

The second server is also an iSeries Model 830 server with eight dedicated processors and 32GB memory. This server is configured with a primary partition and three secondary logical partitions. The details regarding the primary partition are shown in Figure 2-20 and that of the secondary logical partition RCHASCL1, in Figure 2-21. The hardware resources of the secondary logical partition RCHASCL3 is depicted in Figure 2-22.

The details regarding the primary partition of system A are shown in Figure 2-18.

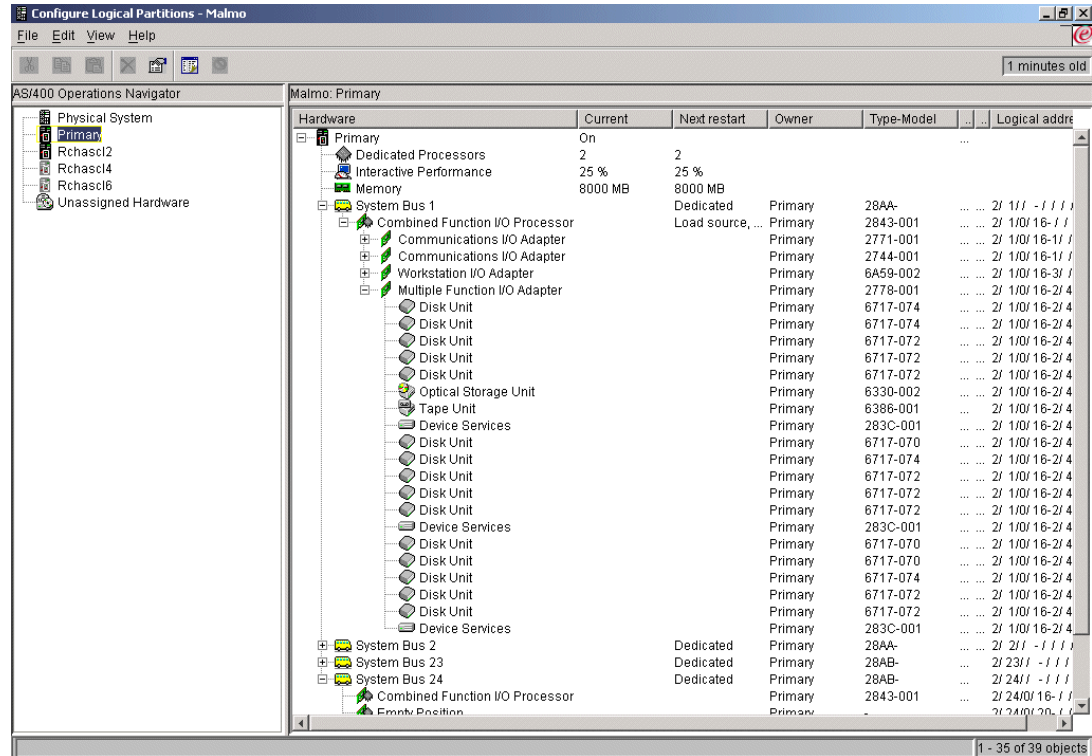


Figure 2-18 Malmø primary partition

This primary partition as shown in Figure 2-18 has two dedicated processors configured and 8GB of memory. The disk units are all attached to Model 2778 Multiple Function I/O Processors (MFIO), which provide 104MB of write cache each.

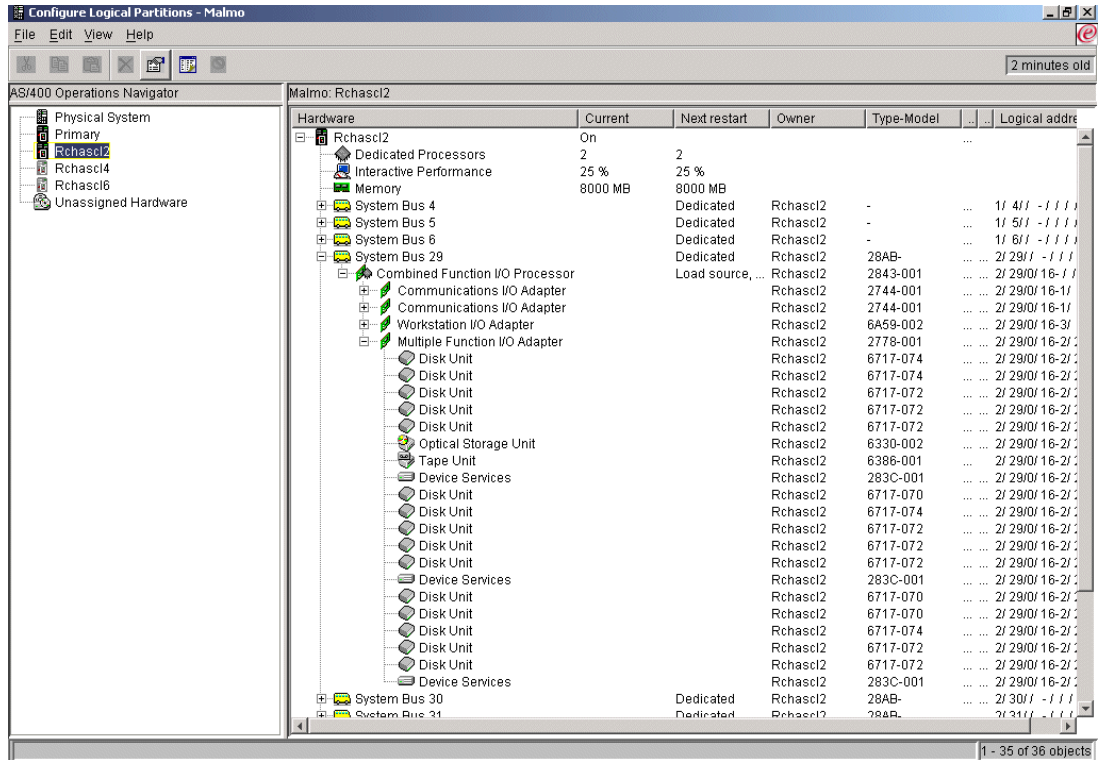


Figure 2-19 Secondary logical partition CL2 on MALMO(A)

The secondary logical partition (RCHASCL2) as shown in Figure 2-19 has two dedicated processors configured with 8GB of memory. All disk units are attached to Model 2778 Multiple Function I/O Processors (MFIO), which provide 104MB of write cache each.

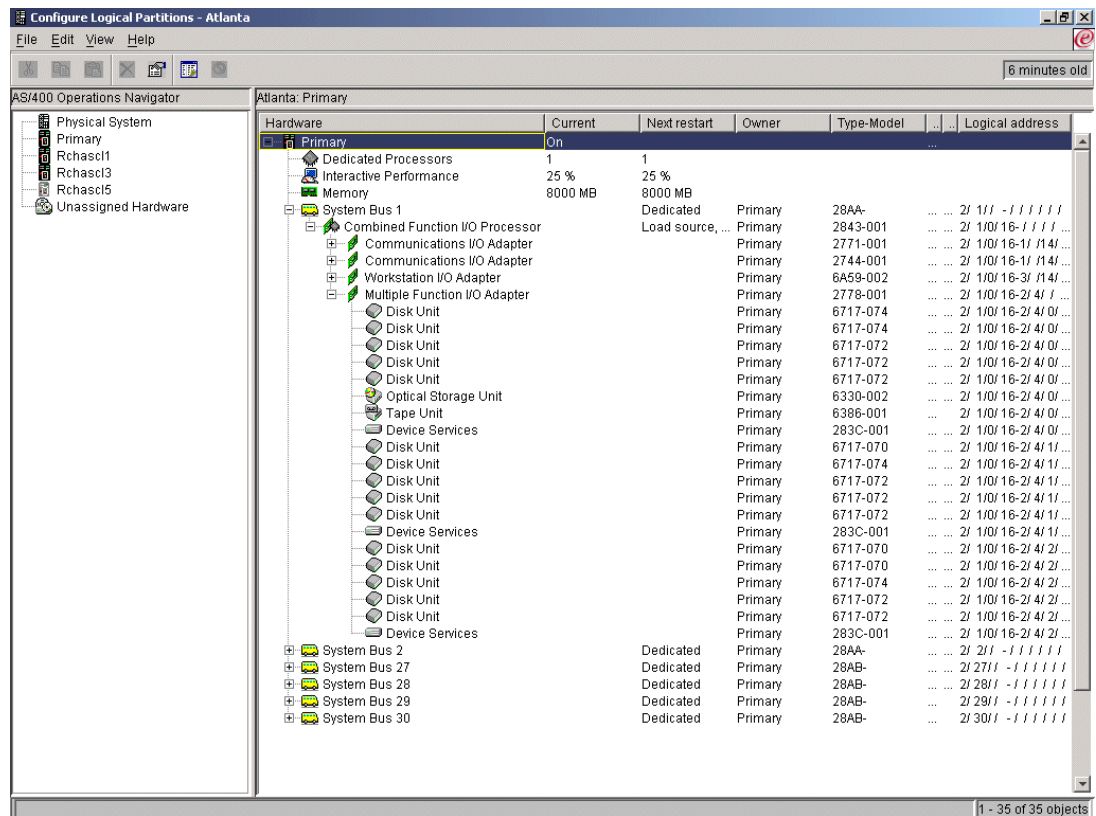


Figure 2-20 Atlanta(B) primary partition

The primary partition on system B was configured with a single dedicated processor and 8GB of memory. The disk units are all attached to Model 2778 Multiple Function I/O Processors (MFIO), which provide 104MB of write cache each.

Configure Logical Partitions - Atlanta

File Edit View Help

AS/400 Operations Navigator

Atlanta: Rchasc1

Physical System

- Primary
- Rchasc1**
- Rchasc3
- Rchasc5
- Unassigned Hardware

Hardware	Current	Next restart	Owner	Type-Model	Logical address
Rchasc1	On				...
Dedicated Processors	4	4			
Interactive Performance	25 %	25 %			
Memory	8768 MB	8768 MB			
System Bus 4		Dedicated	Rchasc1	-	1/ 4/1 - 1/1/1/1
System Bus 5		Dedicated	Rchasc1	-	1/ 5/1 - 1/1/1/1
Shared Bus Adapter			Rchasc1	2685-000	1/ 5/0/ 1-1/1/1/1
System Bus 37		Dedicated	Rchasc1	28A-	2/ 37/1 - 1/1/1/1
Combined Function I/O Processor		Load source,...	Rchasc1	2843-001	2/ 37/0/ 16-1/1/1/1
Communications I/O Adapter			Rchasc1	2744-001	2/ 37/0/ 16-1/1/1/1
Communications I/O Adapter			Rchasc1	2744-001	2/ 37/0/ 16-1/1/1/1
Workstation I/O Adapter			Rchasc1	6A59-002	2/ 37/0/ 16-3/1/1/1
Multiple Function I/O Adapter			Rchasc1	2778-001	2/ 37/0/ 16-2/2/1/1
Disk Unit			Rchasc1	6717-074	2/ 37/0/ 16-2/2/0...
Disk Unit			Rchasc1	6717-074	2/ 37/0/ 16-2/2/0...
Disk Unit			Rchasc1	6717-072	2/ 37/0/ 16-2/2/0...
Disk Unit			Rchasc1	6717-072	2/ 37/0/ 16-2/2/0...
Disk Unit			Rchasc1	6717-072	2/ 37/0/ 16-2/2/0...
Optical Storage Unit			Rchasc1	6330-002	2/ 37/0/ 16-2/2/0...
Tape Unit			Rchasc1	6386-001	2/ 37/0/ 16-2/2/0...
Device Services			Rchasc1	283C-001	2/ 37/0/ 16-2/2/0...
Disk Unit			Rchasc1	6717-070	2/ 37/0/ 16-2/2/1...
Disk Unit			Rchasc1	6717-074	2/ 37/0/ 16-2/2/1...
Disk Unit			Rchasc1	6717-072	2/ 37/0/ 16-2/2/1...
Disk Unit			Rchasc1	6717-072	2/ 37/0/ 16-2/2/1...
Disk Unit			Rchasc1	6717-072	2/ 37/0/ 16-2/2/1...
Device Services			Rchasc1	283C-001	2/ 37/0/ 16-2/2/1...
Disk Unit			Rchasc1	6717-070	2/ 37/0/ 16-2/2/2...
Disk Unit			Rchasc1	6717-070	2/ 37/0/ 16-2/2/2...
Disk Unit			Rchasc1	6717-074	2/ 37/0/ 16-2/2/2...
Disk Unit			Rchasc1	6717-072	2/ 37/0/ 16-2/2/2...
Disk Unit			Rchasc1	6717-072	2/ 37/0/ 16-2/2/2...
Device Services			Rchasc1	283C-001	2/ 37/0/ 16-2/2/2...
System Bus 38		Dedicated	Rchasc1	28A-	2/ 38/1 - 1/1/1/1
Combined Function I/O Processor			Rchasc1	2843-001	2/ 38/0/ 16-1/1/1/1
Communications I/O Adapter			Rchasc1	2838-001	2/ 38/0/ 16-1/1/1/1
Communications I/O Adapter			Rchasc1	2817-001	2/ 38/0/ 16-1/1/1/1
Multiple Function I/O Adapter			Rchasc1	2778-001	2/ 38/0/ 16-2/2/1/1
Disk Unit			Rchasc1	6717-072	2/ 38/0/ 16-2/2/0...

1 - 40 of 58 objects

All disk units are attached to Model 2778 Multiple Function I/O Processors (MFIOP), which provide 104MB of write cache each.

Another secondary logical partition was configured with two dedicated processors and 8GB of memory as shown in Figure 2-22. All disk units are attached to Model 2778 Multiple Function I/O Processors (MFIO), which provide 104MB of write cache each.

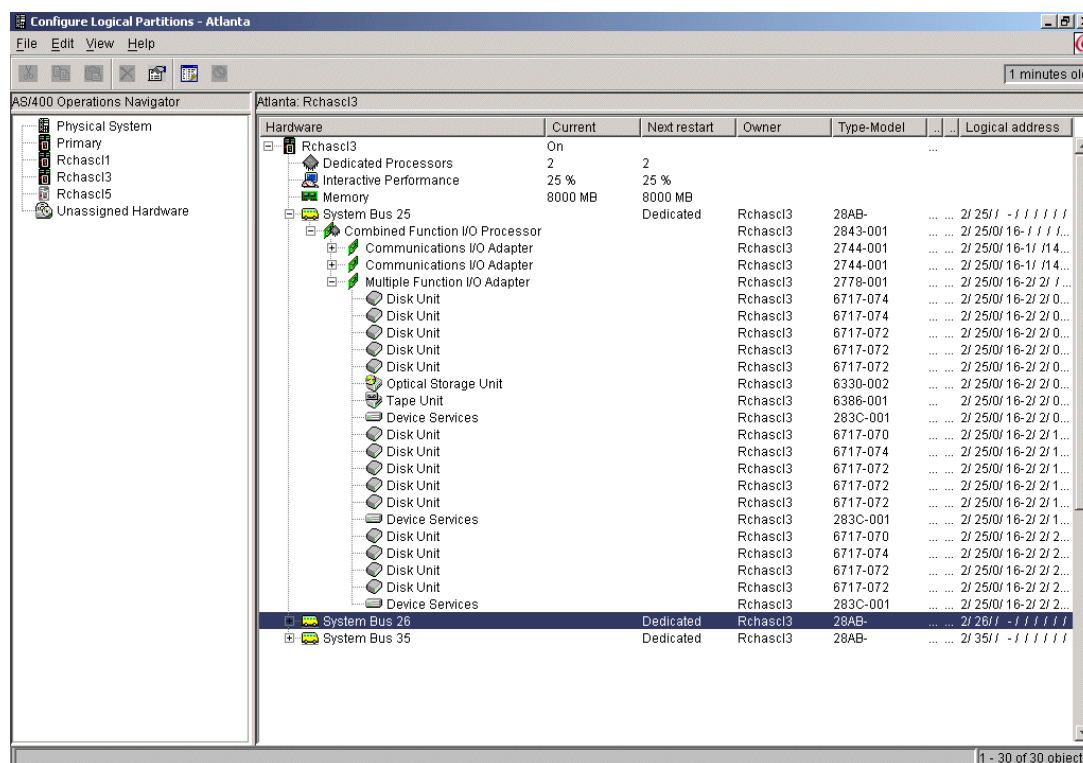


Figure 2-22 Secondary logical partition RCHASCL3 on System B

The logical partitions RCHASCL5 and RCHASCL6 were not used for any tests described in this redbook.

As previously indicated, this environment was primarily used to conduct tests relating to remote journaling. In order to determine the impact of different communication infrastructure on the performance of remote journaling, this environment was equipped with all the relevant hardware to facilitate communications as indicated in Table 2-1.

Table 2-1 Communication methods and protocols

Communication interface type	Cable/physical layer	Communication protocol		
		TCP/IP	SNA	Native OptiConnect
Virtual OptiConnect	Internal Bus			X
HSL OptiConnect	Cable	X		X
SPD OptiConnect	Fibre optic	X		X
Virtual Lan	Internal Bus	X		
1 Gbps Ethernet	Fiber cable	X		
155 Mbps ATM	Fiber cable	X	X	
100 Mbps Ethernet	Fiber cable	X	X	
10 Mbps Ethernet	RJ-45	X	X	

Communication interface type	Cable/physical layer	Communication protocol		
		TCP/IP	SNA	Native OptiConnect
100 Mbps Token Ring	RJ-45	X	X	
16 Mbps Token Ring	RJ-45	X	X	
4 Mbps Token Ring	RJ-45	X	X	

2.1.3 The ITSO laboratory

This laboratory which is dedicated to the ITSO (International Technical Support Organization) in Rochester, Minnesota, USA, has a variety of iSeries servers. These servers are used for testing functions and applications for the purpose of publishing redbooks.

We selected an iSeries Model 170 to use in our testing scenarios. We specifically chose this iSeries server because we wanted to determine what the impact of journal would be on a much smaller server with less processing power and hardware capacity.

Figure 2-23 below shows some of the hardware resources of this iSeries server.

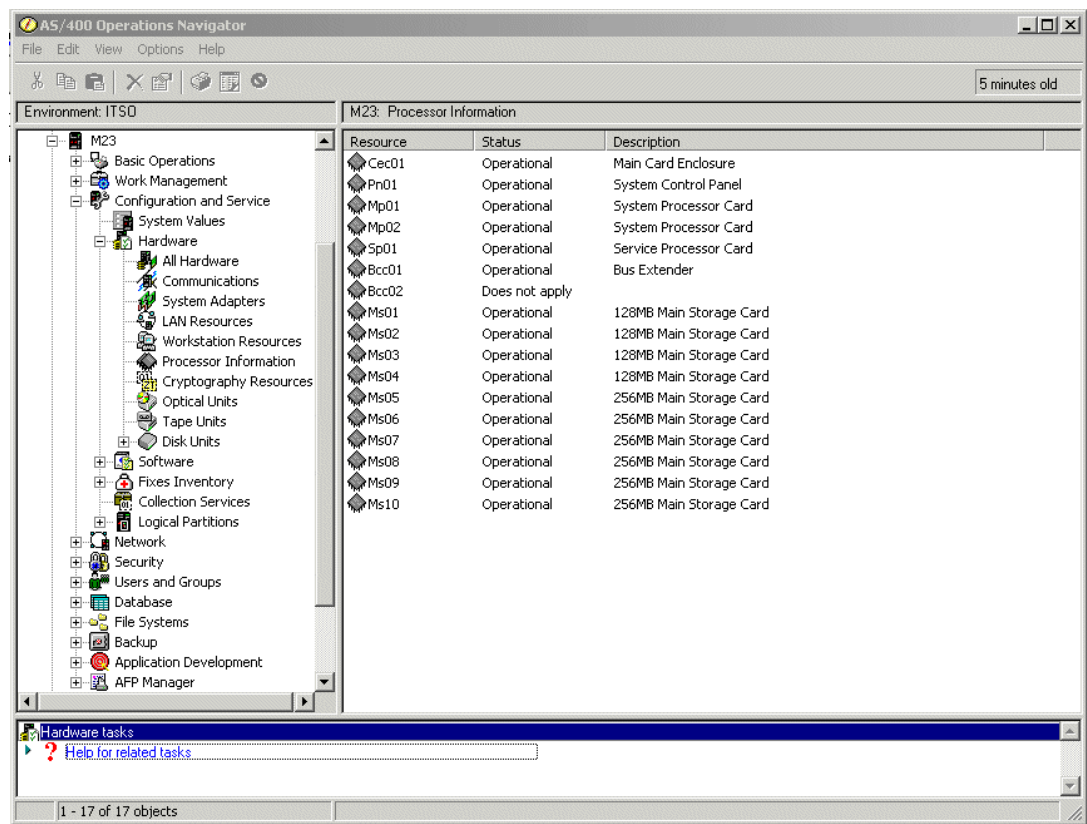


Figure 2-23 ITSO iSeries configuration

As you can see from the above, this is a dual processor server with 2048 MB of memory.

It also had ten disk drives configured in a single system auxiliary storage pool (ASP) with RAID-5 protection. No user ASPs were configured. Each disk drive had a storage capacity of 8.5GB and a rotational speed of 7200 rpm. The I/O Adaptor was a Type 2741 IOA which provides only 4MB of write cache.

2.2 Application environments

In order to conduct our testing we used a variety of application environments. As previously mentioned, we used a real customer overnight batch banking environment. We also did some tests using a retail store application which simulates an interactive workload.

Our intent was to insure that readers of this redbook could observe the journaling performance effects in both batch and interactive environments, on both large and small servers, using both fast and slower disk drives, using both fast and slower communications gear. To that end we varied both the hardware and the applications.

Figure 2-24 shows a basic overview of the applications we used to conduct the tests described in this redbook.

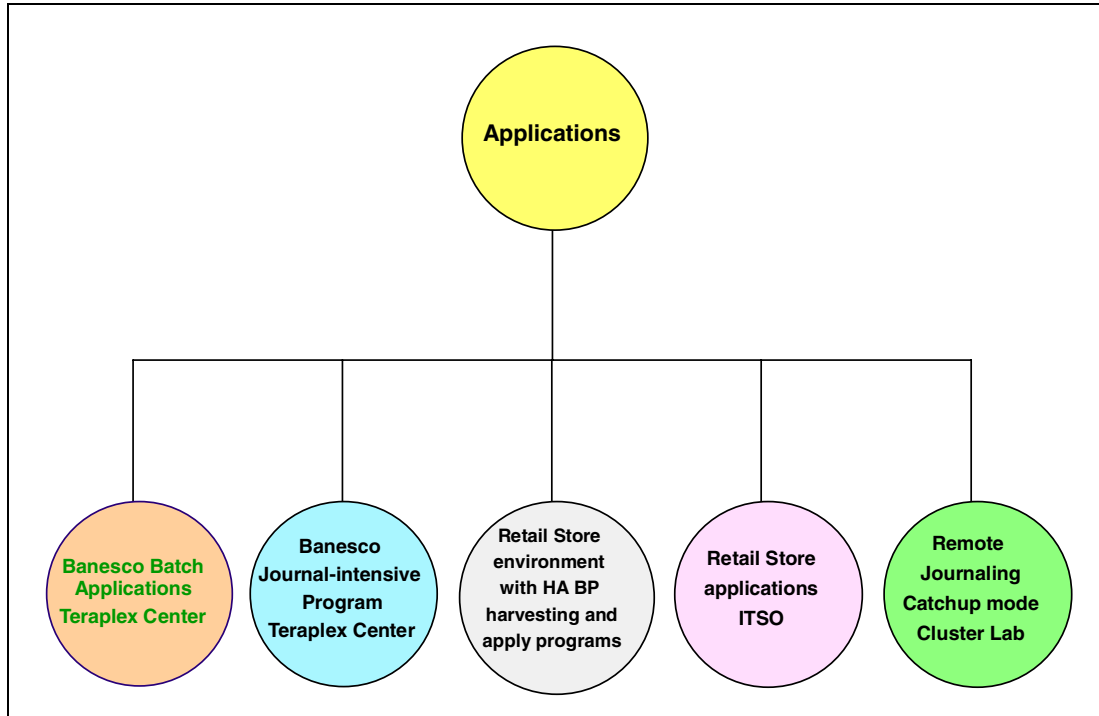


Figure 2-24 Application overview

Each one of these applications were tested in a specific hardware environment that could cater for the specific requirements of these applications and that could provide us with a realistic testing environment.

Note: Obviously, these were ideal testing environments with no other background work taking place on the machine. In that sense, actual performance in more realistic environments in which your batch jobs must share the hardware resources with other programs and interactive work will probably not achieve results quite this good.

2.2.1 Banesco's end-of-day batch process

One of the major fears customers have when they initially start journaling is that their batch processes may run for much longer with journaling enabled than it currently does without journaling. For this reason, we tested Banesco's end-of-day batch process both with and without journaling enabled.

This application runs the bank's end-of-the day process which is executed every night. In 1998, Banesco started implementing a high availability solution using software provided by one of the HABP solution providers. The HA solution required that certain database tables had to be journaled and over a period of time, more database tables needed to be journaled as requirements changed.

This caused the end-of-day batch application to run for longer than what Banesco could tolerate. Clearly both hardware and software tuning were required. The first time the batch process was executed with journaling activated on all the database tables, the total processing time increased from seven hours to 17 hours. This batch run had been a major concern to Banesco for many years and after some benchmark tests were performed at IBM in Rochester, Minnesota, USA, certain recommendations were made that would reduce the runtime of this application. The recommendations included hardware upgrades, journal tuning and application redesign.

By implementing each of these suggested recommendations, Banesco has managed to reduce the overall runtime of the application to only two hours and 34 minutes. That's a whole lot better than 17 hours and illustrates the magnitude of improvement that is often possible by adopting the practices outlined in this redbook. Take note of the fact that the resulting tuned time with journaling enabled was even smaller than the original seven hours without journaling. Some of the suggested recommendations which led to this dramatic performance improvement will be described throughout this redbook.

The end-of-day routine executes more than 400 application programs and updates more than 100 of the 2000 journaled tables present in this shop. Banesco has 2000 journaled tables of which 100 are heavily modified, all sent to the same journal, which grows by 15GB during a full two hour and 34 minute batch run.

The application programs are primarily written in RPG, with some embedded SQL statements and they perform tasks typical to a banking environment, for example calculating interest on savings and checking accounts, updating account balances, and so on. Those batch jobs also populate datamarts which are used in a data warehousing environment. None of the application programs that are executed as part of the end-of-day routine uses commitment control.

The end-of-day batch run typically generates in excess of 16 million journal entries in a little over two hours. One particular program, which forms part of the end-of-day routine, generates 33% of the total amount of journal entries. We will focus on this program in greater detail during some of our test runs.

2.2.2 Retail store application

We also wanted to investigate the effect of heavy journaling in an interactive or transactional based environment. To do this, we used a retail store application. This application simulates interactive user transactions by executing multiple instances of the application program, updating a set of database tables and indexes. The application program is written in the C programming language, with embedded SQL statements which produce the database activity.

The database tables include transactional tables (product sold, refunds issued), which are populated throughout the run and other inventory tables which are updated as the transactions occur. The application also makes use of data areas to maintain a transaction ID which is shared between all the jobs that use the same set of data.

In order to simulate, for the sake of our investigations, a ramped increase in the amount of transactional input, a number of these application environments are set up with multiple jobs producing transactions within each set of data. These jobs are then allowed to run for a fixed period of time. Data is collected during and after these runs to determine how many transactions were completed for each interval. The data sets are completely destroyed and recreated before each new iteration. The number of sets of data, as well as the number of jobs producing transactions, are increased each time in order to increase the workload during each iteration. In effect, we were able to simulate interactive activity at more and more stores all serviced by the same iSeries server, thereby ramping up our measured interactive workload.



Part 2

Performance considerations

Journaling has an impact on system performance. In this part we will provide you with an understanding of the factors which influence journal performance on the iSeries and we will help you identify the choices that you can employ to minimize this impact:

- ▶ Different hardware choices
- ▶ Journaling tuning choices
- ▶ Application tuning choices



Hardware choices

The hardware configuration of your iSeries directly influences the impact that journaling has on the performance of your system.

In order to demonstrate this impact, we conducted a series of tests using a variety of different hardware configurations. The following sections describe each test scenario in detail and show the measured performance during each of these tests. We also explain the reasons behind the measured performance impact of each of the test scenarios.

3.1 Factors to be investigated

We attempted to seek the answers to the following questions:

- ▶ Which factors (CPU consumption, elapsed time, disk traffic) are likely to be most strongly influenced by the presence of journaling?
- ▶ Which disk protection scheme (mirroring, RAID5) provides the best journal performance?
- ▶ How much performance benefit does a journal derive from the presence of private disk arms set aside in a user ASP?
- ▶ What is the effect on journal performance of disk rotation speed (7.2K rpm versus 10K rpm)?
- ▶ What is the effect on journal performance of IOA write cache?

3.2 The testing scenarios

The primary objective of the tests described in the testing scenarios below was to measure what performance impact journaling has on the iSeries server under different hardware configuration options.

In order to do this, we ran a series of comparative tests, using an actual customer production database and application programs on the iSeries server described in 2.1, “The hardware” on page 28. The applications are described in 2.2, “Application environments” on page 51.

For each of the following scenarios, the customer application programs, database tables and indexes were all created in the system auxiliary storage pool (ASP). The journal receivers, however, were not.

In each journal testing scenario, we used a different hardware configuration while maintaining the same OS/400 journal settings.

In all the testing scenarios, with the exception of Scenario 1 — no journaling (no journaling at all), we had the system wide access path recovery time objective set to 70 minutes as shown in Figure 3-1.

```

                                Edit Recovery for Access Paths
                                11/20/01 18:23:21
Estimated system access path recovery time . . . :      4 Minutes
Total disk storage used . . . . . :      .571 MB
% of disk storage used . . . . . :      .000

Type changes, press Enter.
System access path recovery time . . .  70 *SYSDFT, *NONE, *MIN,
                                           *OFF, Recovery time

-----Access Path Recovery Time-----
ASP   Target (Minutes)   Estimated (Minutes)   Megabytes   ASP %
1      *NONE              4                   .286       .000
2      *NONE              0                   .114       .000
3      *NONE              0                   .081       .000
4      *NONE              0                   .090       .000

Bottom

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel

```

Figure 3-1 Enabling Edit Recovery for Access Paths (EDTRCYAP)

This setting activates automatic system managed access path protection (SMAPP). SMAPP is a background housekeeping system function that automatically determines which access paths to protect, in order to reduce the amount of time required to perform an initial program load (IPL) in the event of an abnormal system end. This is achieved by implicit journaling of access paths as soon as an access path becomes exposed due to changes to an underlying table or physical file. The system continually evaluates the exposure of access paths and adjusts the amount of implicit journaling it performs in the background in order to meet the specified system access path recovery time objective. By selecting a value of 70 minutes for our scenarios we were being moderately aggressive. Some customers elect to establish even more aggressive targets, some may set this target as low as 10 minutes. The more aggressive the target, the more journaling overhead will ensue.

For more information on SMAPP, refer to the *iSeries Backup and Recovery Guide*, SC41-5304, or refer to the following URL:

<http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/books/c4153045.pdf>

Each testing scenario also uses a journal recovery ratio (JORECRA) setting of 50,000. This value can be adjusted by calling the API **QJOCHRV**. This influences how aggressively the background SLIC sweeper tasks will flush modified database tables and indexes from main memory while journaling is enabled. For more information on JORECRA, refer to 4.2.2, “Journal recovery ratio (JORECRA)” on page 87.

For each test, we also enabled the symmetric multiprocessing (SMP) feature as shown in Figure 3-2.

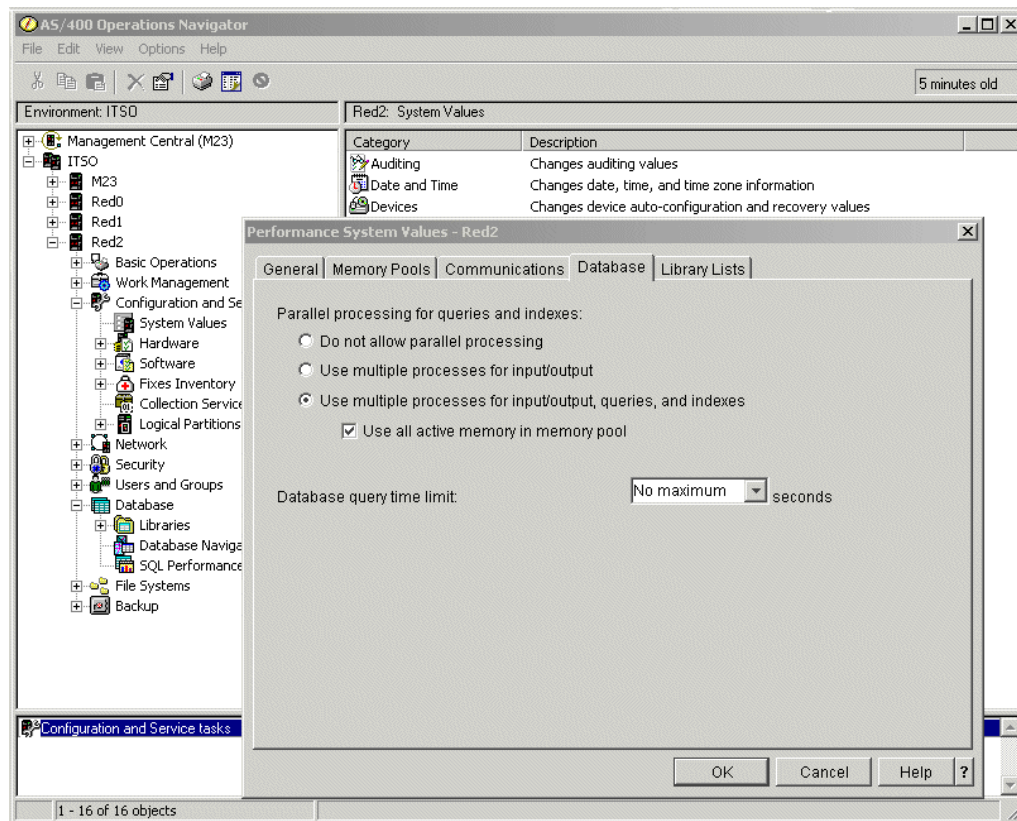


Figure 3-2 Symmetric Multiprocessing (SMP)

Our interest in selecting an aggressive SMP setting for this application is driven by the fact that selecting such an SMP setting will induce OS/400 to perform some of our database index maintenance operations in parallel thereby generating SMAPP induced journal entries in parallel as well. We deliberately wanted to maximize such parallelism so as to increase the journal efficiency. A similar SMP setting may be attractive for your largest batch jobs.

With SMP enabled, both the central processing unit (CPU) and the Input/Output (I/O) processing units (disk) are shared by multiple background SLIC tasks which will maintain each of the access paths comprising our indexes in parallel. This form of SMP allows multiple database operations such as access path maintenance and journaling to take place simultaneously on multiple processors. Thus a single request by your application job to add 100 new rows to an SQL table and to similarly maintain the affected indexes is farmed out to dozens of SLIC SMP tasks which perform these index and journal operations in parallel. To achieve this substantial parallelism, an individual request to add multiple rows to a table and correspondingly add a set of keys to each access path is split into many smaller sub-tasks. Each sub-task runs independently on a separate processor. Once the sub-tasks complete their mission the results of each sub-task are combined to form the complete index maintenance request. The more indexes you have defined over your table, the more this technique pays dividends. Actual CPU parallelism requires a system with multiple processors. SMP parallelism requires the installation of the DB2 Symmetric Multiprocessing for OS/400 system feature.

The above settings allow the index maintenance microcode to employ SMP parallel processing to maintain the access paths. We used this SMP setting because it helps use the underlying hardware more efficiently.

Scenario 2 — journaling in the system ASP

In this testing scenario we enabled journaling on all of the banking application's database tables.

We created both the journal and journal receivers referenced by these database tables in the system auxiliary storage pool (ASP). The library containing the database tables and indexes also resides in the system ASP. Figure 3-4 shows the hardware configuration of the system ASP.

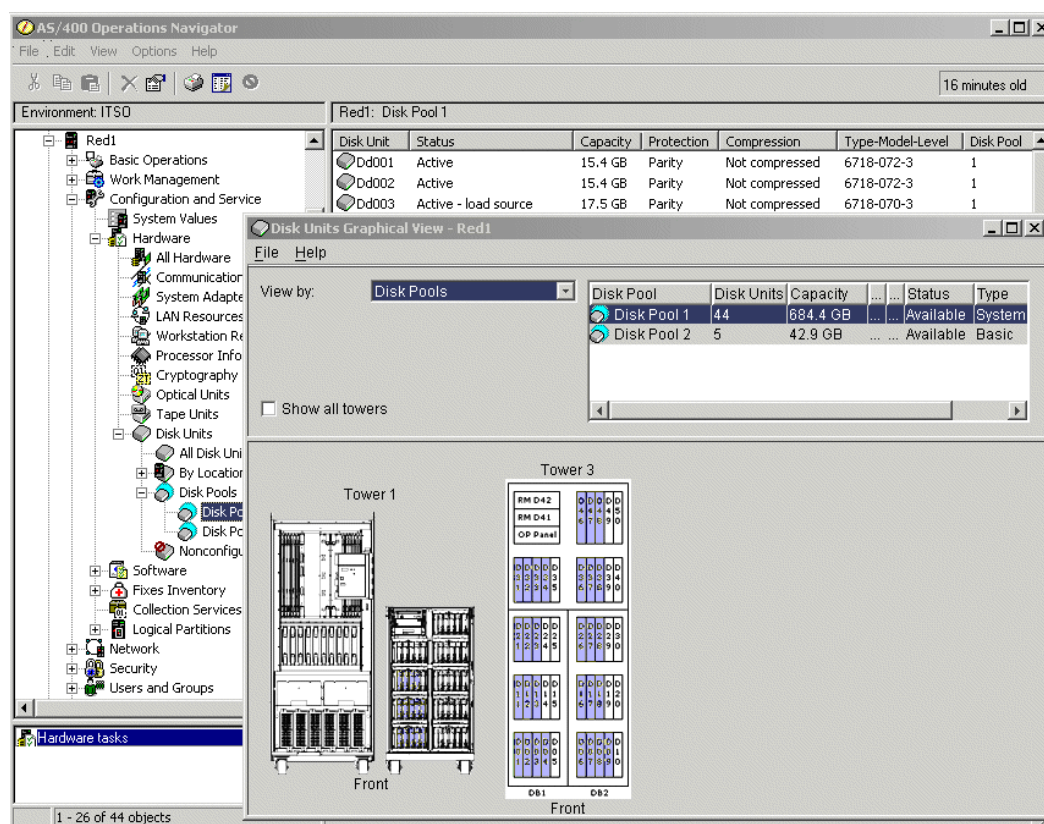


Figure 3-4 System ASP configuration

This figure shows that the system ASP has RAID-5 protection enabled. It consists of 44 disk units, each with a storage capacity of 17.5GB and a rotational speed of 10,000 revolutions per minute (rpm).

For this scenario, the journal and the attached journal receiver are also in the system ASP.

After executing the same set of application programs that we used in the test described in “Scenario 1 — no journaling” on page 61, we compared the runtime and performance statistics collected during each of these two test scenarios.

Scenario 3 — journaling on a RAID-5 protected user ASP

This testing scenario is similar to the one described in “Scenario 2 — journaling in the system ASP” on page 62. However, in this case, our journal receivers resided in a user auxiliary storage pool (ASP), rather than in the system ASP. Figure 3-5 shows the configuration of this user ASP.

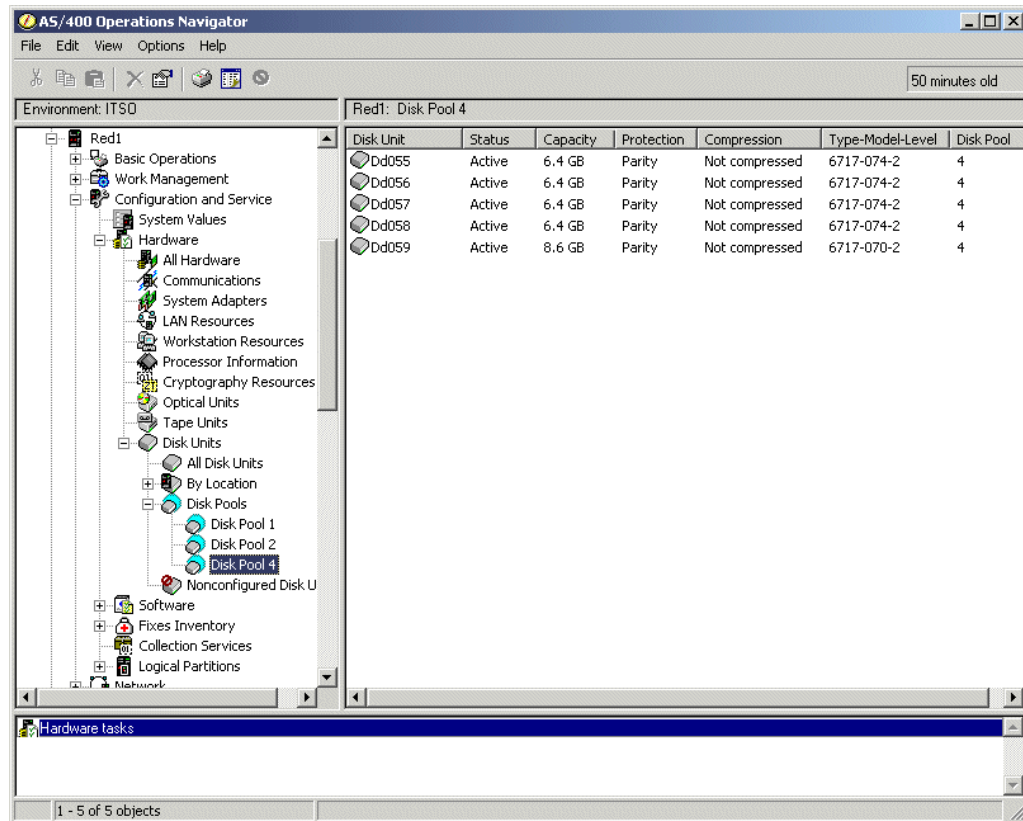


Figure 3-5 RAID-5 protected user ASP configuration

The figure shows that this user ASP had RAID-5 protection enabled. It consisted of five disk units, each with a storage capacity of 8.6GB and a rotational speed of 10,000 revolutions per minute (rpm).

Note: Since the five disk drives are RAID protected the available space is 80% of the total space. If you sum up the available space it adds up to 34.2GB which is 80% of 43GB.

We executed the same set of application programs that were used in the previous test scenarios and compared the runtime and performance statistics.

Scenario 4 — journaling on a disk level mirrored user ASP

In this testing scenario we created the journal receivers in a different user auxiliary storage pool (ASP). Figure 3-6 below shows the configuration of this user ASP.

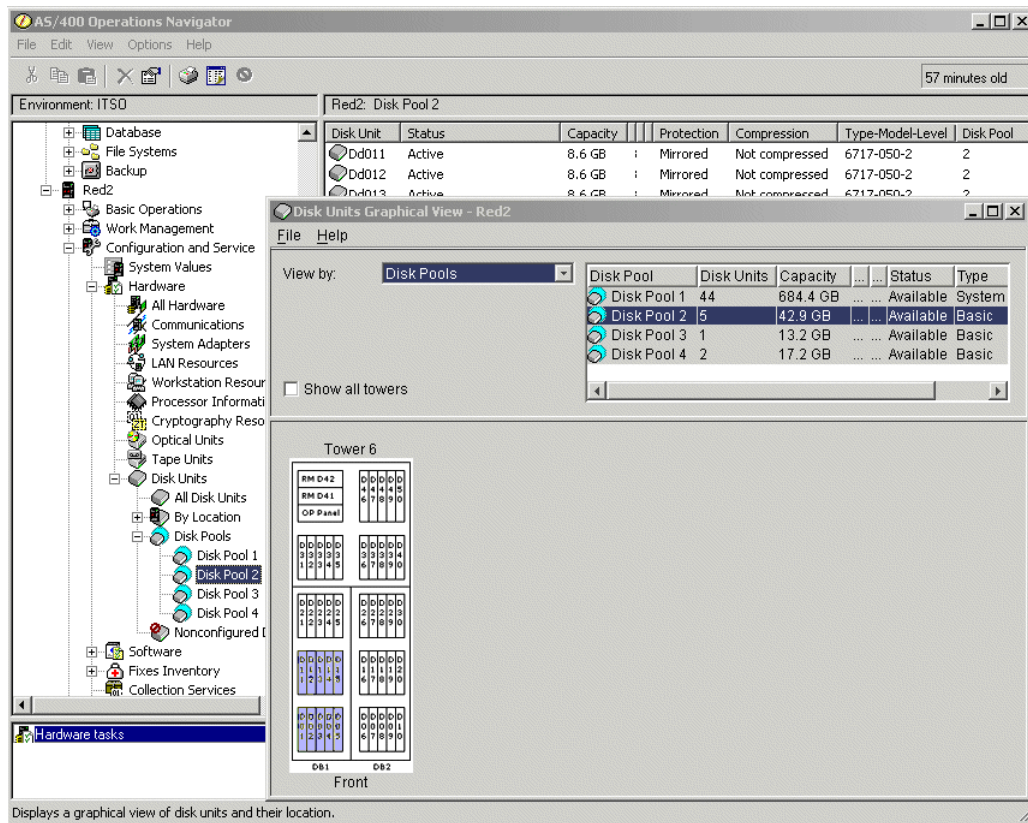


Figure 3-6 Configuration of user ASP mirrored at the disk level

This user ASP consists of ten mirrored disks, each with a storage capacity of 8.6GB and a rotational speed of 10,000 revolutions per minute (rpm). These disk are all located in the same physical hardware tower (disk-level mirroring).

We again executed the same set of application programs as used in the previous test scenarios and compared the runtime and performance statistics.

Scenario 5 — journaling on slower RAIDed disk drives

In this test, we wanted to determine what the impact of slower disk units would have on journaling. We therefore selected a user auxiliary storage pool (ASP) which was configured with slower disk units in which to create the journal receivers. Figure 3-7 shows the configuration of this user ASP.

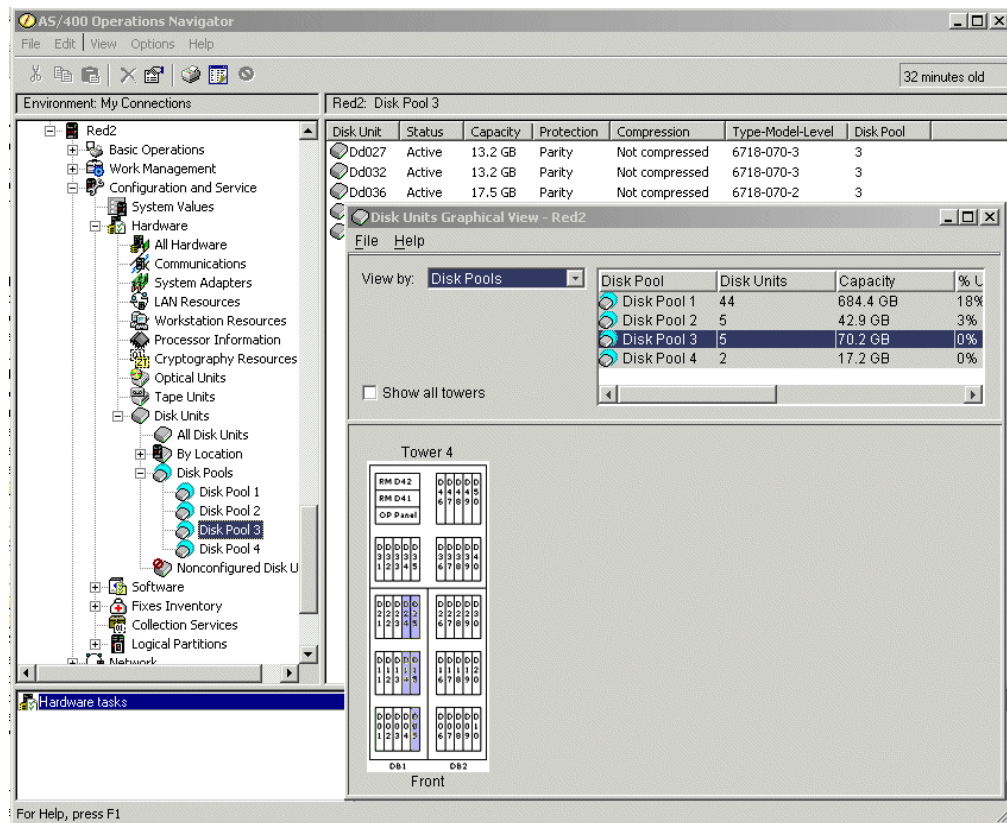


Figure 3-7 User ASP with slower disk units

The above figure shows that this user ASP consists of five disks in a RAID-5 configuration, each with a storage capacity of 17GB and a rotational speed of 7 200 revolutions per minute (rpm).

We executed the same application program and compared the runtime and performance statistics.

3.3 Test results and findings

This section highlights the findings of the test results collected during the hardware test scenarios described in the previous section.

Note: The average size journal entry produced on behalf of the rows of our database tables for the above tests was 575 bytes. Obviously a journal receiver is also likely to house operating system provided journal entries such as SMAPP induced entries which tend to be much wider. This value was determined using the OS/400 command **DSPJRN** with output format Type1.

Elapsed runtime

Figure 3-8 shows a graph of the overall elapsed runtime measured during each of the test scenarios.

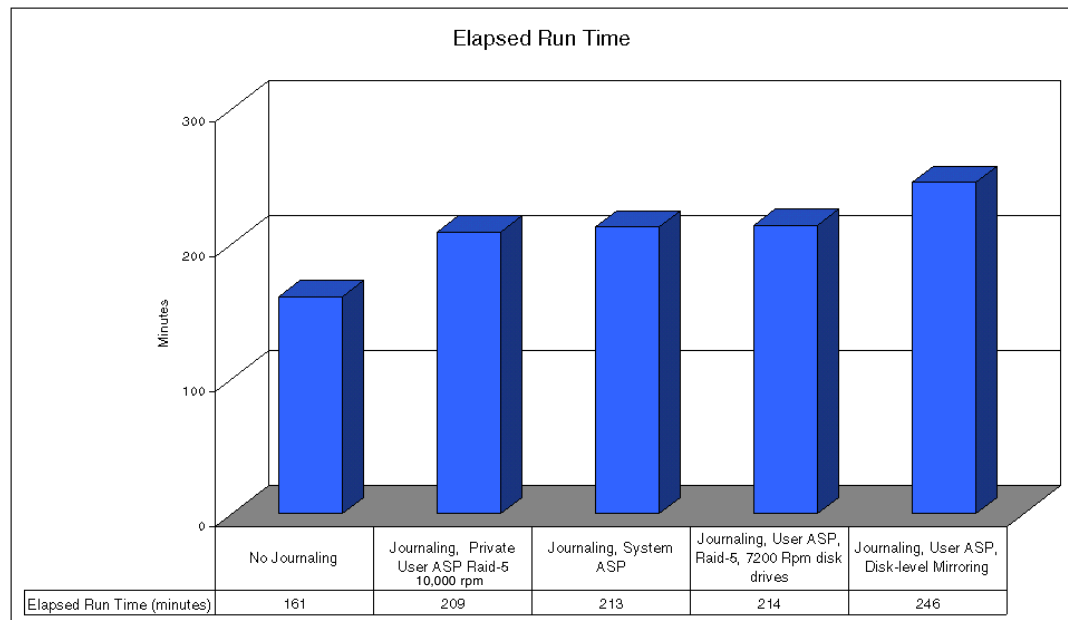


Figure 3-8 Elapsed runtime

Figure 3-8 illustrates the effect untuned journaling and hardware choices can have on the overall runtime of batch applications.

Figure 3-9 illustrates more clearly the percentage difference measured between the different test scenarios.

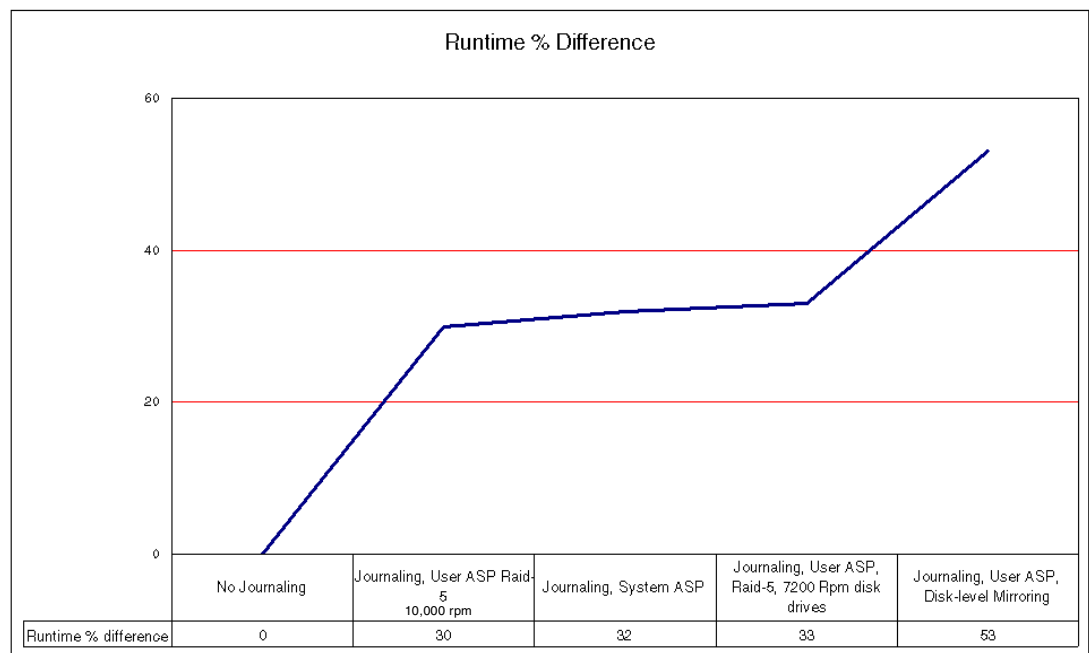


Figure 3-9 Runtime percentage variance

Figure 3-9 shows that we measured between 30% and 53% penalty in elapsed runtime when activating journaling on the application's database tables as compared to a non-journaling environment. While performance penalties this high are often the initial experience of customers whose hardware, applications, and journal parameters have not been optimally tuned, we'll see later that it's often possible to achieve journal overhead much lower than these values. We deliberately left other factors untuned during these tests so that we could fully examine the disk related impact.

We found less impact when the journal receivers were located in a private parity-protected user ASP than when it was located in the parity protected system ASP. One of the main reasons for this is due to the fact that there is less disk arm contention in a private user ASP. In general, the disk arms in the system ASP have to perform general system management tasks. Also, the user application programs and database tables resided in the system ASP and hence contended for those same disk arms. Creating the journal receivers in the system ASP therefore added an extra burden on these disk arms. In our particular test we did not attempt to simulate the extra system ASP disk traffic of both reads and writes which most busy systems would be likely to experience as a result of all the "other work" such as queries that might be going on simultaneously. As a consequence, our test may give the system ASP an unfair advantage and not portray the full benefit associated with the use of private disk arms via a user ASP. Creating the journal receivers in a system ASP that is kept very busy with other tasks, for example an ASP that contains a database that is heavily updated or queried, could produce very different results than what we observed. The busier your system disk drives, the more performance benefit you'll probably realize by configuring a user ASP to house your journal receivers.

Figure 3-9 shows that for this particular environment and application the rotation speed of the disk drives did not have a significant effect on the overall run time. The elapsed run time varied by only about 1% when the journal receivers were placed in an ASP with disk units rotating at 7200 revolutions per minute (rpm) compared to using an ASP with disk units rotating at 10,000 rpm. Clearly, our Banking environment was too tame to push the disks hard. Other customers have reported as much as a 30% journal performance benefit by installing faster disk drives. The effect of the disk speed would undoubtedly become more evident if your application generated enough journaling traffic to overrun the disk write cache. This could typically occur if you use some of the older models of IOAs with less write cache. Our write cache was quite ample for this application.

Creating the journal receivers in a mirrored ASP shows a marked increase in the overall run time. One of the reasons for this result is due to the fact that twice as many physical writes occur when an entry is written to a journal receiver located in a mirrored ASP. Although somewhat overlapped in time, two physical images of the same journal receiver entry are written to disk — one to each disk unit in the mirrored pair. This results in extra overhead, more CPU pathlength and double the traffic through the IOA write cache and subsequently more time is required to perform the physical write operation. Another contributing factor which allowed our RAID-5 configurations to consistently outperform our mirrored configurations is the improved performance and efficiency of modern IOAs, which speeds up the parity checking dramatically in a RAID-5 configuration. Whereas mirrored disk writes tax the main CPU, RAIDed disk writes tax the CPU within the IOP.

Rule of thumb: If you have sufficient write cache configured, our tests suggest that RAID protected disk drives will probably give better journal performance characteristics than mirrored disk units.

Although not illustrated in Figure 3-9, we also conducted tests to determine the difference in elapsed runtime between locating the journal receivers in a user ASP configured with disk-level mirroring and a user ASP configured with tower-level mirroring.

Our tests revealed no significant difference in the overall runtime between these two types of mirrored ASPs. One of the explanations to this finding is that the banking application we selected produced an insufficient amount of journal traffic for the amount of write cache, combined with the high performance IOAs. In short, we simply didn't push the disks and write cache hard enough.

Rule of thumb: RAID-protected disk drives generally absorb high levels of journal traffic better than mirrored disk drives until you reach the point where your IOA write cache is getting overrun.

However, the high level of redundancy obtained by employing mirroring, especially tower-level mirroring, provides very real high availability benefits and should under no circumstances be ignored. For more information on high availability, refer to the following URL:

<http://www.redbooks.ibm.com/redpapers/pdfs/redp0111.pdf>

CPU consumption

Figure 3-10 below shows some of the measured results pertaining to CPU consumption.

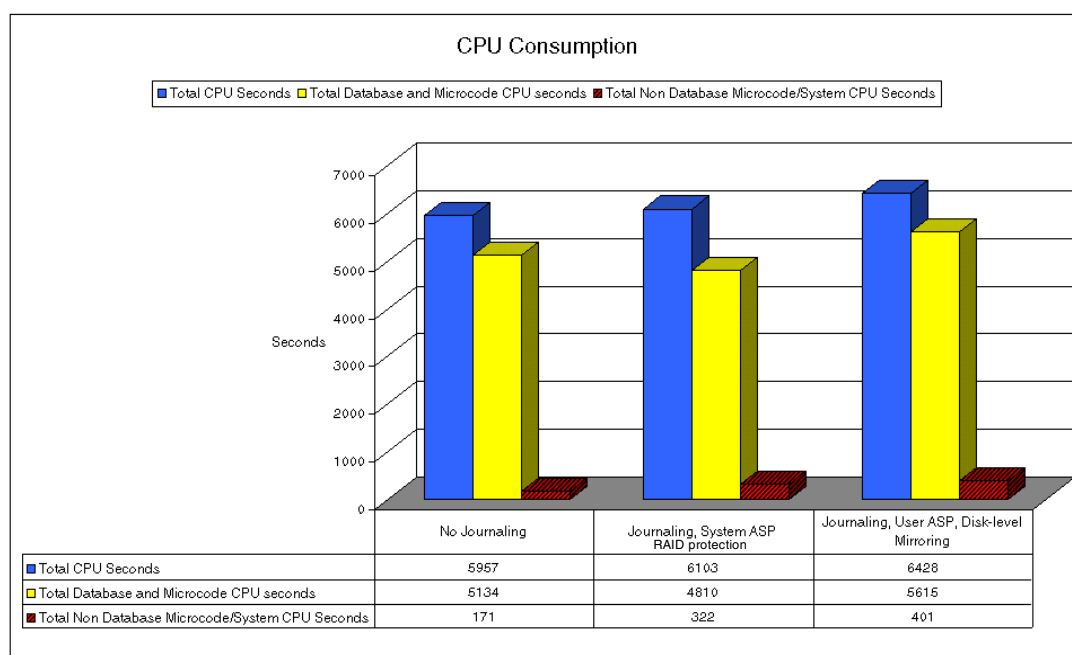


Figure 3-10 CPU consumption

It is evident from Figure 3-10 that journaling requires a bit more CPU resources than a non-journalled environment. CPU cycles are spent building journal entries and the impact of this will be greater in systems that do not have much CPU to spare. The good news is that the extra CPU consumed is only about 2.4 percent.

We also see that more CPU time is spent on journaling when the journal receivers are located in a mirrored user ASP. More CPU time is required to build the duplicate disk images of the journal receivers in the mirrored ASP. The memory frames of the disk images are prepared by the CPU and not by the IOA, resulting in a storage management overhead.

We further notice a reduction in the database and microcode flavored CPU time consumed when journaling is activated in the system ASP. In a journaled environment, some of the processing burden is transferred from pure database tasks to journal management tasks.

Clearly our 2.4% increase in CPU consumption is far less than the 30% increase in elapsed time that we documented on Figure 3-9. So if CPU isn't the primary contributor to increased elapsed time we need to look elsewhere. Let's do so by examining the disk write operations.

Journal deposits and database writes

Figure 3-11 shows the measured comparisons between disk write activity in a journaled and non-journaled environment.

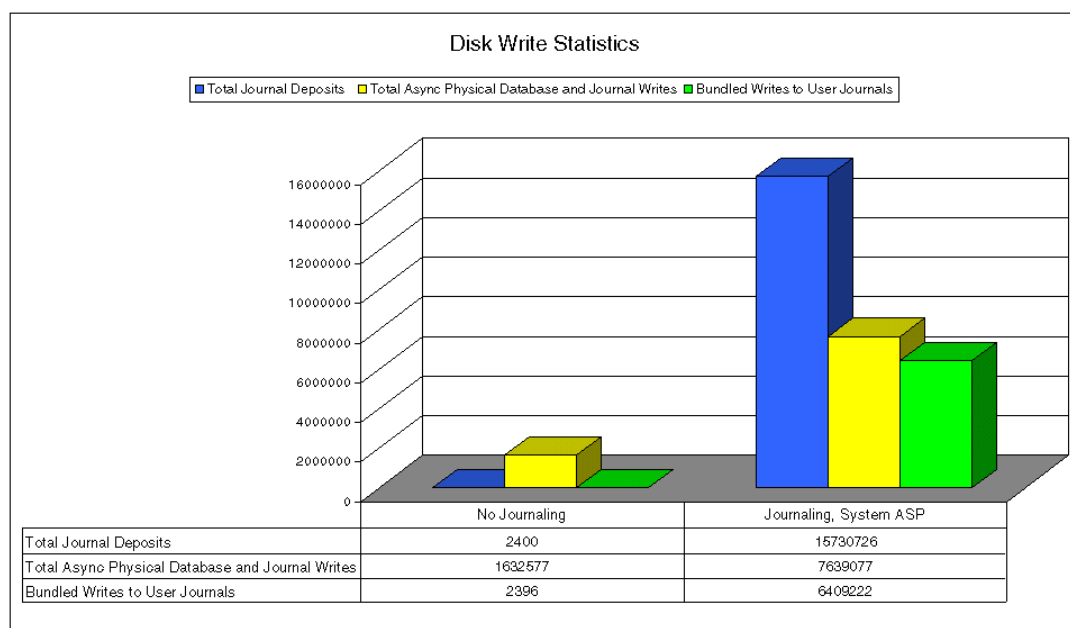


Figure 3-11 Disk write statistics

As you can see from Figure 3-11, activating journaling (not surprisingly) causes a significant increase in the amount of total physical disk writes. The system protects certain system provided objects via default journaling. Therefore, a very small amount of journal activity can be seen even in the scenario which has no explicit application database files journaled.

The graph also shows the high degree of bundled writes that occur in a journaling environment. The basic concept of bundling is a technique used by the journal manager whereby the journaling support allows its data pages to be buffered in main memory. If the journal volume is high enough the system licensed internal code (SLIC) groups together multiple journal entries into a single memory buffer of up to 128KB in size before issuing a write request. This group of journal entries sent to the disk in unison is known as a bundle. This results in fewer, but larger and more efficient disk write operations. This is one of the ways in which the system automatically attempts to reduce the performance impact associated with journaling. In the above test environment, the bundling rate would have improved further if the application would have been able to generate journal entries at a faster rate.

In this particular testing environment, while the total quantity of journaled writes produced was high, the number of disk arms available to service these writes were sufficient, and therefore the average disk arm utilization never exceeded three percent. In addition, 99 percent of all disk write requests were fast writes. The term fast write implies that the IOA write cache had room to absorb a fresh journal write request without having to flush an earlier journal page image to make room for the new arrival. These results reinforce the efficiency and adequacy of the 26MB write cache we had configured for this test, coupled with the fact that the user application could not produce journal entries any faster. In short, we had plenty of write cache configured and our performance benefited as a result.

Database writes and I/O waits

Figure 3-12 illustrates the impact that journaling has on logical database writes and I/O wait situations.

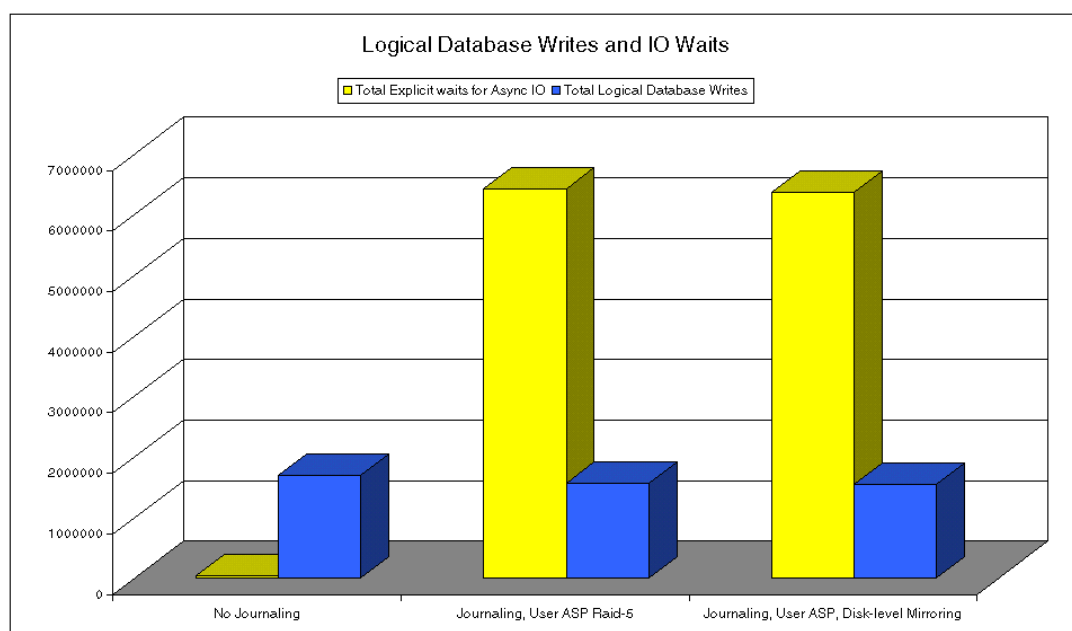


Figure 3-12 Logical database writes and I/O waits

The term logical database writes refers to the number of times your application program empties its ODP buffer and delivers a set of new database rows to the OS/400 layer who in turn hands this whole set of new rows to the SLIC microcode. Each delete or update row request is counted as a separate logical database write and results in a corresponding journal entry. Each delete or update row request similarly empties the contents of the ODP buffer. The same can not be said for blocks of added rows. Depending on the blocking factor (NBRRCDs) you specified on your OVRDBF command, hundreds of new added rows may be buffered in your ODP and accumulated into a single logical database write request.

Therefore, what is counted by the performance tools as a single logical database write may actually generate hundreds of journal entries. Since some of the banking application batch jobs deliberately selected a substantial blocking factor along with SEQONLY(*YES) so as to achieve optimized performance, the number of IO waits shown in Figure 3-12 exceeds the number of logical database writes. This is *not* to say that it exceeds the number of rows added to our table, which it clearly does not. Instead, what we can observe is that the number of logical database writes shown in this figure are consistent from one run to the next.

The logical database writes indicated on the previous graph influences the quantity of the journal entries produced. The explicit waits for asynchronous I/O shown is typical of the behavior of journaling. As mentioned previously, when journaling is active, the asynchronous writes of the journal entries are bundled. Although journal management issues an asynchronous write operation for each journal bundle written, the storage management job that is triggered to write a corresponding set of memory frames to disk requires confirmation that the write operation was in fact successful before it will release the memory frame to another process. This results in an I/O wait situation. Hence a wait ensues for each bundle written. The ratio between total journal entries produced and number of I/O waits shows the main memory caching efficiency via bundling.

Similarly, the non-journaling environment performs almost no bundled writes and therefore shows very few I/O wait situations.

You will also notice from the above that the logical database writes and I/O wait situations in the environment with mirrored protection are virtually the same compared to those measured in the environment with parity protection. Although the total amount of bytes written to disk is significantly more in the mirrored environment (two copies instead of one), the journal bundling behavior is unaffected. The bundled writes to the two mirrored disk units occur in parallel, thereby not increasing the amount of logical database writes. Therefore, the quantity of I/O wait situations are virtually the same as in the environment with parity protection.

This simply reinforces our observation that mirroring consumes only one frame of main memory for each two disk sectors written.

Journal entries per second

Figure 3-13 shows some of the results collected pertaining to the journal entry deposit rate during this banking run.

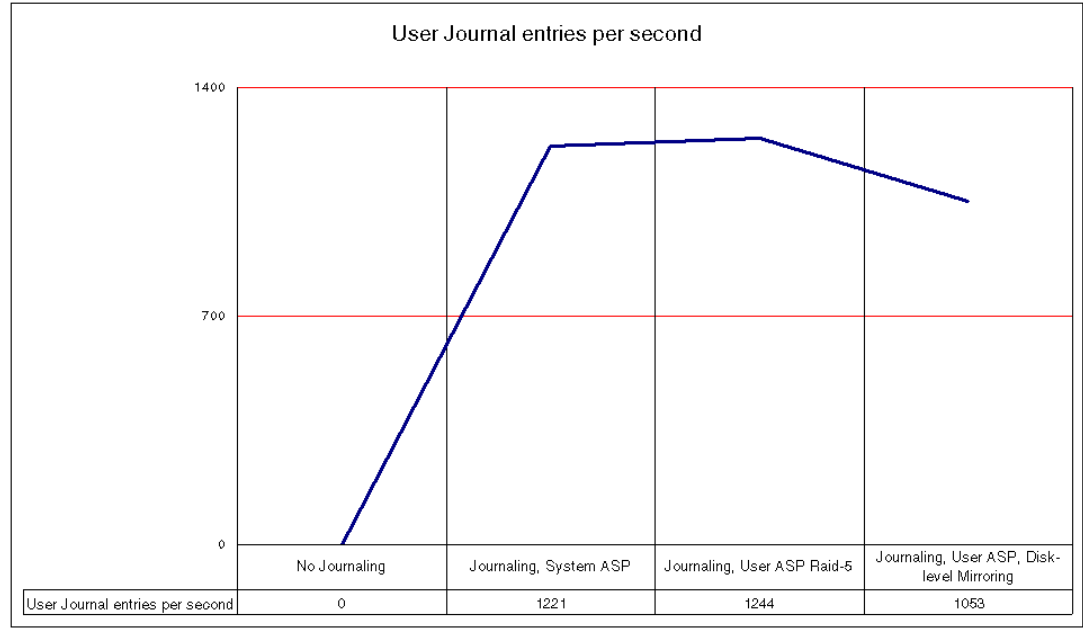


Figure 3-13 Journal entries produced per second

As you can see from the above, we measured a slightly better rate of journal entry deposits in the parity protected user ASP, compared to the system ASP. This is largely due to the disk arms in the system ASP being busier than those in the user ASP. In situations where the application generates journal entries at an even faster rate or where the system ASP is much more busy, than was true for our environment, this difference could potentially become more noticeable.

The rate of journal entry deposits our banking application was able to sustain in the mirrored user ASP is also lower compared to the RAID protected ASPs. Both, the disk arms in the mirrored ASP as well as the associated write cache have to service twice as much traffic, as can be seen in Figure 3-14.

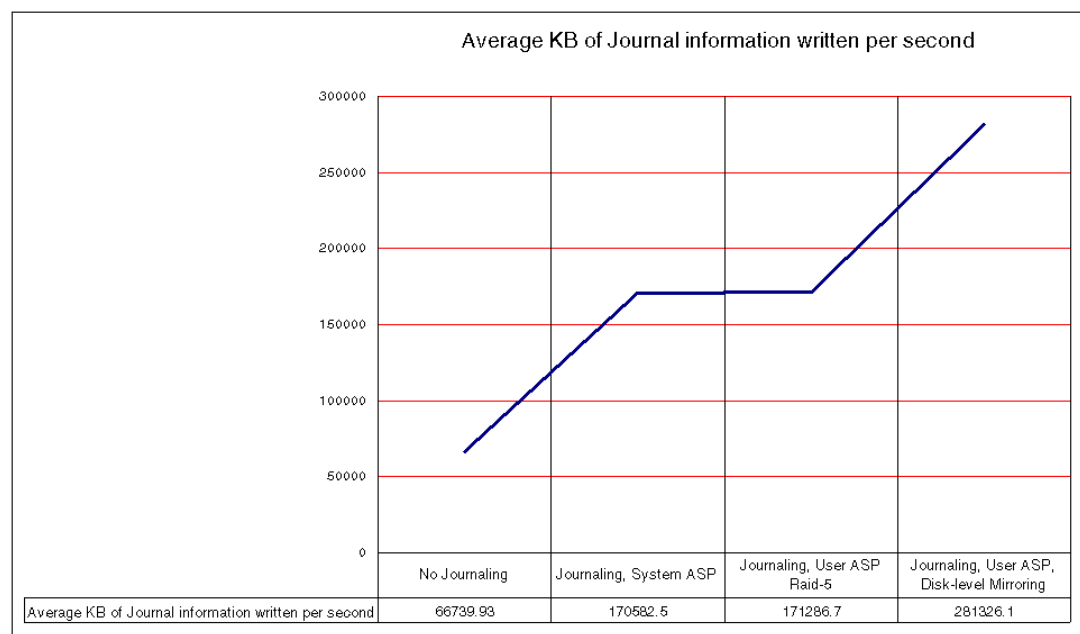


Figure 3-14 Average kilobytes per second written

The average kilobyte per second traffic to the mirrored user ASP is substantially higher than in the parity protected ASPs. Two images of the same journal entry are written in the mirrored user ASP. The system CPU waits for the slower of these two parallel writes, thereby resulting in a decline in the rate of journal entry deposits.

3.4 Conclusions

The hardware configuration of your iSeries server often has a profound impact on journaling performance. The following are based on the conclusions derived from the tests described in the previous sections and will give you some insight into hardware factors that you should consider in a journaling environment.

Auxiliary storage pool configuration

When deciding whether to locate your journal receiver in the system ASP, a user ASP with parity protection or a mirrored user ASP, you should take the following into account:

- How busy are the disks in your system ASP?

As mentioned in the previous section, the amount of work that your system ASP has to do could increase disk arm contention and thereby potentially result in a substantial decline in journal entry throughput if you create your journal receivers in an already busy system ASP. The biggest single cause of this slowdown would be disk arm contention.

- What journaling benefit does a user ASP have?

A journal receiver resides in a library which in turn resides in an ASP. When you create the journal receiver, the system automatically spreads the journal receiver across a limited quantity of disk arms in the selected ASP. As journal receiver entries are constructed in main memory they are bundled together and written to the disk arms servicing the relevant journal receiver in a “round robin” fashion. This method ensures that each journal bundle is written to a different disk arm than the preceding journal bundle. However, when your journal traffic arrival rate is high enough a second bundle may be ready to leave main memory before the first bundle has reached disk. In that circumstance, the presence of secondary disk arms can be helpful. If you create the journal receiver in a user ASP and only one journal receiver is active in that ASP at one time, this round-robin method reduces the risk of disk arm contention when multiple concurrent writes to the same journal receiver occurs.

Rule of thumb: Don't allow the number of active, aggressively populated journals housed in the same User ASP to exceed two times the number of disk arms configured.

- Does the number of journals sending entries to the same user ASP make a difference?

Figure 3-15 illustrates the effect of having more than one journal receiver in the same ASP.

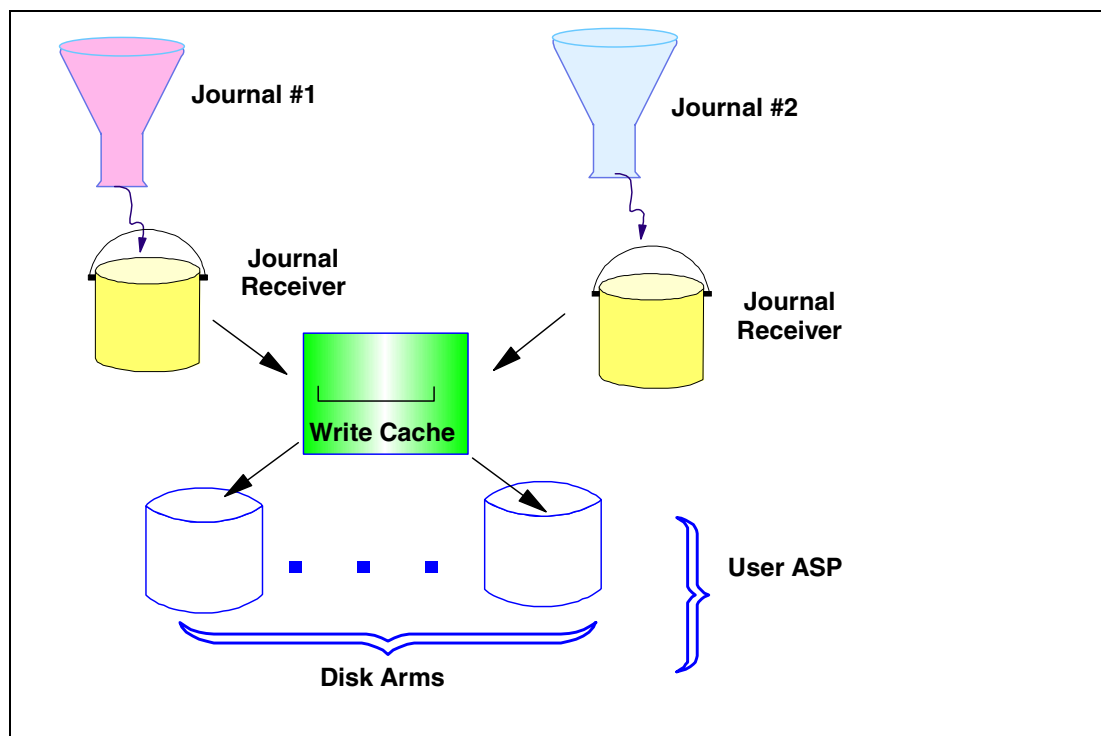


Figure 3-15 Journal receivers sharing the same ASP

We can see from the above illustration that the likelihood of both disk arm contention and write cache overflow increase when multiple journal receivers attached to distinct journals share the same user ASP.

Giving sufficient write cache, some modest sharing is practical. A safe rule of thumb is probably to avoid having the number of distinct journals populating the same User ASP exceed two-times the number of disk arms configured in that ASP.

► Will a dedicated user ASP make a difference?

When an ASP is dedicated to journaling and only one journal receiver therein is active at any given point in time, the disk arm is continually positioned at the correct track until, of course, the disk cylinder becomes full. This eliminates seek time and other disk arm movement and thereby speeds up the write operation (It also helps the write cache empty faster). If, however, the ASP is shared amongst multiple journal receivers, database tables, indexes and application programs, the same disk arms are servicing all these functions simultaneously and hence arm movement is likely to become a consideration. The disk arms remain positioned at the location of the last write operation and thus each arm has to reposition itself for every new write operation. If database and journal writes are interspersed, each subsequent write has a high likelihood of discovering that the write operation is slowed by the need for an arm reposition operation. This will impact the performance of all the applications and functions that share the same ASP.

However, while we've illustrated that private disk arms may help reduce disk arm contention and seek time, it can be counterproductive to place a journal receiver on a user ASP if that user ASP houses only one disk arm. The lack of a second disk arm thwarts the ability of the underlying journal microcode to issue overlapped parallel disk bundle writes in unison. Hence any user ASP housing journal receivers should contain a minimum of two disk arms unless your total journal traffic is modest. The presence of a second disk arm is even more crucial for optimal performance if you've specified `RCVSILOPT(*RMVINTENT)` on your `CHGJRN` command.

Disk protection method

Each write request to a parity protected ASP ultimately requires four disk input/output operations, two overlapped read operations followed by two overlapped write operations. Also, the actual data resides close to the outer edge of the disk unit while the parity information is located towards the center of the disk unit. Therefore, when a write operation is issued, the disk arm spends seek time to position itself at the data portion after which a neighboring disk in the same parity set has to seek towards the center of the disk to locate the parity information. This could potentially cause slow response time in a journaling environment, especially if the disks were rather empty. As a consequence of this observation, some shops elect to deliberately keep RAID-5 protected disks in user ASPs far from empty, especially if their journal traffic is high and their quantity of IOA write cache is undersized. Such a practice helps to mitigate the potential RAID-5 performance overhead by reducing the average disk arm seek distance.

A mirrored ASP, on the other hand, has to perform two physical write operations of the same data, one to each arm. However, as mentioned previously, the two write operations occur simultaneously. The write operation is considered successful only after both write operations signal their respective completion. Therefore, the duration of the write operation is equal to the duration of the write request to the slowest of the two disk units. In an environment where different disk technologies coexist, performance may be impacted by slower disk units.

Our test results in the previous section showed an increase in the overall runtime of the application when we created the journal receivers in a mirrored ASP. The main contributors to the fact that our RAID-5 configuration performed better than our mirrored configuration was the presence of the new high performance IOAs which help reduce the overhead of RAID-5 protection and our use of a very large write cache. This caused a dramatic reduction in the traditional overhead observed in a parity protected ASP. The following section regarding Disk write cache explains the concepts and effect of the IOA write cache in greater detail.

Despite the performance impacts, the high level of redundancy that mirroring provides still makes it a very attractive option for ensuring high availability and should not be overlooked. Depending on your requirements, you can implement mirroring at the disk level up to the tower level.

For more information on high availability, refer to the following URL:

<http://www.redbooks.ibm.com/redpapers/pdfs/redp0111.pdf>

Disk write cache

As indicated by our test results, the amount of write cache servicing the disk units housing your journal receiver has a direct influence on the performance of journaling.

Figure 3-16 shows a basic overview of the behavior of the disk write cache.

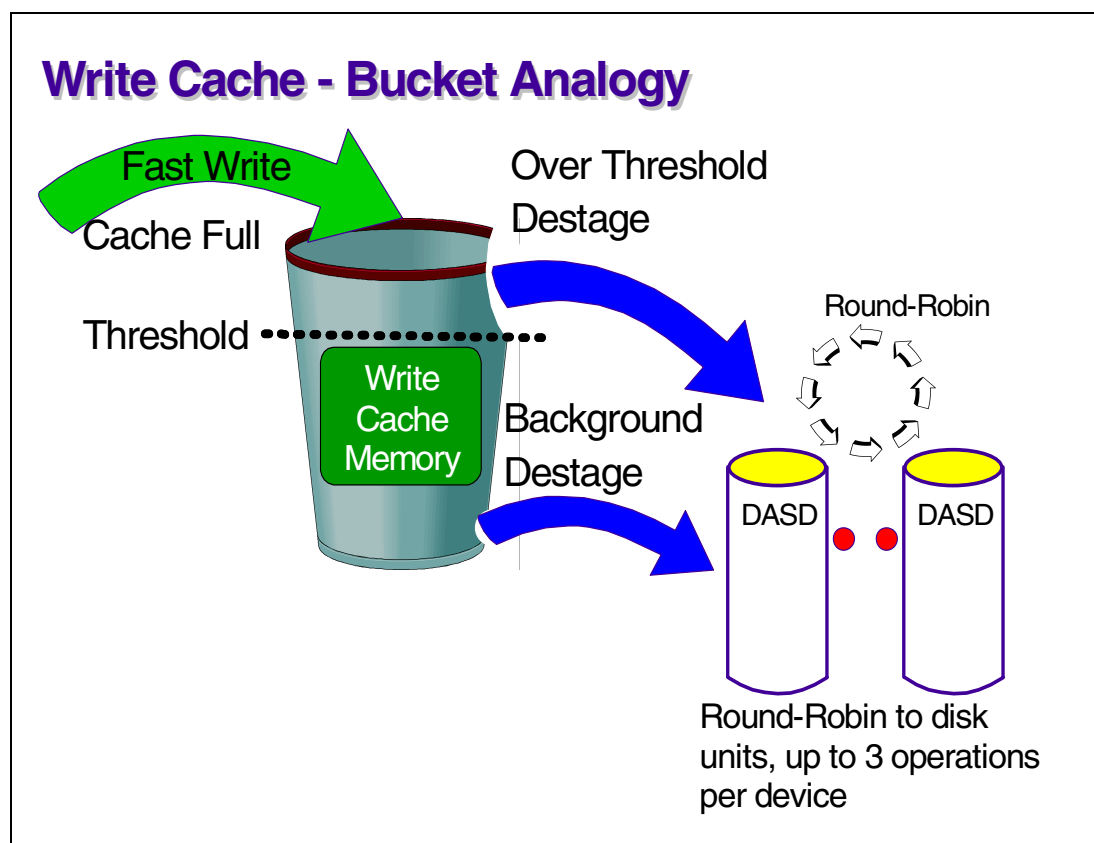


Figure 3-16 Disk write cache overview

Figure 3-16 shows the general flow of data once a journaled disk write request has been issued. The journal data is written from main memory to the write cache memory. Provided the IOA write cache isn't already full, this is known as a fast write operation. Subsequent write requests also deposit journal data into the same write cache memory until the write cache memory threshold is reached. At the same time, the write cache memory is constantly being emptied, aging out older pages as described under, "Coalescence of write cache contents" on page 16, writing the journal data to the physical disk units. This process is known as a destage operation and continuously occurs in the background.

If journal traffic is so aggressive that the write cache memory is being filled faster than the background destage operation can write the data to the disk units, the cache memory will continue to fill up until the cache full state is reached. Once the cache full state value has been reached, subsequent journal write requests will be forced to destage earlier fast writes to the disk units immediately. Such write requests are no longer fast writes and thus result in slower write operations thereby impacting journal performance. Fast writes can take as little as 0.5 ms to service while slow writes and the related destaging can consume anywhere between 6 ms and 100 ms apiece depending on the rotation speed of your disk arms and depending on how busy the arms are servicing READ requests. Also, it is possible for disk arm contention to start occurring at this stage. This is the reason that configuring sufficient write cache for your journal is such an important consideration.

Once sufficient data has been destaged to the disk units and the contents of the write cache memory falls below the threshold, fast writes will start to occur once again.

The bundling behavior associated with journal tends to deliver large quantities of journal bytes to this IOA write cache in bursts. The IOA write cache algorithm attempts to smooth out these bursts and keep the underlying disk drives busy at an even pace.

We can therefore see from the above that a larger write cache can accommodate larger bursts of sustained journal traffic. Thus larger write caches will dramatically decrease the possibility of non-fast journal write operations and unwanted destaging operations occurring. The newest technology IOAs and faster spinning disk drive units provide even faster destaging of data to the disk units, thereby reducing the buildup of journal data in the IOA cache. The more disk arms you configure in your user ASP housing your journal receivers, the more write cache you'll need, this combination of additional write cache and more disk arms for destaging gives significant journal bandwidth improvement.

Rule of thumb: Generally, no single 26 MB write cache IOA should be asked to service more than 10 disk arms when heavy journaling is present and no older vintage 4 MB write cache IOA should be asked to service more than 4 or 5 disk arms.

Figure 3-17 below shows the evolution of IOA write cache memory sizes.

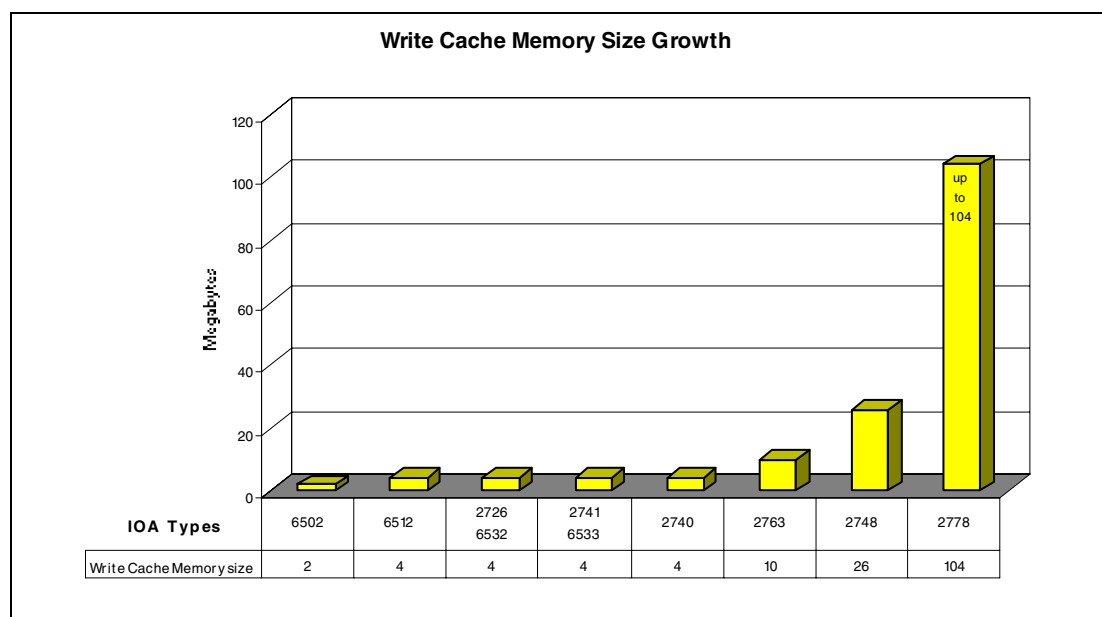


Figure 3-17 Write cache memory sizes

As you can see from Figure 3-17, disk write cache has become larger with each new IOA type. The latest of today's IOAs, the model 2778, behaves as though it has up to 104 megabytes of disk write cache.

If you want to find out what model of IOP and IOA technology your system has type the following command:

```
WRKHDWRSC TYPE(*STG)
```

For more information on the different IOA types and models, as well as available disk write cache sizes available, refer to the *iSeries Handbook*, GA19-5486

To prove how important the write cache is to the performance of journaling we ran a series of tests with one of Banesco's end of day programs on a smaller iSeries system. In this case we used a model 170 iSeries with an IOA that had only 4MB of write cache. In some of our runs we even disabled the write cache.

Table 3-1 Disabling write cache

Test	Elapsed time
Program without journaling but with write cache (4MB)	78 min
Program without journaling and no-write cache	86 min
Program with journaling and write-cache (4MB)	108 min
Program with journaling and no-write cache	1010 min

Figure 3-18 illustrates the difference of having and not having write cache in a non journaled environment.

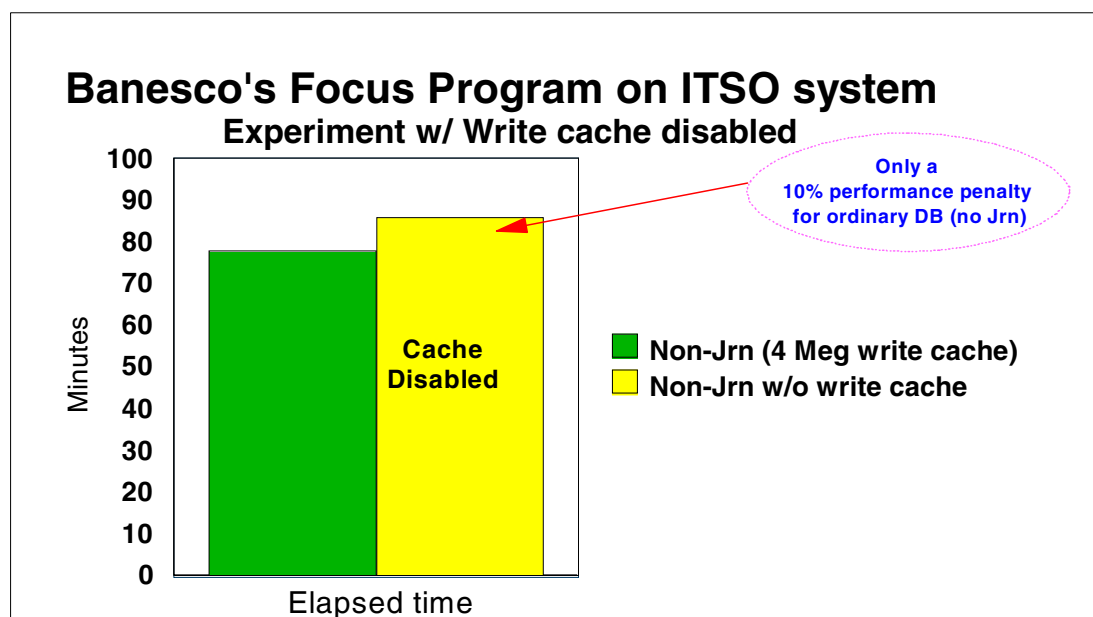


Figure 3-18 No journaling with and without write cache

Conclusion

Pure database operations rarely wait for disk writes to complete, hence they are not as sensitive to the presence or absence of write cache as we'll see for journal. That's one of the reasons that smaller systems and older vintage systems with limited write cache which ran well prior to journaling see such a dramatic slow down when an HA BP enables journaling for the first time. This same observation probably also suggests that the Journal Caching PRPQ is likely to show even more benefit on the smaller/slower machines with limited IOA write cache than on the faster machines.

Figure 3-19 illustrates the difference of having journaling versus not having journaling with write cache.

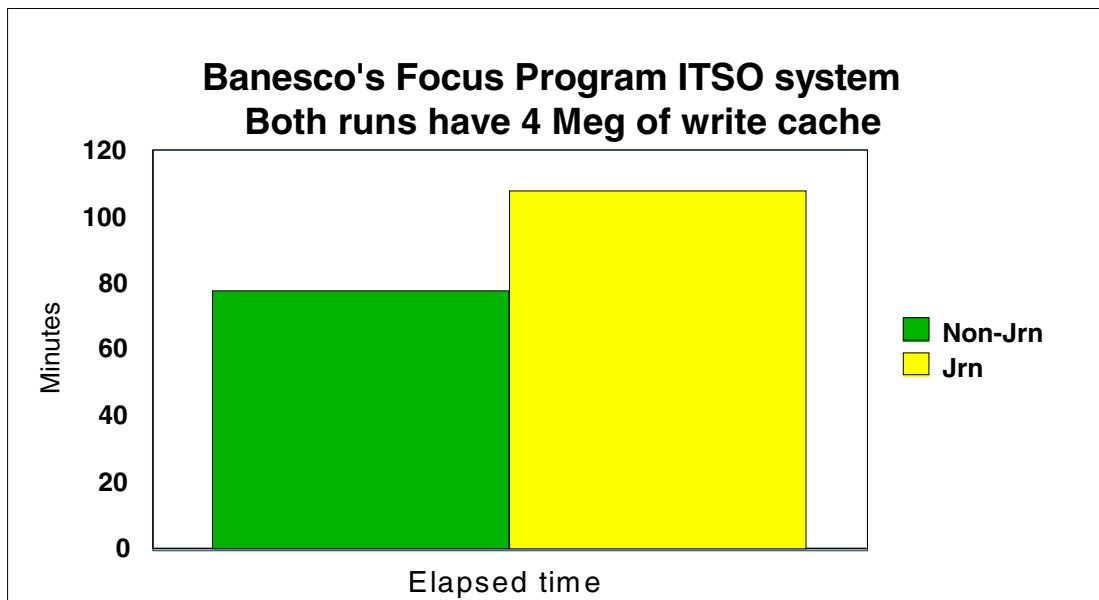


Figure 3-19 Journaling penalty on smaller system even with write cache

Conclusion

Unlike pure database, journaled operations do wait for disk writes to complete. On systems with inadequate write cache, the performance penalty (38%) can be substantial. The solution is generally either software caching via the Batch Journal Caching PRPQ or stepping up to more modern IOA technology which houses a larger physical hardware write cache.

We've seen that having an inadequate quantity of write cache can harm performance. While 4MB was clearly insufficient for the banking application, it was our hunch that we were at least getting some benefit from even this modest amount of write cache. Our question was, "How much benefit are we deriving?" We put this question to the test by deliberately disabling the write cache for our next run. By doing so we were able to investigate what older vintage machines serviced by no write cache whatsoever might experience. The results were dramatic.

Figure 3-20 illustrates the impact of not having any write cache in a journal intensive environment.

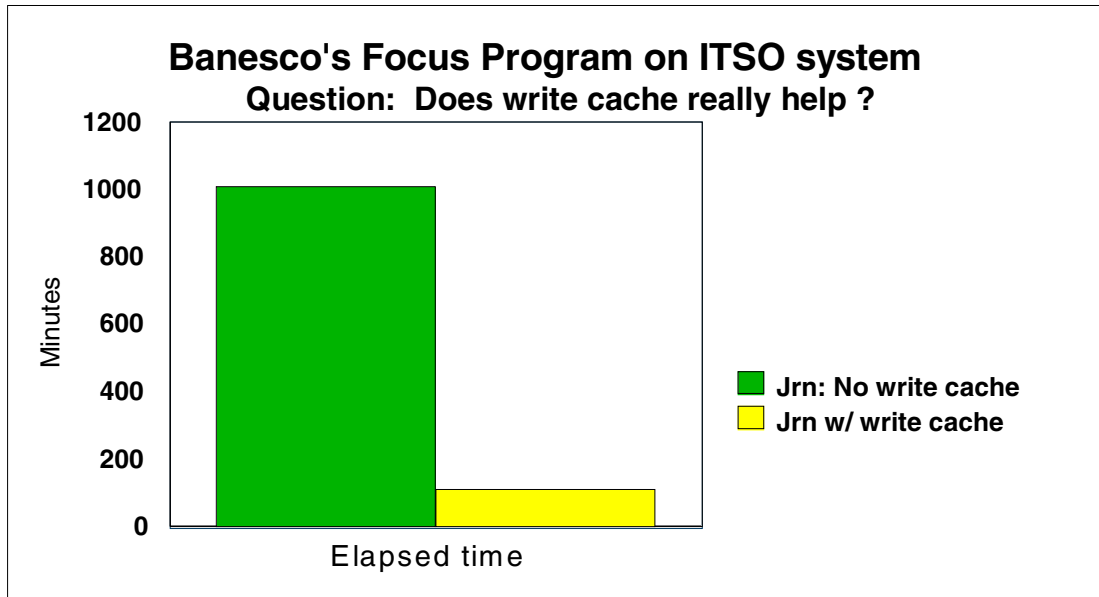


Figure 3-20 Journaling with no write cache

Conclusion

Supplying sufficient IOA write cache is probably the most influential hardware decision you can make in order to reduce journal overhead. It's more important than RAID versus Mirroring, more important than user ASP versus SYS ASP, more important than selecting the optimal number of disk arms, more important than 7.2K rpm versus 10K rpm disk drive speed.

Rule of thumb:

- ▶ Get your hands on the maximum amount of IOA write cache you can afford.
- ▶ If your environment is “journal intensive”, avoid configuring the maximum 15 allowable drives per IOA, because journal can swamp the write cache.
- ▶ Limit yourself to four disks per IOA if you’ve got only 4MB.
- ▶ Limit yourself to 10 disks per IOA if you’ve got 26MB of write cache.



Journaling tuning

As mentioned previously, journaling has an impact on the performance of your system.

In this section, we will provide you with insight into some of the options and attributes of both journals and journal receivers that you can exploit in order to further reduce this impact.

4.1 System Managed Access Path Protection (SMAPP)

SMAPP is a system function that runs in the background under control of the System Licensed Internal Code (SLIC).

Its function is to manage the protection of access paths (SQL indexes, referential constraint indexes, and keyed logical files) on your system in order to provide faster database recovery during an IPL in the unlikely event of an abnormal system shutdown.

SMAPP makes use of implicit journaling to ensure that it can recover in-flux access paths in the event of a system failure. It accomplishes this objective by dynamically selecting which access paths to protect via journaling. Notice that not all access paths get such journal protection. Depending on the amount of recovery time you specify, SMAPP will be more (or less) aggressive in protecting exposed access paths. It periodically evaluates, based on the number of changed access paths and their corresponding sizes, how long it would take to rebuild each of these access paths, generally targeting only the larger of the exposed access paths.

Besides continually re-evaluating which access paths need to be protected, some background SMAPP jobs retune the amount of protection required, others start and stop implicit journaling of certain access paths and still others sweep changed index data from main memory by initiating disk writes. All of these jobs aggressively consume CPU cycles and you'll therefore want to select the proper balance between recovery times versus run-time performance overhead in order to maintain system performance at an acceptable level.

Rule of thumb: Identify your largest and most critical access paths. Issue STRJRNAP for these indexes. This is more efficient than having SMAPP make this decision repeatedly for you.

SMAPP settings

The SMAPP target recovery value is intended to act as a cap on the maximum amount of IPL/Recovery time that should be devoted to repairing invalidated access paths. It is set to a default value each time you install a new release of OS/400. For those who first installed OS/400 beginning with V5R1 this default was 90 minutes. A trend has been established with nearly every release having a default SMAPP setting more aggressive than its predecessor. You can modify this recovery time using the OS/400 command **EDTRCYAP**. Figure 4-1 shows how to use this command.

```

                                Edit Recovery for Access Paths
                                RED1
                                11/17/01 18:08:37
Estimated system access path recovery time . . . :          3 Minutes
Total disk storage used . . . . . :          .514 MB
% of disk storage used . . . . . :          .000

Type changes, press Enter.
  System access path recovery time . . .   70 *SYSDFLT, *NONE, *MIN,
                                              *OFF, Recovery time

-----Access Path Recovery Time-----
ASP      Target (Minutes) Estimated (Minutes)
  1          *NONE              3
  2          *NONE              0
  4          *NONE              0
                                Megabytes      ASP %
                                .286 .000
                                .114 .000
                                .114 .000
                                Bottom

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel
  
```

Figure 4-1 Edit Recovery Access Paths (EDTRCYAP)

The possible values for system access path recovery time are as follows:

***SYSDFT** Specifying *SYSDFT sets the default value for the current release (90 minutes for V5R1).

***NONE** When this setting is specified SMAPP does not perform any access path protection (that is, there is no upper limit on how long an abnormal IPL can take). SMAPP will however continue to monitor your IPL exposure in terms of estimated access path recovery time, so that you can view your current exposure on this screen. When this value is selected, no implicit journaling takes place.

***MIN** This choice sets the access path recovery objective to the minimum recovery time that can be achieved for your system and as a consequence nearly all of your access paths will be implicitly journaled.

***OFF** With this setting SMAPP does not perform any access path protection. SMAPP will also not monitor your exposure in terms of estimated access path recovery time. (We strongly recommend against ever selecting this setting unless directed to do so by the development laboratory since such a setting disables any opportunity to monitor or determine access path rebuild time exposure).

target-values	You can specify a value between 10 and 1440 minutes. The system rounds the value up to the nearest 10-minute boundary.
----------------------	--

Some access paths, however, are never eligible for protection by SMAPP no matter how large they become. These special ineligible access paths include:

- ▶ A database table that has *MAINT(*REBLD)* specified for its access paths
- ▶ Any access path created in the QTEMP library
- ▶ Multi-format access paths whose underlying physical files are journaled to at least two different journals
- ▶ An access path for a database table that has the *FRCACCPH(*YES)* parameter specified
- ▶ Access paths for an SQL encoded vector index
- ▶ Temporary access paths created by Query, SQL or DFU

The two most troubling ineligible categories in this list are *FRCACCPH(*YES)* and multi-format access paths built over two or more underlying physical files where the physical files are indeed journaled but, unfortunately, are not journaled to the same journal. These two cases are especially onerous for good performance and hence should be avoided. If you still employ *FRCACCPH* (a very ancient option) we strongly suggest that you turn it off and employ *STRJRNP* instead. If you employ multi-format access paths (those built over two or more physical files) we strongly suggest that you insure both physical files are journaled to the same journal.

Rule of thumb: Replace *FRCACCPH(*YES)* protected access paths with explicitly journaled access paths (via *STRJRNP*).

You can use the OS/400 command **DSPPCYAP** (Display Access Path Recovery) to display your current exposure in terms of access path recovery time. The OS/400 Performance Collector tool in conjunction with the Performance tools licensed program product also provides information about access-path recovery time.

Considerations

As mentioned before, the background SMAPP jobs do consume system resources. The following are a few points that you should be aware of when employing SMAPP:

- ▶ In order to achieve its target recovery time, SMAPP produces additional journal entries, which puts an increased workload on your disk IO processors. This could potentially have an adverse effect on overall disk response time as well as fill your IOA write cache more rapidly.
- ▶ The journal receivers for SMAPP take up additional auxiliary storage. The system creates an internal SMAPP journal and journal receiver for each ASP on your system. These journal receivers are spread across all the arms of an ASP, up to 100 arms. These hidden journals are used only if your access paths are large and the underlying database table is not already journaled.
- ▶ If a database table is explicitly journaled, SMAPP avoids sending the access path related journal entries to the hidden SMAPP journal set aside for this ASP and will, instead, place these hidden internal SMAPP journal entries into the same user-provided journal which is housing your SQL row images. These internal entries could cause your user journal to become very large. To prevent this dramatic increase in size, you should consider specifying the parameter *RCVSIZOPT(*RMVINTENT)* on the *CRTJRN* or *CHGJRN* commands.
- ▶ Your system must be in a restricted state if want to change the SMAPP value from **OFF* to another value which is another good reason to never move into the **OFF* state in the first place.

For more information on SMAPP, refer to the *iSeries Backup and Recovery Guide*, SC41-5304, or the following URL:

<http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/books/c4153045.pdf>

4.2 Journal parameters

There are several parameters on the **CRTJRN** and **CHGJRN** commands that can assist you in reducing the impact of journaling on your iSeries. In this section, we will introduce you to some of the most influential journal parameters.

4.2.1 CRTJRN and CHGJRN parameters

Receiver Size Option (RCVSIZOPT)

As implied by its name, this parameter directly influences the size of your journal receiver and the rate at which it fills.

The following is a brief description of some of this parameter's options:

- ▶ ***MINFIXLEN**: When you specify this value, descriptive information identifying the source of these journal entries such as the job name, user profile information and the application program identity are not deposited as a part of the journal entry. That can be good thing if you're looking for a way to reduce journal overhead. By specifying this option, you can reduce your journal entry size (generally by about 5%) and the quantity of CPU consumed as well. However, you should carefully consider the impact if you require this type of information for other purposes, for example auditing.
- ▶ ***RMVINTENT**: This option allows both the SMAPP-induced entries as well as the index journal entries produced by STRJRNAP to be removed from the journal receiver when they are no longer required by the system thereby saving in some instances substantial space. Also, with this option specified the implicit journal entries deposited by SMAPP are written to a different set of disk arms than those used to house traditional user journal entries. Figure 4-2 depicts how the two types of journal entries are separated when you specify this option.

As you can see from Figure 4-2, although both the traditional user journal receiver entries produced on behalf of your physical files and SQL tables and the SMAPP-generated journal entries produced on behalf of your access paths are written to the same journal receiver in the same user ASP, they are stored on different sets of disk arms only when ***RMVINTENT** is specified.

Provided you have a sufficient quantity of disk arms configured in your ASP this reduces the likelihood of disk arm contention and therefore improves throughput. Such separation is not as practical if your user ASP is configured to house only one disk arm.

Rule of thumb: If your User ASP has at least three disk arms, strongly consider use of ***RMVINTENT**.

Furthermore, this option also reduces the rate of growth of your journal receivers, since SMAPP-generated entries that are no longer required are gradually aged and removed from the journal receiver and the corresponding disk space is freed. This produces an immediate saving of time and backup media when the journal receivers are saved. As Figure 4-2 suggests, it's only the content of the primary disk arms shown on the left, not the arms housing these SMAPP entries shown on the right, which get written to tape.

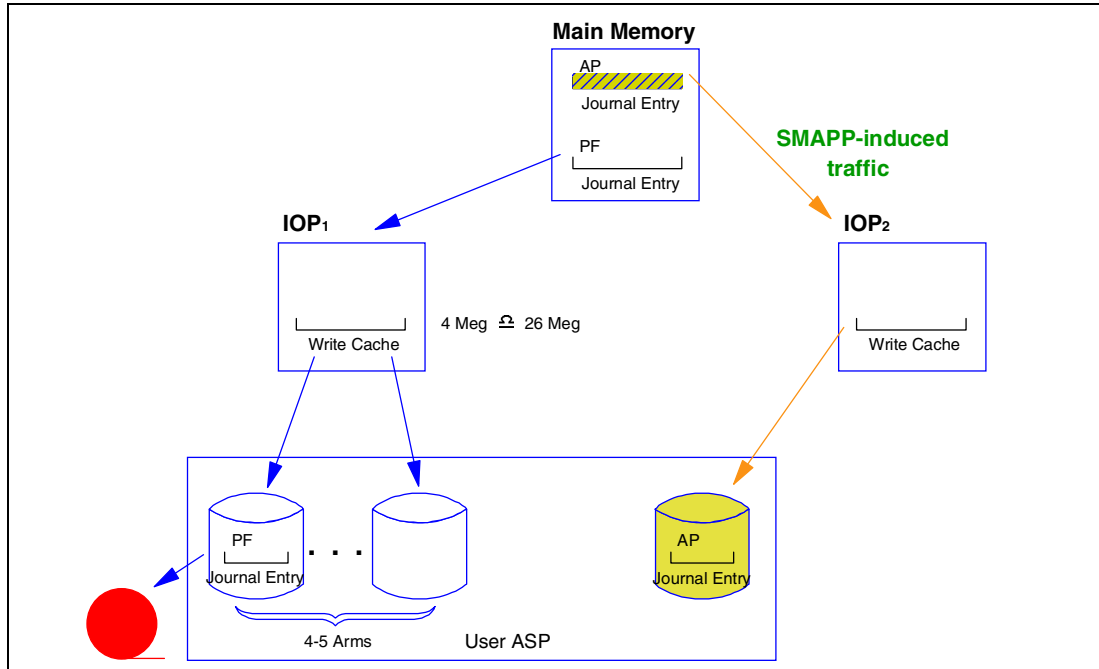


Figure 4-2 Remove internal entries (*RMVINTENT)

You could also benefit from the smaller rate of growth of these journal receivers when applying journal entries to your database via the APYJRNCHG command, because APYJRNCHG only needs to read into memory the contents of the disk arms shown on the left. Furthermore, the communication overhead is also reduced when you transmit these smaller journal receivers across a communication line as part of a remote journaling high availability scheme. Here too it's only the contents of the disks shown on the left which need to be sent to your target machine.

In short, in environments in which large files covered by lots of large access paths are modified, the use of this option can be a space saver which helps speed up a variety of subsequent operations which are sensitive to journal receiver size.

- ***MAXOPT1:** When you specify this value, the journal receiver as a whole can have a maximum size of about 1 terabyte (TB) instead of the traditional maximum capacity of a mere 2GB and it can have a maximum journal entry sequence number of approximately 9,999,999,999 instead of the traditional limit of a mere 2,000,000,000. Similarly, the maximum size of each journal entry which can be deposited is 15,761,440 bytes. Provided you also specify a sufficiently hefty journal threshold, this 1TB maximum size will reduce the frequency with which you must change journal receivers which in turn can lead to improved performance. This is especially attractive if you tend to generate more than 1GB of journal receiver data per hour. The less frequently you must pause to change journal receivers the better your performance.

In addition, *MAXOPT1 also tends to make the duration of a CHGJRN command smaller when attaching a new journal receiver. This benefit is most likely to occur if you are journaling thousands of objects to the same journal.

- ***MAXOPT2:** When you specify this value, the journal receiver can have a maximum size of about 1 terabyte (TB) and a maximum journal entry sequence number of 9,999,999,999 just like the *MAXOPT1 setting, but single journal entries may now be a maximum size of 4,000,000,000 bytes rather than 15,761,440 bytes. This is particularly useful if you are journaling large IFS files or your journal entries contain large objects (LOBs) either of which might exceed 15 MB. Also, this option will further assist you in reducing the

overhead of changing journal receivers if your system generates large volumes of journal entries. If in doubt, there seems to be little if any downside to selecting *MAXOPT2.

Minimize entry-specific data (MINENTDTA)

This parameter (which is new for V5R1) will also assist you in reducing the rate at which your journal receiver grows, thereby reducing the frequency of journal receiver changes, as well as the time required to save your receivers to media or to transmit your journal receivers across a communication line.

For a detailed explanation of this parameter, refer to 1.3.2, “Minimize the Entry-Specific Data parameter” on page 18.

4.2.2 Journal recovery ratio (JORECRA)

The journal recovery ratio is implemented as a count of journal entries to be processed. The ratio establishes an upper limit on the number of journal entries which will need to be examined and applied/replayed if the machine were to terminate abnormally. This recovery ratio controls the aggressiveness of background SLIC tasks which sweep changed journaled object pages (such as SQL rows) out of main memory and onto disk. Set it too small and you end up consuming substantial quantities of CPU, IOA write cache and disk bandwidth sweeping instead of doing useful production work. Set it too high and you risk longer IPL recovery processing. The higher the ratio, the less frequently journaled objects get written and the better the run time performance. The default value is 50,000. That means on average you can expect that an abnormal IPL will have to examine and replay 50,000 journal entries and visit 50,000 SQL rows before the IPL step completes. You can change this value by invoking the Change Recovery Ratio API. For example:

```
Ca11 QJ0CHRV 100000
```

You can specify a recovery ratio ranging from 10000 up to 2 billion.

For most shops who care more about reducing journal performance overhead than about extremely fast IPL recovery (perhaps because they have a second iSeries server as their HA standby machine) and who have a substantial amount of main memory, the 50,000 default is probably too aggressive, thus it may make sense for such shops to bump this recovery ratio up to at least 250,000.

A gradual trickle of changed pages from main memory to disk is far less disruptive to other applications sharing the same disks and IOA than a massive memory flush.

4.2.3 Changing and managing journal receivers

Each time a current attached journal receiver is replaced with an empty journal receiver, all the associated journaled objects must gradually divorce themselves from a dependency on the contents of this former journal receiver. All such dependencies must be curtailed before the journal receiver can be deleted. In order to sever this dependency, the main memory-resident changed pages of each of these journaled objects must be written to disk. Rather than initiate this massive set of disk writes at the time that the CHGJRN ensues and thereby slow down the CHGJRN command, background SLIC jobs gradually sweep these changed pages out of main memory continuously. Thus, little by little your recently changed table rows are written to disk. Once all such rows reach disk, it's safe to delete the corresponding journal receiver. If you were to schedule your own CHGJRN action to swap journal receivers and follow it up immediately with a corresponding attempt to delete this same detached journal receiver, your delete operation would be held up waiting for all the modified pages to reach disk. A wiser choice might be to let the system initiate the change of your journal receivers via the MNGRCV(*SYSTEM) option on CHGJRN. That way the system

is both managing the swap of journal receivers and the sweep of changed pages from memory. Provided that you don't try to delete the journal receiver too quickly after the system senses that the journal receiver has reached its specified size threshold, the ensuing delete operation should complete rather rapidly.

Rule of thumb: Unless your business has a natural lull, let the system manage the receivers for you using the MNGRCV(*SYSTEM) parameter. The receiver must be created with a threshold value of at least 5000KB.

Figure 4-3 illustrates what happens when you change journal receivers.

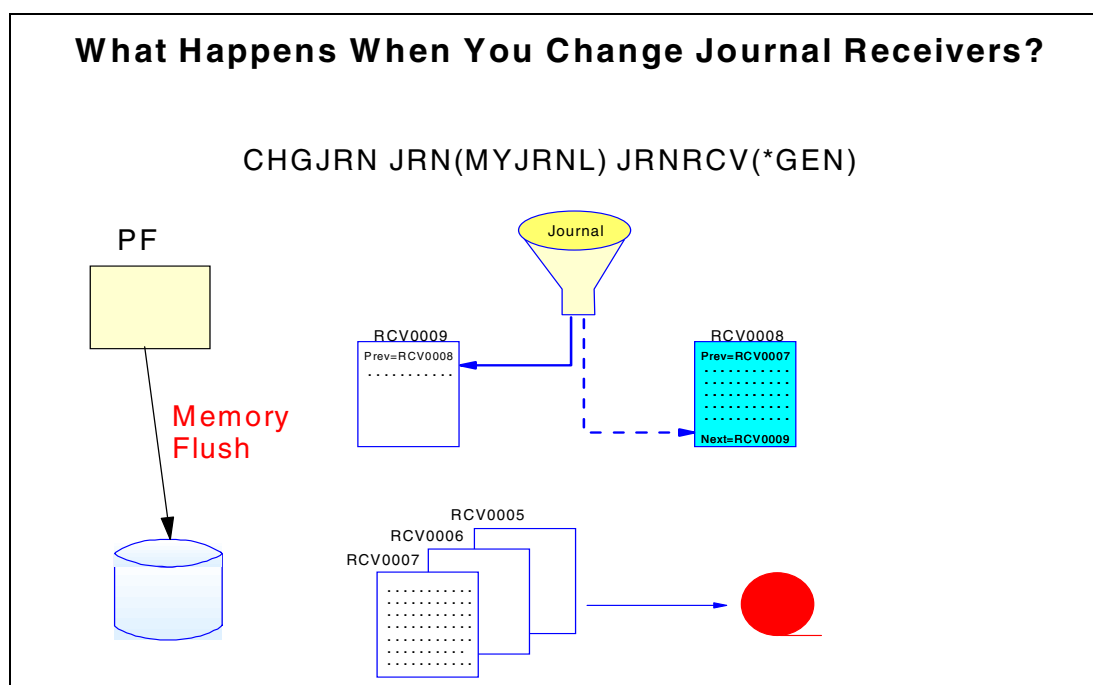


Figure 4-3 Changing journal receivers

Attention: If you do not change receivers promptly and regularly, the user ASP can overflow and it can cause poor performance.

Another important consideration related to the management of your journal receivers is how often should I change them? You get to select both the maximum journal receiver size and a threshold value which alerts you when the receiver is approaching its maximum size. Starting in V4R5 you can elect to move away from the traditional 2GB maximum journal receiver size and step up, instead, to the new 1TB receiver size. When you do so, be sure to remember to similarly increase your threshold to a comparable value if you want to shoot for some capacity value greater than 2GB. By increasing both the threshold and capacity you're decreasing the frequency with which journal receivers need to be swapped and hence reducing the instances of any performance spike associated with the flush which accompanies such a swap. In V4R5 the larger capacity (1TB) journal receiver size is designated by specifying RCVSIZOPT(*MAXOPT1) on your CHGJRN command. For V5R1 you can step up to *MAXOPT2, which not only gives you the 1TB size limit but also allows individual journal entries to be wider when necessary, a need you may face if you elect to store large BLOBs in your database files or SQL tables.

Rule of thumb: If your applications produce a Gig or more of journal traffic per hour you'll probably want to step up to a *MAXOPT1 setting for your journal and increase your receiver threshold.

Note: Most HA BP products provide screens and interfaces by which you can tune most of the journal options. Doing so from within the HA BP product has the advantage that the product knows about the selections you've made and can adapt its behavior accordingly so as to help you achieve the best performance.

4.3 Recovering at a hot site - APYJRNCHG

There are some shops that do not have an HA BP solution. These shops have to rely on the tools provided with the operating system to be able to recover from a system failure. A reasonable question for such sites is how they can speed up this recovery process. One of the CL commands that can help to do this is APYJRNCHG. Figure 4-4 illustrates how APYJRNCHG operates.

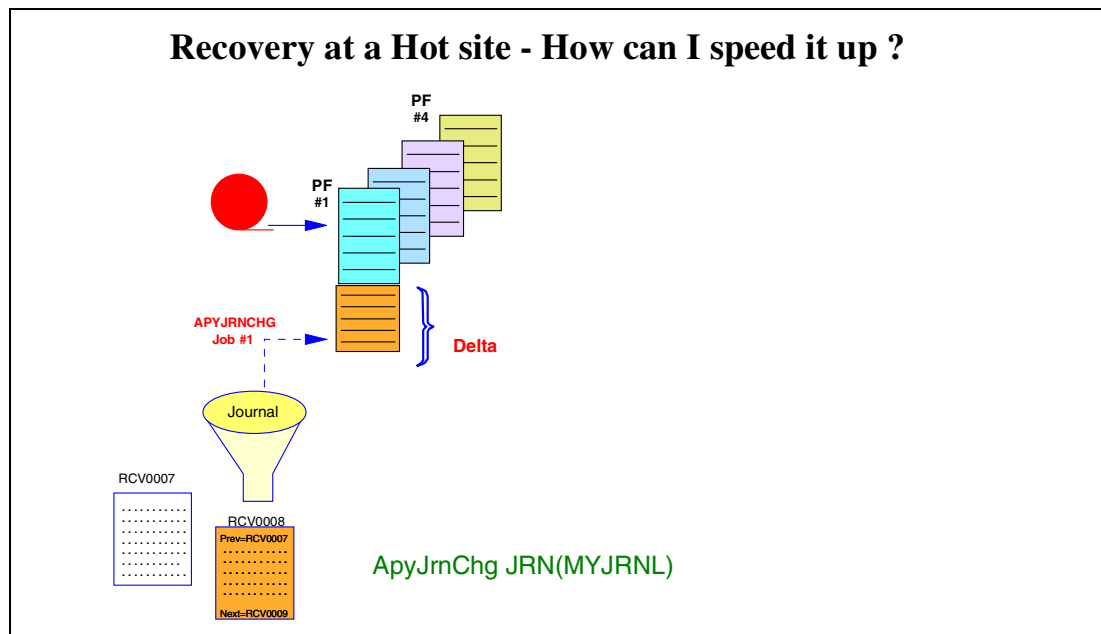


Figure 4-4 Recovery at a hot site - APYJRNCHG

In the next section you will find recommendations concerning this topic.

4.4 The testing environment

The tests described in this section were all conducted at the IBM Teraplex Center, the first eleven scenarios using the actual Banesco database and application programs, and the twelfth scenario using the retail store environment. The hardware configuration of the iSeries server is described in detail in 2.1.1, "Teraplex Center" on page 29 while 2.2.1, "Banesco's end-of-day batch process" on page 52 contains an overview of the Banesco application programs.

4.5 The testing scenarios

The primary objective of the tests described in this section was to evaluate and measure how you can reduce the impact of journaling on your iSeries by setting various journaling parameters and options.

You'll find these scenarios to be remarkably similar to those we used in the last chapter to explore the impact of hardware choices.

In order to measure the impact of each of the journaling parameters individually, we executed a series of tests using the same hardware configuration, applications programs and database, while selectively activating particular journaling parameters during each of the testing scenarios.

In all the testing scenarios, with the exception of Scenario 1 - No journaling, we had the system access path target recovery time, SMAPP, set to 70 minutes.

For more information on SMAPP, refer to the *iSeries Backup and Recovery Guide*, SC415304, or the following URL:

<http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/books/c4153045.pdf>

Each testing scenario also used a journal recovery ratio (JORECRA) of 50,000. This value can be set by calling the API **QJOCHRCV**. For more information on JORECRA, refer to the section "Journal recovery ratio (JORECRA)" on page 87.

For each test, we also enabled the symmetric multiprocessing (SMP) feature as shown in Figure 3-2 on page 60 so as to increase the quantity of parallelism employed during index maintenance.

In each testing scenario where journaling was active, we created both the journal and journal receiver for the database tables in the system auxiliary storage pool (ASP). The library containing the database tables and indexes also resided in the system ASP. We also consistently used the following journal settings, except where otherwise indicated:

- A journal receiver threshold of 1,500,000 kilobytes (KB). The following shows how to specify the journal receiver threshold:

```
CRTJRNRCV JRNRCV(<library_name>/<journal_receiver_name>) ASP(n) THRESHOLD(1500000)
```

- System-managed journal receivers. In order to reduce the overhead of deleting journal receivers, we elected not to let the system delete journal receivers that are no longer required. The following shows an example of how to specify these journal settings:

```
CRTJRN JRN(<library_name>/<journal_name>) JRNRCV(<library_name>/<journal_receiver_name>)
MNGRCV(*SYSTEM) DLTRCV(*NO)
```

We also used the standard OS/400 performance collector tools and the Performance tools licensed program product to collect, analyze and compare the runtime statistics of each of the tests. In certain cases, we used additional tools to analyze the performance data in greater detail.

The following sections describe each testing scenario in detail.

Scenario 1 - No journaling (not even implicit SMAPP journaling)

In order to obtain a baseline benchmark against which we could compare the effect of each of the journaling parameters that we enabled in the testing scenarios that follow, we first executed the application program with no journaling active on any of the database tables. In order to ensure that no background journaling would ensue during this baseline, we also disabled system managed access path protection (SMAPP).

Scenario 2 - Journaling with default values

In this testing scenario we enabled journaling on all of the application database tables, using only the default OS/400 journal and journal receiver parameter settings. No additional tuning was performed. The following shows the command *CRTJRN* with the system default parameters specified:

```
CRTJRN JRN(<library_name>/<journal_name>) JRNRCV(<library_name>/<journal_receiver_name>)  
ASP(1) MSGQ(*LIBL/QSYSOPR) MNGRCV(*SYSTEM) DLTRCV(*NO) RCVSIZOPT(*NONE) AUT(*LIBCRTAUT)
```

Scenario 3 - Journaling with omit both open and close entries specified

In this testing scenario, we again created the journal as shown in Scenario 2 - Journaling with default values, but elected to omit the database table open and close operations from the journal receiver. This parameter can be specified when you start journaling your database table. The following shows an example of the OS/400 command used to start journaling the database tables:

```
STRJRNPF FILE(<library_name>/<table_name>) JRN(<library_name>/<journal_name>) IMAGES(*BOTH)  
OMTJRNE(*OPNCLO)
```

Notice that we requested that both the before and after images of each updated SQL row be reflected on the journal.

Scenario 4 - Journaling with omit open and close and only After images

In this testing scenario, we again omitted both the database table open and close operations. In addition, we also specified that the original image of the database row, that is the image of the database row prior to the application program changing it, should not be written to the journal receiver. The following shows an example of the OS/400 command used to specify these parameters:

```
STRJRNPF FILE(<library_name>/<table_name>) JRN(<library_name>/<journal_name>)  
IMAGES(*AFTER) OMTJRNE(*OPNCLO)
```

Scenario 5 - Journaling with *RMVINTENT

We used this testing scenario to determine the impact of the journal parameter *RCVSIZOPT(*RMVINTENT)* on journaling performance.

We created the journal with the following parameters specified:

```
CRTJRN JRN(<library_name>/<journal_name>) JRNRCV(<library_name>/<journal_receiver_name>)  
ASP(1) MSGQ(*LIBL/QSYSOPR) MNGRCV(*SYSTEM) DLTRCV(*NO) RCVSIZOPT(*RMVINTENT)  
AUT(*LIBCRTAUT)
```

For more information on the **RMVINTENT* parameter, refer to 4.2.1, “CRTJRN and CHGJRN parameters” on page 85.

Scenario 6 - Journaling with *MAXOPT2

The purpose of this testing scenario was to determine what journaling performance impact the *RCVSIZOPT(*MAXOPT2)* journal parameter has.

We therefore created the journal with the following parameters specified:

```
CRTJRN JRN(<library_name>/<journal_name>) JRNRCV(<library_name>/<journal_receiver_name>)  
ASP(1) MSGQ(*LIBL/QSYSOPR) MNGRCV(*SYSTEM) DLTRCV(*NO) RCVSIZOPT(*MAXOPT2) AUT(*LIBCRTAUT)
```

For more information on the *MAXOPT2 parameter, refer to 4.2.1, “CRTJRN and CHGJRN parameters” on page 85.

Scenario 7 - Journaling with *MINFIXLEN

In this testing scenario, we wanted to determine what journaling performance impact the journal parameter *RCVSIZOPT(*MINFIXLEN)* has.

We therefore created the journal with the following parameters specified:

```
CRTJRN JRN(<library_name>/<journal_name>) JRNRCV(<library_name>/<journal_receiver_name>)  
ASP(1) MSGQ(*LIBL/QSYSOPR) MNGRCV(*SYSTEM) DLTRCV(*NO) RCVSIZOPT(*MINFIXLEN)  
AUT(*LIBCRTAUT)
```

For more information on the *MINFIXLEN parameter, refer to 4.2.1, “CRTJRN and CHGJRN parameters” on page 85.

Scenario 8 - Journaling with MINENTDTA

The purpose of this testing scenario was to determine the performance impact of specifying the journaling parameter *MINENTDTA*, for our database tables.

We created the journal, specifying the following parameters:

```
CRTJRN JRN(<library_name>/<journal_name>) JRNRCV(<library_name>/<journal_receiver_name>)  
ASP(1) MSGQ(*LIBL/QSYSOPR) MNGRCV(*SYSTEM) DLTRCV(*NO) RCVSIZOPT(*MINFIXLEN)  
MINENTDTA(*FILE) AUT(*LIBCRTAUT)
```

This scenario was tested with the Banesco application.

Note: During a later test we used the Retail Store application adjusted such that only one out of every five rows we updated involved refreshing a photo of the sporting goods item. This represents what a retailer might do when an old model of the product is retired and replaced by a newer style. We deliberately represented our photos as BLOB columns in order to investigate the behavior and effectiveness of MINENTDTA when handling BLOBs.

For more information on the MINENTDTA parameter, refer to 4.2.1, “CRTJRN and CHGJRN parameters” on page 85 and 1.3.2, “Minimize the Entry-Specific Data parameter” on page 18.

Scenario 9 - Journaling with ALL performance options

In this testing scenario, we simultaneously activated all the individual journal parameters mentioned in testing Scenario 3 - Journaling with omit both open and close entries specified through testing Scenario 8 - Journaling with MINENTDTA above. This enabled us to determine what the combined impact of all these parameters are on journaling performance.

Scenario 10 - Journaling with Journal Caching PRPQ

The purpose of this testing scenario was to measure the performance impact of the batch Journal Caching PRPQ (5799-BJC). We selected the same default journal parameters as in Scenario 2 - Journaling with default values above. We then activated the Journal Caching PRPQ (5799-BJC) by calling the system API *QJOSPEED* as follows:

```
CALL QBJC/QJOSPEED PARM('<journal_name>' '*ON')
```

For more information on the Journal Caching PRPQ (5799-BJC), refer to “Journal PRPQs (5799-BJC, 5799-AJC)” on page 21.

Scenario 11 - Journaling with ALL options plus Journal Caching PRPQ

In this final testing scenario, we wanted to determine the combined performance impact of all the journal parameters together with the Journal Caching PRPQ (5799-BJC).

We therefore created the journal, specifying all the journal parameters as in Scenario 9 - Journaling with ALL performance options above. In addition, we also activated the Journal Caching PRPQ (5799-BJC) as shown in Scenario 10 - Journaling with Journal Caching PRPQ above.

Scenario 12 - APYJRNCHG for recovering at a hot site

In this testing scenario, we used the Retail Store application in one of the Teraplex Center partitions to test the effectiveness of employing parallelism to help speed up the APYJRNCHG command in the process of moving to a hot site. The retail store application updates 40 SQL tables and it has 90 indexes built over these tables that are also updated. For an explanation of this command, refer to 4.3, “Recovering at a hot site - APYJRNCHG” on page 89.

Table 4-1 provides a summary of all of these scenarios.

Table 4-1 The testing scenarios

Scenario	Description
Scenario 1.0	No journaling (not even implicit SMAPP)
Scenario 2.0	Journaling with default values
Scenario 3.0	Journaling with omit open and close
Scenario 4.0	Journaling with omit open and close and generate only After images
Scenario 5.0	Journaling with *RMVINTENT
Scenario 6.0	Journaling with *MAXOPT2
Scenario 7.0	Journaling with *MINFIXLEN
Scenario 8.0	Journaling with MINENTDTA
Scenario 9.0	Journaling with ALL performance options (Scenarios 3-8 combined)
Scenario 10.0	Journaling with Journal Caching PRPQ with no Options
Scenario 11.0	Journaling with ALL options plus Journal Caching PRPQ
Scenario 12.0	APYJRNCHG for recovering at a hot site

4.6 Test results and findings

We collected performance statistics during each of the testing scenarios described above, then compared and analyzed the results.

The following describes these results and findings in detail.

Elapsed runtime

Figure 4-5 shows a graph of the overall elapsed runtime measured during each of the test scenarios.

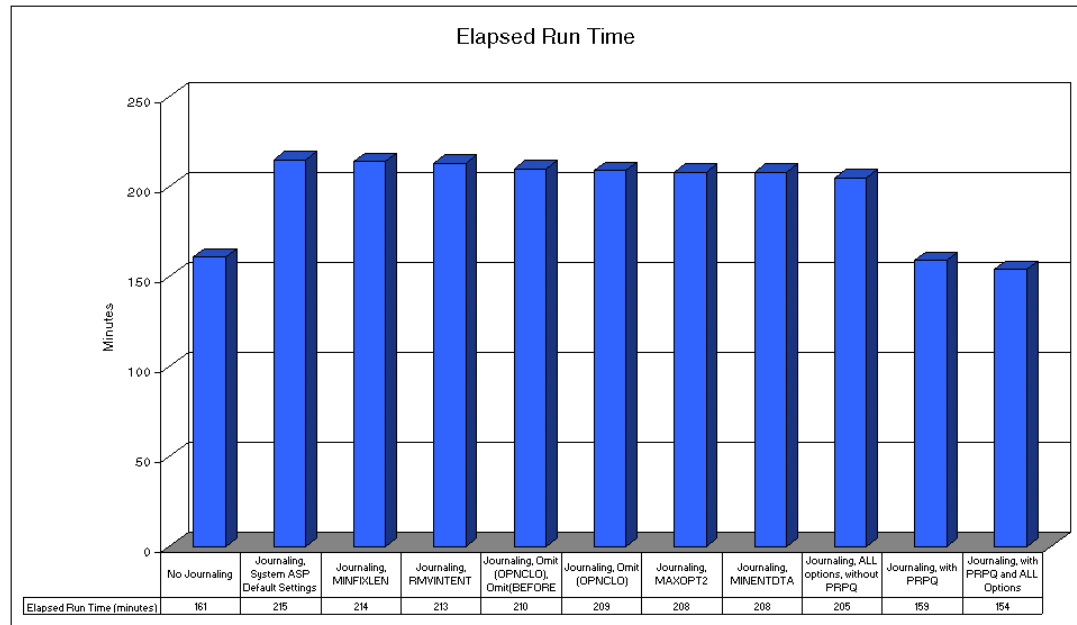


Figure 4-5 Elapsed runtime

The impact of journaling is clearly noticeable from Figure 4-5. Also, the graph further demonstrates that generally all the journal parameter options that we have used reduced somewhat the impact of journaling on the system. The actual quantity of impact from each parameter will vary from one application to another. Just because some parameters had only modest positive effect for our banking application should not cause you to conclude that it will have minimal effect for your environment. In fact, we selected these attributes because past experience with other customer applications has demonstrated that, in the right environment, each one of these journal tuning attributes can have a positive effect.

We remind you that this particular Banking application has been successively refined over the past few years and was the subject of intense analysis and tuning during two prior visits to the Rochester Benchmark Center. Our data suggests that even a highly tuned and optimized application like this one can still eke out a few more performance gains by selecting optimal journal settings. If we can achieve such results for a highly tuned application, imagine what kind of performance benefits such journal choices may make in your shop.

Now, when we first saw these measured results we were surprised. Could it really be true that our final run, the one which enabled all the journal tuning options along with the Journal Caching PRPQ behavior (154 minutes), could actually complete in slightly less time than the original batch program (161 minutes), which used no journaling at all? Impossible you say? We thought so too at first and then began to dig more deeply into the underlying algorithms employed by the SLIC layer of the journal and database support along with the design of this particular application.

Here's what we discovered. When journaling is not enabled for our tables, each updated row is individually, asynchronously written to disk as soon as the update completes. True, this is an asynchronous disk write, but there are lots and lots of them generated during the course of our run and each one takes a bit of time to schedule and consumes some CPU. Hence, our batch job is slightly slowed by this scheduling action. By contrast, when we have journaling enabled, well tuned, and especially when we also have the Journal Caching PRPQ enabled, this normal row by row SLIC database behavior is noticeably changed.

Instead of issuing separate asynchronous disk write requests for each row that we update, the operating system caches not merely the resulting journal entries in main memory but also the associated updated database rows. In fact, all of these updated database rows remain resident in main memory until the corresponding journal entries are written to disk, and that doesn't ensue until a full 128KB of journal data has been generated. That means that until we've generated a full 128KB of journal data, we've not needed to burden our batch job with the responsibility to schedule disk writes for these rows.

Provided that our application tends to update sequential rows, or at least rows that have a high locality of reference (that means they tend to be in the same neighborhood), we can end up with a similar 128KB or so of database rows resident in main memory as well. Up to this point no database rows have been written to disk and hence our batch job still has not incurred any overhead to schedule hundreds of disk write requests. Then along comes the journal, who decides that the journal caching buffer is full, writes it to disk, and schedules a similar write of the whole neighborhood of changed database rows — not one by one but rather as a single entity. This whole neighborhood is often able to be written in a single disk revolution because the rows have such high locality of reference. Wow! Not only does the Batch Journal Caching PRPQ make our journal disk write activity more efficient, but it makes the database disk writes occur in larger blocks as well — especially if our application is written such that the changed rows tend to be nearly sequential.

The underlying iSeries algorithms further enhance the performance of this kind of marriage between a nearly sequential application and a well tuned journal environment but capitalizing on a feature that most other platforms don't provide: skip writes. That means that in one single disk revolution the iSeries can write out pages 1, 2, 4, and 5 and skip over page 3, rather than having to break such a write up into two revolutions, the first to write out pages 1 and 2 and the second to write out pages 4 and 5.

We suspect that's what's happening in this final run. The application has a high locality of reference, the Journal Caching PRPQ is doing an excellent job of scheduling efficient disk writes, and the skip-write hardware is responding well — all of which work in harmony to help speed up our batch program.

The result? Our final run took slightly less time, about 4% less for this test, than our initial non-journal run.

Obviously results this good aren't going to ensue for every application. But it's a nice surprise when they do!

Figure 4-6 illustrates more clearly the measured runtime percentage differences between the different test scenarios.

As you can see from Figure 4-6, the Journal Caching PRPQ (5799-BJC) can by itself often reduce the impact of journaling down to original elapsed runtime when compared to the non-journaling run. As we stated before in our runs it even took less time, than our initial non-journal run.

With the impressive Journal Caching PRPQ (5799-BJC) activated in conjunction with all the other journal parameters, our tests have shown that the overall elapsed runtime for this particular banking environment is 4% less than executing these application programs without journaling active on any of the database tables. Again, the explanation for this behavior is better, more optimal caching.

Note: You should further bear in mind that although some of the individual journal parameters might not have a very large effect on the overall runtime of the particular banking application programs we used, runtime is not the only factor to consider. They do present other benefits, for example, reducing journal receiver sizes, reducing the possibility of disk arm contention and so forth.

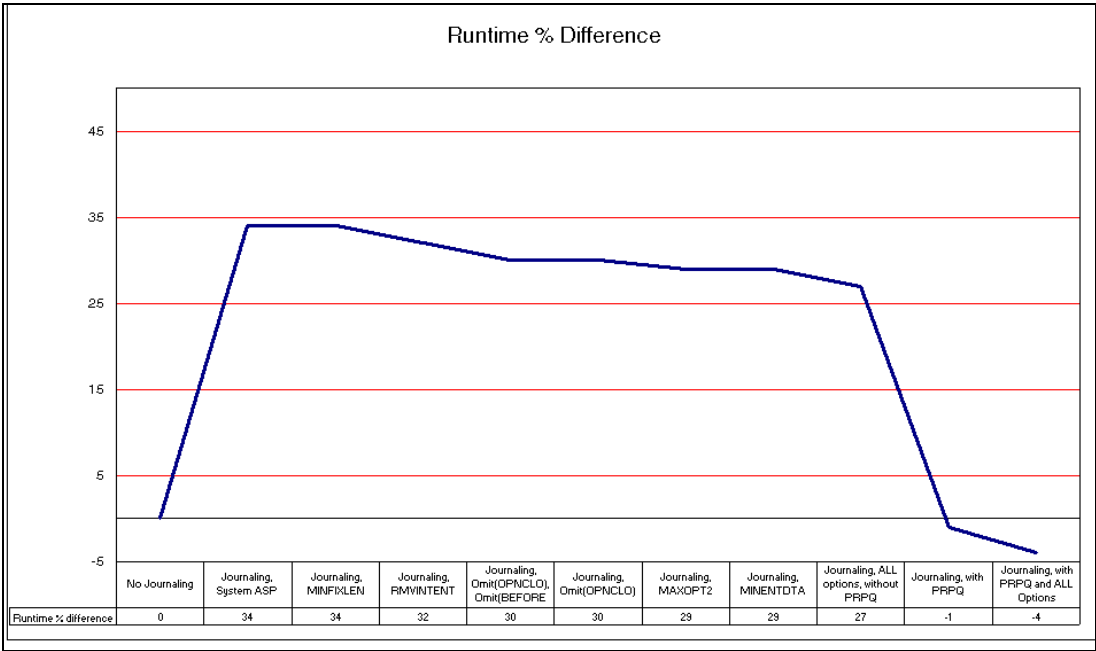


Figure 4-6 Runtime percentage variance

CPU consumption

Figure 4-7 shows some of the measured results pertaining to CPU consumption.

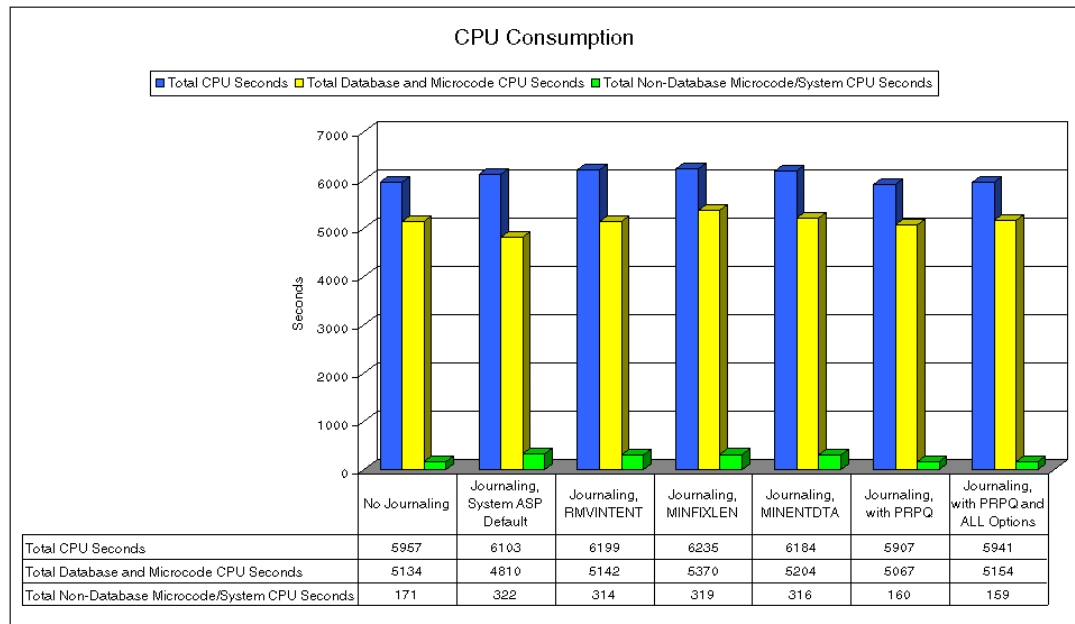


Figure 4-7 CPU consumption

As you can see from Figure 4-7, activating journaling on your database tables modestly increases the overall CPU consumption, as well as the total microcode/system CPU consumption.

However, database CPU consumption actually shows a decrease when journaling is activated. This is due to the fact that some of the main memory and disk management workload is transferred from the database to the journal management functions.

The journal parameter *MINENTDTA* in fact consumes more CPU than some of the other options. With this parameter option activated, the CPU has to work harder to determine which data bytes have changed in order to minimize the amount of data deposited into the journal receiver. If, as was true for our banking application, the quantity of bytes eliminated from the journal was rather modest, the total CPU consumed can edge up slightly. By contrast, when very few bytes within an SQL row are being changed, use of the *MINENTDTA* setting can actually decrease the quantity of CPU consumed because fewer resulting bytes are moved into the journal receiver and therefore fewer are written to disk.

For the same reasons, the journal parameter *RMVINTENT* showed a similar trend in slightly increased CPU consumption. However, when it's disk bandwidth rather than CPU which is harming your application performance, this may be the right choice to make.

The Journal Caching PRPQ (5799-BJC) on the other hand reduces CPU consumption. In fact, CPU is generally not the primary factor one should focus on when attempting to reduce journal performance overhead. Instead, disk writes are generally a far more influential factor, so let's look at the disk writes for each scenario.

Journal deposits and database writes

Figure 4-8 below shows the measured comparisons in disk write activity between four of the most interesting and dramatic testing scenarios.

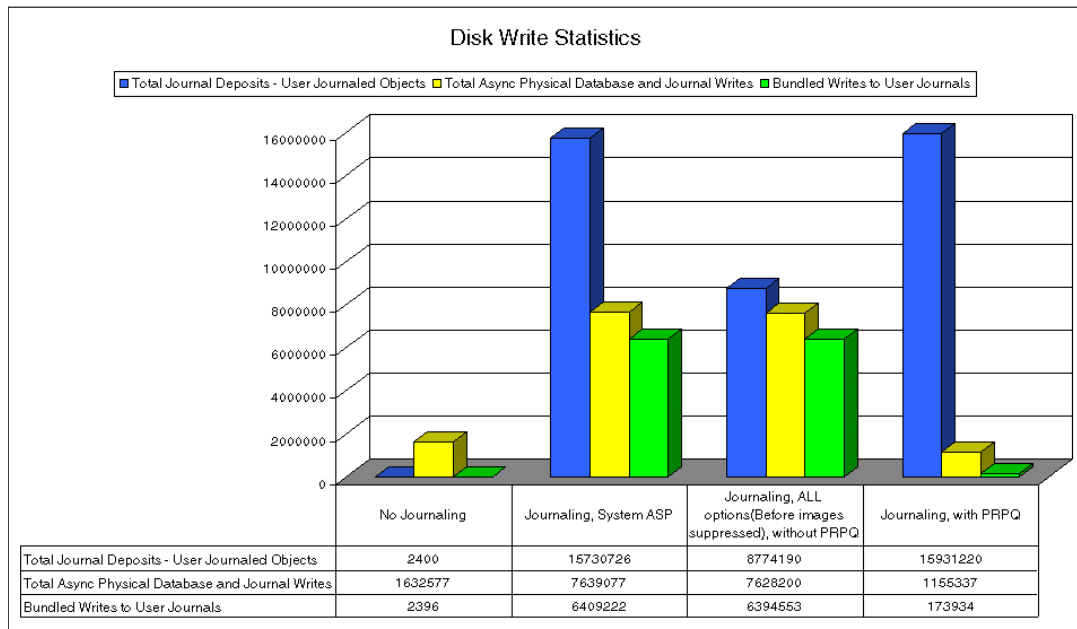


Figure 4-8 Disk write statistics

As you can see from the graph above, activating journaling on your database tables results in a dramatic increase in disk write activity. This is the primary factor you should concentrate on managing and reducing if you want to achieve optimal journal performance.

Rule of thumb: Concentrate on reducing disk writes rather than CPU as the most influential performance factor affecting journal behavior.

Don't be concerned over the extremely small amount of journal deposits which are reported even in the non-journaling testing scenario. They are not related to the Banesco database. The system regards certain system-provided audit journal and SQL cross reference tables as journaled objects and therefore this automatic system journaling activity shows up reflected as a modest amount of journal deposits in the Performance Collector, even for this so called non-journaled environment.

Our "all options" measurements employed the option which caused all of the before images to be omitted. That's why the total number of journal deposits dropped so dramatically.

The graph also shows the high degree of bundled writes that occur in a journaling environment. The basic concept of bundling is a technique used by the journal manager whereby the journaling task allows its data pages to be buffered in main memory. The system licensed internal code (SLIC) groups together multiple journal entries into a single memory-resident buffer of up to 128KB in size before issuing a write request. This results in fewer, but larger and more efficient disk write operations. Bundling is good. Wider bundling is better. If you compare the ratio of total journal deposits to the number of bundled writes you can determine the average number of journal entries present in each bundle. The higher this number, the more efficient your journal disk traffic. Clearly, as shown in this graph, use of the PRPQ substantially enhances this bundling activity. The ordinary default journal settings provided an average bundle size of 2.4 journal entries per bundle. By contrast the PRPQ run reveals an average bundle size of nearly 92 journal entries per bundle.

The journal parameters *OMTJRNE(*OPNCLO)* and *IMAGES(*AFTER)* resulted in a marked decrease in total journal entry deposits. This can be seen on the third set of bars on Figure 4-8 which is labeled Journaling ALL options without PRPQ, which includes the *OMTJRNE(*OPNCLO)* and *IMAGES(*AFTER)* options.

The Journal Caching PRPQ (5799-BJC) substantially decreased the number of asynchronous physical disk writes, as well as the number of bundled writes. This is due to its very efficient bundling capability. From a performance point of view, writing larger bundles of data less frequently has definite advantages. As you can observe from the above graph, the performance in terms of total database disk write activity is even better than the original testing scenario where journaling was not activated for the database tables.

This becomes even more pronounced for environments in which an application tends to update the same table row multiple times. The PRPQ has the beneficial effect of caching in main memory both the journal entries and the database rows. Hence both make fewer trips to disk.

Reducing database writes and I/O waits

Figure 4-9 illustrates the impact that some of the journaling parameter options have on database writes and I/O wait situations.

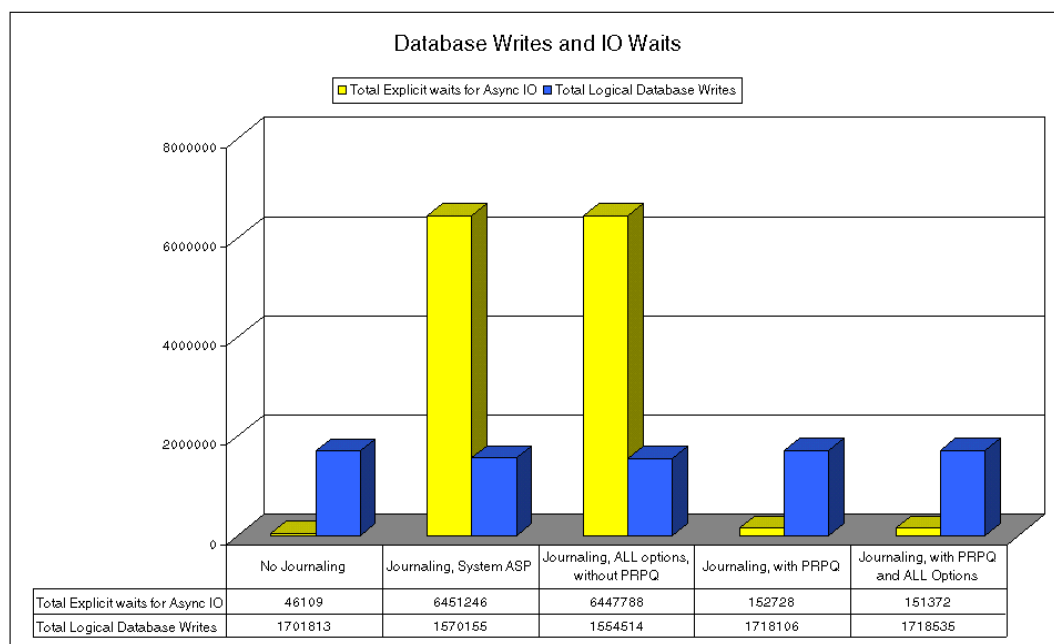


Figure 4-9 Database writes and I/O wait

In a non-journaled environment, there are virtually no bundled write operations and confirmation of a successful write operation is also not required until the table is closed at the end of job. This results in much fewer IO wait situations.

With journaling activated on the database tables, a large amount of bundled writes occur and journal management requires confirmation of a successful journal write operation before releasing the memory frame containing the data. Therefore, the number of IO wait situations dramatically increases. The more times your job is forced to sit idle and wait for a disk response, the worse your performance. Obviously, you want to decrease the number of such idle periods, and the PRPQ does this quite effectively.

You will notice from Figure 4-10 that there is a substantial decrease in the amount (megabytes) of data written when the Journal Caching PRPQ (5799-BJC) is activated. This is largely due to the fact that data remains in memory longer, resulting in repeated updates of adjacent data in frames comprising the memory buffers, instead of writing each new journal entry directly to disk as and when they occur. Most journal entries are substantially smaller than the size of a frame of main memory (4K). Without bundling, an application modifies one database row, produces a journal entry whose size is 500 bytes or less yet must write the full 4K to disk. With bundling, multiple 500 byte journal entries are accumulated in the same frame of memory and written only once. Also, the quantity of IO wait situations was drastically reduced (more than 40-fold in our tests). This can directly be attributed to the very effective bundling that occurs when this feature is activated.

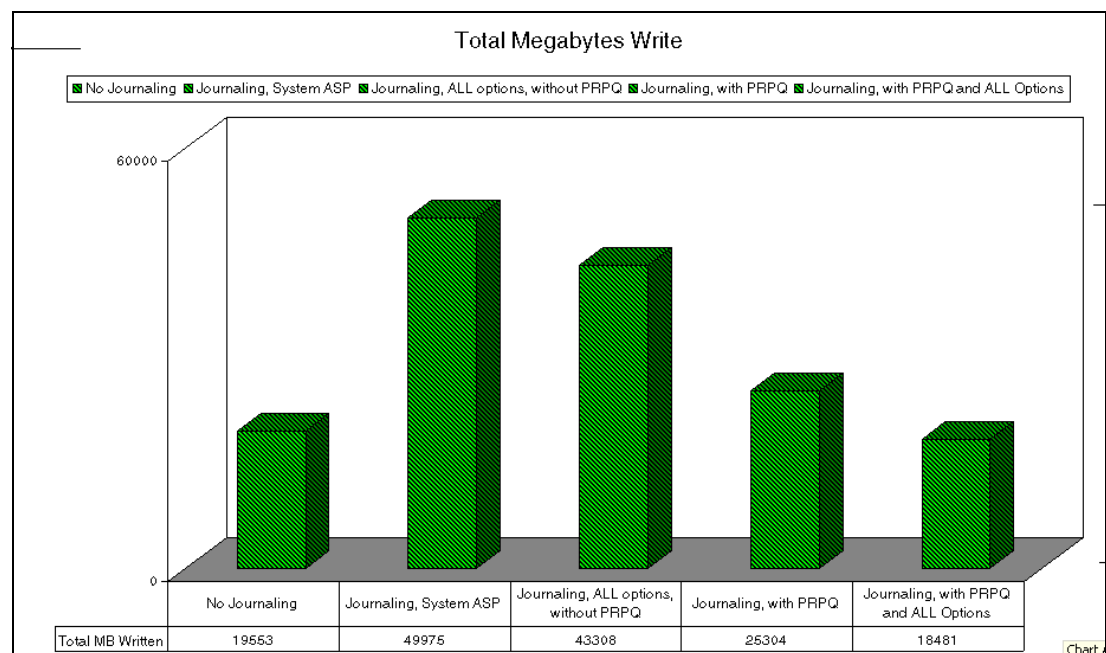


Figure 4-10 Total MB written

Rule of thumb: Most batch jobs that update lots of rows within journaled tables will experience impressive performance gains by enabling the Journal Caching PRPQ.

Journal entries per second

In interactive environments, it's not merely elapsed time that matters, but also how many journal deposits the system can absorb per second. Figure 4-11 below shows some of the results collected pertaining to the journal entry deposit rate.

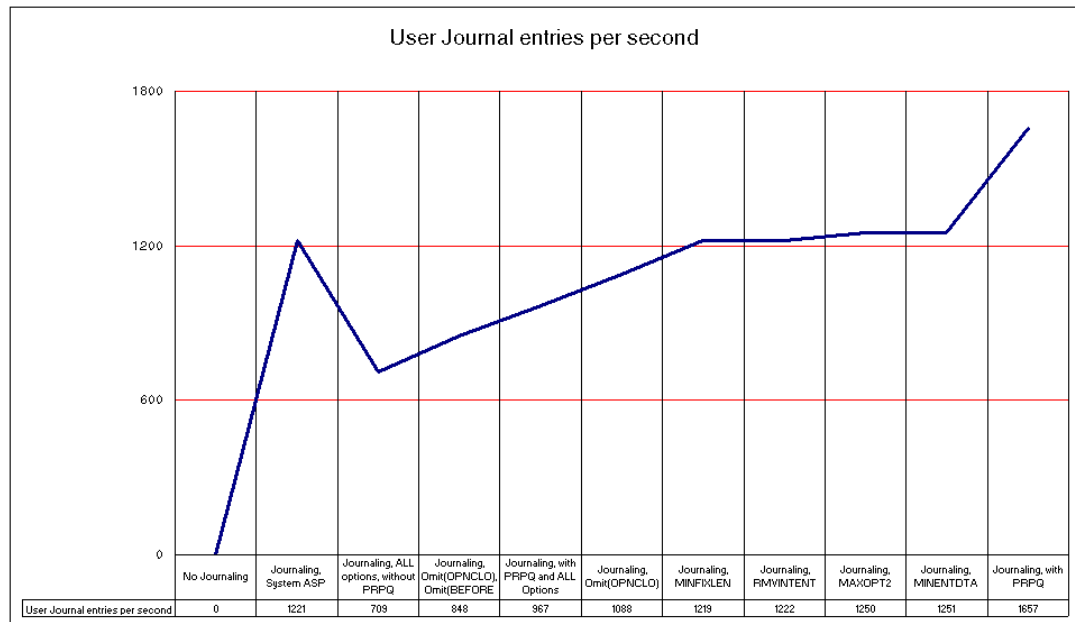


Figure 4-11 Journal entries per second

As you can see from Figure 4-11, certain journal parameter options result in fewer journal entries produced per second because certain flavors of journal entries are omitted completely, for example, the journal parameters *OMTJRNE(*OPNCLO)* and *IMAGES(*AFTER)*. Others, facilitate faster creation of journal entries because the overhead per entry is reduced.

Again, we see that we're able to sustain a significantly higher journal entry deposit rate when activating the Journal Caching PRPQ (5799-BJC), which can be directly attributed to the very efficient bundling and blocked write capabilities of this feature. In short, batch jobs that need to modify a large quantity of database rows and generate a corresponding large quantity of journal entries should see substantial elapsed runtime benefit from use of the Journal Caching PRPQ.

Total writes and I/O wait

Figure 4-12 illustrates the impact that some of the journaling parameter options have on the total write and I/O wait situations.

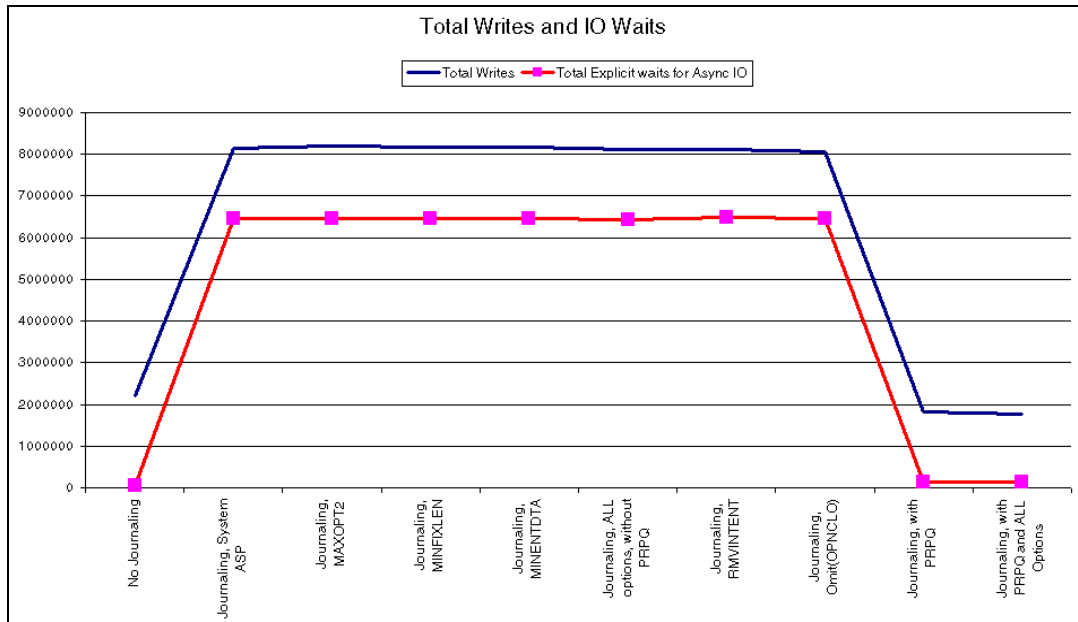


Figure 4-12 Total writes and I/O wait

In a non-journaling environment, the database manager schedules an asynchronous disk write of the data representing an updated SQL row at completion of each update function.

When journaling is activated, this behavior is different. The journal entry must be written to disk successfully before the image of the updated SQL row in the memory frame will be written to disk and released. Therefore, the entire page housing the updated SQL row is “pinned” in memory until the journal image is successfully written. This causes the memory page housing the updated SQL row to persist in main memory for a longer period of time, which could lead to repeated updates in memory, as well as data coalescence. This results in larger, but fewer disk IO operations, which in turn leads to improved performance.

Our tests further indicated that almost 99% of all writes were fast writes. This is a direct result of our use of the high performance IO adapters each housing a large 26MB write cache. Obviously, if your application programs generate journal entries at a faster rate than your IOA write cache can absorb, the percentage fast writes would be reduced and performance would suffer. For more information on the different IO adapter types and models, as well as available disk write cache sizes available, refer to the *iSeries System Handbook*, GA19-5486

You will notice from the above graph that both the total writes and the number waits for asynchronous IO are substantially lower when the Journal Caching PRPQ (5799-BJC) is activated. As previously mentioned, this is largely due to the very efficient bundling of this feature, which results in fewer disk IO operations and subsequently fewer IO wait situations.

Rule of thumb: If your fast write percentage is 90% or above, you probably have sufficient IOA write cache configured.

If not, you've probably got too many disk arms hanging off the same IOA and ought to entertain the idea of either backing off on the number of disk arms serviced by this IOA or stepping up to a newer vintage of IOA with a larger effective write cache.

Runtime summary

Figure 4-13 below shows a summary of some of the runtime statistics collected during our testing scenarios.

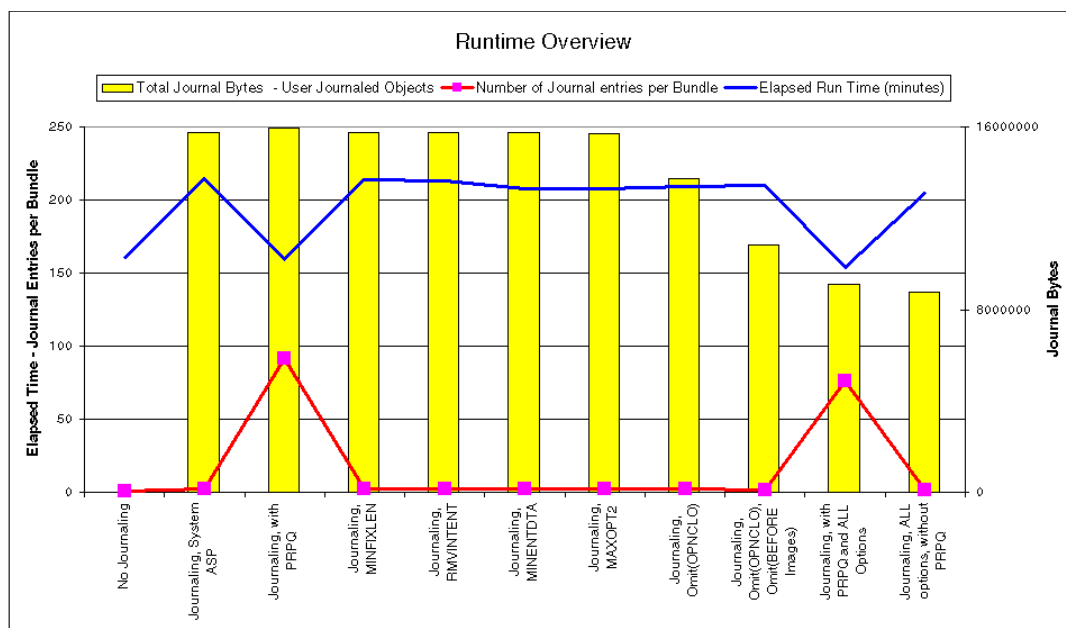


Figure 4-13 Runtime summary

You will notice from the above that none of the traditional journal parameter options as found on the CHGJRN command contributed much towards increasing the number of journal entries per bundle. The Journal Caching PRPQ (5799-BJC), on the other hand, is specifically designed to improve bundling, hence the very high bundling when we enabled this feature.

While some of these journal parameter options reduced the average size of a journal entry, the average number of entries per bundle remained unaltered.

The improved bundling rate of the Journal Caching PRPQ (5799-BJC) directly contributes to a reduced elapsed runtime when this feature is activated. Again, none of the other journal parameter options vastly reduced the overall elapsed runtime, but they contributed towards improved performance in other areas.

As you can also see from the above graph, some of these journal parameter options contributed highly towards reducing the number of journal entries deposited, which reduces overhead, which results in improved performance.

Journal overhead guidelines (CPU consumption)

One of the concerns that customers who have never journaled before often express is a concern regarding how much CPU overhead is likely to ensue. During our runs we measured this overhead and Table 4-2 summarizes these findings.

Table 4-2 CPU overhead of journaling

Batch run	% Increase in elapsed time	% Increase in CPU time	CPU/journal entry (microseconds)	CPU/bundle (microseconds)	Extra disk writes
No journaling	0%	0%	—	—	100%
Journaling no options	33%	2%	9.27	22.78	367%
User ASP, Raid-5, 10K rpm	30%	4%	15.06	37.08	367
User ASP, Raid-5, 7.2K rpm	33%	4%	14.76	36.75	364%
Disk level mirroring	54%	8%	29.96	74.38	648%
*MAXOPT2	30%	4%	15.06	37.02	371%
*AFTER and omit *OPNCLO	31%	3%	17.26	29.09	371%
Omit *OPNCLO	30%	3%	10.97	23.51	364%
RMVINTENT	33%	4%	15.36	37.70	366%
MINENTDTA	30%	4%	14.41	35.37	369%
*MINFIXLEN	33%	5%	17.61	43.17	369%
All options	28%	3%	22.30	30.65	366%
With PRPQ alone	0%	-1%	—	—	83%
With PRPQ and All options	-4%	0%	—	—	80%

Conclusions

- It is evident that journaling requires more CPU resources than a non-journaled environment. The good news is that the extra CPU consumed is only about 2.4%.
- The worst case of CPU overhead is when we are using mirroring which is an 8% increase.
- CPU should rarely be the primary factor influencing your journal tuning choices. Instead, as evidenced by the extra elapsed time column and the extra disk writes column, your primary focus should be on reducing the quantity of disk writes performed.

4.6.1 Recovering from a hot site

We've examined runtime journal performance. But how about journal behavior and performance when you need it most — at your hot site as you are trying to recover your files?

As we stated previously, one of our purposes in our test scenarios was to see how we could speed up the recovery from a system failure when moving to a hot site where we'll be using the APYJRNCHG command. This is the variety of recovery, driven by the journal, which most customers who do not have a HA BP solution would select. Figure 4-14 shows the results of the tests that were conducted.

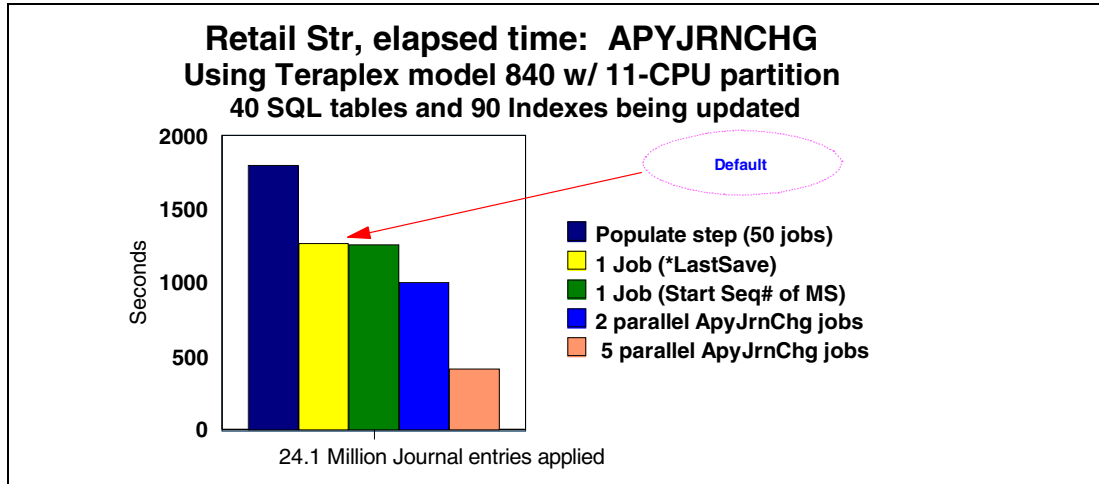


Figure 4-14 APYJRNCHG on a hot site

Conclusions

- ▶ One apply Job pegged one CPU of the 11 available CPUs suggesting that APYJRNCHG is well optimized to look ahead and avoid most page faults on both the journal receiver and the database tables — that's good news. However, it employs only one CPU and leaves the rest sitting idle. By contrast when we deliberately separated the 40 SQL tables in our collection into five groups and issued a separate concurrent APYJRNCHG command for each group we finished our recovery mission much faster. Use of five parallel jobs pegged all five CPUs so it seems to scale well.
- ▶ Apply with starting Seq# specified is 30% faster than the original populate step. That means the APYJRNCHG algorithm is well optimized.
- ▶ However, it took over a minute to locate starting point(s) for 40 Tables when using *LastSave. For customers with tens of thousands of journaled tables, it would make sense to speed up the primary step which APYJRNCHG experiences if you kept track of the journal sequence number to start from (the one housing the Microsoft flavored journal entry), instead of making the machine search for it.
- ▶ Using parallel jobs (5) cut recovery time at our hot site by 70%.
- ▶ As a general rule of thumb, you'll want to peg as many CPUs as you have available.
- ▶ It's obvious that you can probably similarly reduce the elapsed time for recovery processing for your tables if you do some simple planning up front, identify which tables or sets of tables can be individually recovered, and put in place a scheme by which you'll resist the temptation to issue a single APYJRNCHG command which recovers all tables journaled to the same journal in unison. But rather issue separate parallel APYJRNCHG commands from separate jobs for each logical subset of your tables.

Rule of thumb: You can substantially reduce elapsed recovery time at a hot site by breaking the journal apply phase into multiple parallel threads. Consider starting at least as many parallel streams as you have CPUs.

4.6.2 Throughput impact (MINENTDTA with BLOBs)

As we defined in the test "Scenario 8 - Journaling with MINENTDTA" on page 92 we wanted to measure the benefit of using the MINENTDTA parameter when the tables that are being journaled have BLOB columns.

Throughput impact: *MinEntDta w/ BLOBs

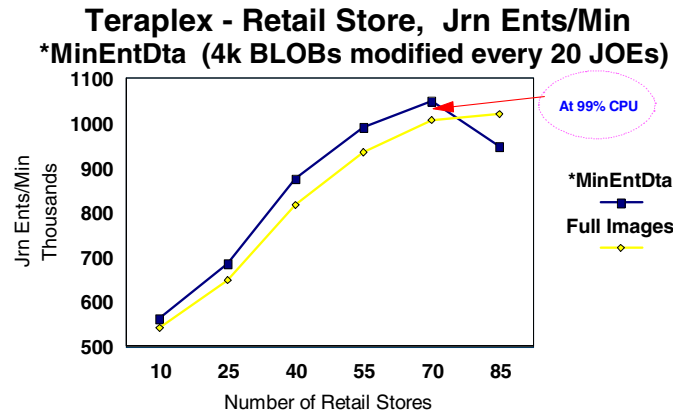


Figure 4-15 Throughput impact: MINENTDTA with BLOBs

Conclusion

When only one out of every 20 rows modified witnessed a BLOB column update, the use of *MINENTDTA paid off handsomely: total journal bytes written were reduced by up to 45% with a matching 6% increase in throughput and 7% increase in CPU consumed. The extra CPU seems justified by the fact that each job is processing more transactions per minute.

4.7 Focus Program tests

During the course of this project, we prepared another test scenario using one of the customer's programs which we called our "Focus Program". In the bank's end of day run, there's a program that calculates the interest earned on a daily basis for all the accounts (for example, savings, check, CDs). It also updates the balance of all such accounts. Two years ago, this program took four hours to run on a smaller untuned AS/400 machine. Hence it became the focus of intense scrutiny in that shop. This history led us to similarly single out this particular journal intensive program for additional analysis and testing.

In the past few years the program has gone through some dramatic application modifications which will be discussed in the next chapter. In this chapter we are interested in how the journal tuning parameters affect the elapsed time of this, so called "Focus Program". We had two versions of the program, the Older less tuned version which used to run a couple of years ago, and may represent some of the applications found in your shop, and the newest, highly optimized version which is the one that is running nightly at the bank today. Our detailed test scenario is summarized in Table 4-3.

Table 4-3 Focus Program test scenario

Case number description	Version of Focus Program
1. No journaling	Older version
2. Journaling with no tuning options	Older version
3. Journaling with all options no PRPQ	Older version
4. Journaling with PRPQ	Older version

Case number description	Version of Focus Program
5. Journaling with all options plus PRPQ	Older version
6. No journaling	New version
7. Journaling with no tuning options	New version
8. Journaling with ALL options and no PRPQ	New version
9. Journaling with PRPQ	New version
10. Journaling with PRPQ and all options	New version

4.7.1 Focus Program results

We tested both the old and new version of this critical Focus Program using the same conditions and the same journal tuning scenarios. The following two graphs show the results.

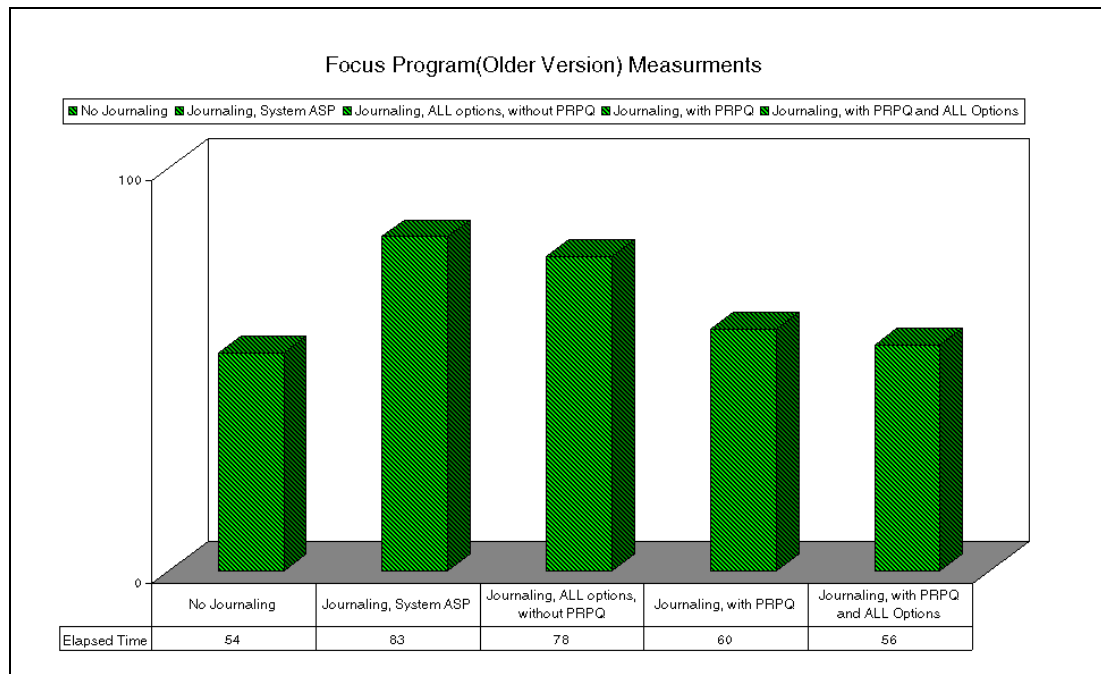


Figure 4-16 Elapsed time to execute Focus Program (old version)

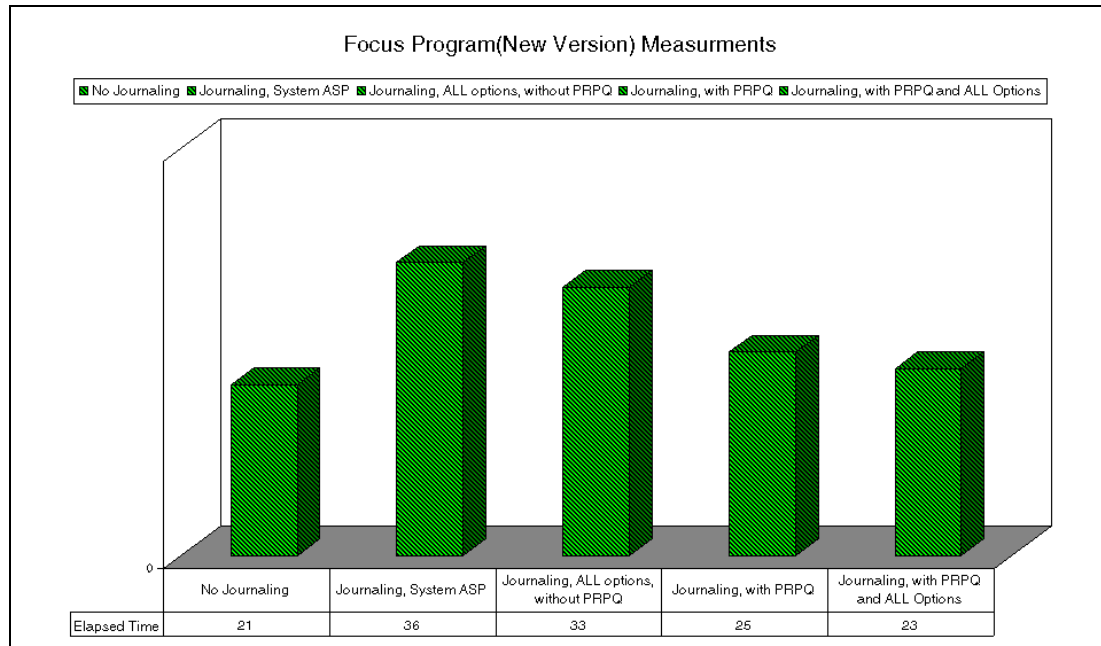


Figure 4-17 Elapsed time to execute Focus Program (new version)

Conclusions

- ▶ In both cases, there is a considerable penalty when we go from a non-journaled environment to a journaled environment without benefit of the PRPQ.
- ▶ It is the PRPQ that most helps to bring down the elapsed time.
- ▶ Specifying all journal-related tuning options also contributes to further improve performance.
- ▶ Whether you have an untuned application or a highly tuned application, it seems likely that you can further improve performance by capitalizing on the journal tuning suggestions identified in this chapter.

In the next chapter, we'll show you how you can reach beyond merely the hardware selection on journal tuning and make your application journal-friendly.



Application and software environment tuning

In the previous chapters we have shown how both hardware and journaling options can reduce the performance penalty associated with journaling on your system.

Optimal journal performance tuning is usually best achieved by doing all three:

- ▶ Configure optimal hardware
- ▶ Select optimal journal tuning parameters
- ▶ Make wise application and database tuning choices which are journal-friendly.

Once you've made the optimal hardware and journal tuning choices, you can, therefore, further reduce the impact of journaling on your system by applying a number of basic application tuning principles. In this chapter we introduce you to these concepts and we illustrate some of them with one of Banesco's programs.

When you think about tuning choices which affect application performance, don't limit yourself to thinking only about your programs. As you'll see in this chapter you can also adjust database file parameters and/or select OS/400 tuning values which will both speed up your application and make journal more efficient.

5.1 Application tuning

In today's globalization of the economy, it is frequent to see companies merging with others and in this process, a bank for example, may need to grow rapidly from 400,000 accounts to two million. Their concern is how they are going to process more transactions (5 times more in our example) in the same elapsed time. Another of their big concerns is what will happen to their batch processing if they start journaling most of their tables as part of their pursuit of achieving high availability.

With this in mind, the basic requirement from an operational point of view is to get the maximum amount of processing work done in the shortest available time frame.

In order to meet this challenge, two fundamental aspects of system processing have to be satisfied: maximize your CPU utilization and maximize your disk usage up to the recommended utilization threshold.

Also, proper application structure and an optimum runtime environment will greatly improve your application's runtime performance. Let's see what can we do from an application point of view to obtain such goals.

Job processing types

There are two basic types of processing that the iSeries can perform. The first type relates to interactive work which is associated with input from 5250 emulation clients. These are identified as type-I (interactive) jobs which are generally of short duration broken into transactions each of which produces only a small handful of journal entries per transaction. The second type of processing, identified as type-B jobs, is known as *batch processing*. Both these types of jobs can process data in a database, as well as other data types. Job types can easily be identified when you run the command to work with active jobs (WRKACTJOB).

The job type ensures that the job's unit of work is routed to the appropriate subsystem. Generally, batch jobs are routed to the QBATCH subsystem, while interactive jobs are routed to the QINTER subsystem. Furthermore, the CPU cycles consumed by an interactive workload are debited against the Interactive Commercial Processing Workload (ICPW) of your iSeries server, while the CPU cycles consumed by the batch processing workload are debited against the total processing capacity of your iSeries server. For more information on CPW ratings, refer to *iSeries Handbook*, GA19-5486.

In this chapter we will focus more on the batch type of jobs because they tend to push the journal harder and are generally more responsive to journal and application tuning.

5.2 Analyzing application performance

In order to improve the performance of your application, it is important to be able to measure the performance of your application, both before and after you have made changes, in a controlled environment. To achieve this, you should set up an environment where the same application can be run repeatedly with the same input data, in order to compare the results. Prior to each comparative test, any residual data must be removed from main storage. This will prevent the physical disk IO operations from being influenced by prior memory contents. For example to remove a database file from main memory, you can use the OS/400 command set object access (SETOBJACC) with the *PURGE option. The following is an example of how to specify this command:

```
SETOBJACC OBJ(<library_name>/<file_name>) OBJTYPE(*FILE) POOL(*PURGE)
```


We employed these disciplines as part of the performance comparisons we document in this chapter. The performance of a batch job is mainly measured by analyzing the amount of work completed over an elapsed period of time along with the corresponding amount of system resources consumed to complete the required work.

When your batch job is processing database data, you can easily determine the amount of work done by counting the number of logical IO operations performed by the job. OS/400 Data Management uses logical IO operations as a method of counting database file accesses.

Note: You can collect your job's performance data using OS/400 Collection Services (previously called the Performance Collector). The following are some performance collection types of information that you can use to analyze the performance of your job:

- ▶ Job name
- ▶ User ID
- ▶ Job number
- ▶ Job elapsed time
- ▶ CPU time
- ▶ Number of physical IO operations
- ▶ Number of logical IO operations
- ▶ Total CPU time for all active jobs during your job's runtime
- ▶ The number of collection intervals
- ▶ The time length of each collection interval

The items listed above are collected by the OS/400 Collection Services Tool and stored in the file QAPMJOBL.

5.3 Increasing performance

Once you have determined your job's processing requirements, you need to evaluate what you can do to achieve these requirements. The following are some considerations that you should take into account when making your evaluation:

- ▶ Is there sufficient CPU time available to carry out the work in the allocated time frame?
- ▶ What is required to increase the throughput rate to achieve the calculated elapsed time requirement?
- ▶ How many disk arms are needed to maintain the required throughput before the disks will become the bottleneck?
- ▶ How much write cache is required to comfortably absorb your peak disk-write rates?

5.3.1 Batch job parallelism

We can often increase the throughput and reduce elapsed time by increasing the CPU usage. CPU usage can generally be increased by making use of batch job parallelism. This means running multiple concurrent jobs in order to maximize CPU utilization. You can achieve this by restructuring your runtime environment in order to have multiple copies of your job running at the same time. For example, if your single job utilizes 25% of available CPU, four jobs should theoretically drive your CPU utilization to 100% utilization.

You can then distribute the input workload (for example, input data) evenly amongst the four instances of your four jobs. In our banking environment this could be as simple as splitting the end-of-day program into four copies and asking each copy to process 1/4th of the accounts. For example, if your input file contains 10000 rows and you have decided to run 4 concurrent jobs, then the first job could process rows 1 through 2500, the second job could process rows 2501 through 5000 at the same time, and so on.

In our case study of a bank application it could mean to break the processing of its 1000 branch offices into 4, 5 or n concurrent jobs. Each job would process $1000/n$ offices.

Before deciding on this type of approach, you should carefully study your application program, as well as your input data, in order to determine if this type of processing can be implemented in your particular environment. For example, there are applications that read and lock a control table at the beginning of the process and release it at the end. This type of situation must be addressed before splitting the program.

Note: This can be done if the nature of the application allows it. There are some batch processes that are serial or sequential by nature. Those processes usually can not be changed to run in parallel. Very often, legacy batch applications had been designed as single threaded jobs. The design of the application and job stream requires that a job completes the processing of database data before a next job processes the same (or related) set of database data. For example, an application that processes daily online transactions, updating the master files, must complete its processing before the next application is allowed to print the account balances from the master files. In this type of environment, fewer opportunities for parallelism may exist.

Due to the fact that a single job or job thread can only use one CPU at any given point in time, you can only achieve the maximum benefit of n -way processors if you are able to split your job into parallel job execution streams, thereby driving multiple CPUs simultaneously. There are two ways to achieve this:

1. You can overtly split your input data into two parallel streams and initiate a separate job thread for each.
2. You can take advantage of the fact that your modified physical files or tables are covered by lots of access paths and instruct the SMP feature to maintain your indexes in parallel.

Note: Once you've achieved parallelism for your long-running batch jobs, you'll want to assure that the parallel threads are not locking each other out thereby making some of your CPU's idle.

Rule of thumb: If you've got CPU cycles to spare, continue to split your batch job into more and more parallel streams until you drive your CPU usage well above 90%.

5.3.2 Database indexes

One of the ways to help reduce the likelihood of cross-thread or cross-job lockout is to make wise selections when you set up your indexes. In order to achieve optimum database and journal performance in an environment where multiple concurrent jobs are performing add, update and delete operations on the same database table, you should create your database indexes (or change existing ones) to use four byte nodes, by setting the Access Patch Size to *MAX1TB. This can be done using the Create Logical File, Change Logical File, Create Physical File or Change Physical File CL commands. The following are examples of these CL commands:

```
CRTLF (<library_name>/<index_name>) SRCFILE(<library_name>/<sourcefile_name>)  
SRCMBR(<source_member>) ACCPTHsiz(*MAX1TB)
```

```
CHGLF FILE(<library_name>/<index_name>) ACCPTHSIZ(*MAX1TB)
```

```
CRTPF FILE(<library_name>/<database_table>) SRCFILE(<library_name>/<sourcefile_name>)  
SRCMBR(<source_member>) ACCPTHSIZ(*MAX1TB)
```

```
CHGPF FILE(<library_name>/<database_table>) ACCPTHSIZ(*MAX1TB)
```

By changing the access path sizes to *MAX1TB, a tree consisting of 4-byte index nodes is created, rather than 3-byte index nodes. This results in reduced index contention and improved processing performance in an environment where multiple jobs concurrently update your database. The improvements that you will achieve are a direct result of enhanced algorithms used by the SLIC database management functions when the access path sizes are set to *MAX1TB.

When multiple concurrent jobs update the same database table that has the older vintage 3-byte index node structure, locking occurs at the highest level of the index, resulting in only one of the jobs having exclusive control of the index which means all the other jobs are sitting idle. This could result in a throughput bottleneck which in turn could cause low system utilization. It also reduces the number of journal entries handled in unison on behalf of both your indexes and your tables.

In the case of 4-byte node indexes, on the other hand, locking occurs at the lowest leaf level of the binary radix tree thereby reducing contention among jobs. With less contention, journal entries arrive more rapidly and therefore journal bundles tend to be wider which in turn reduces the numbers of disk writes that must be serviced.

Figure 5-1 shows the difference in the levels of locking between 3-byte node indexes and 4-byte node indexes.

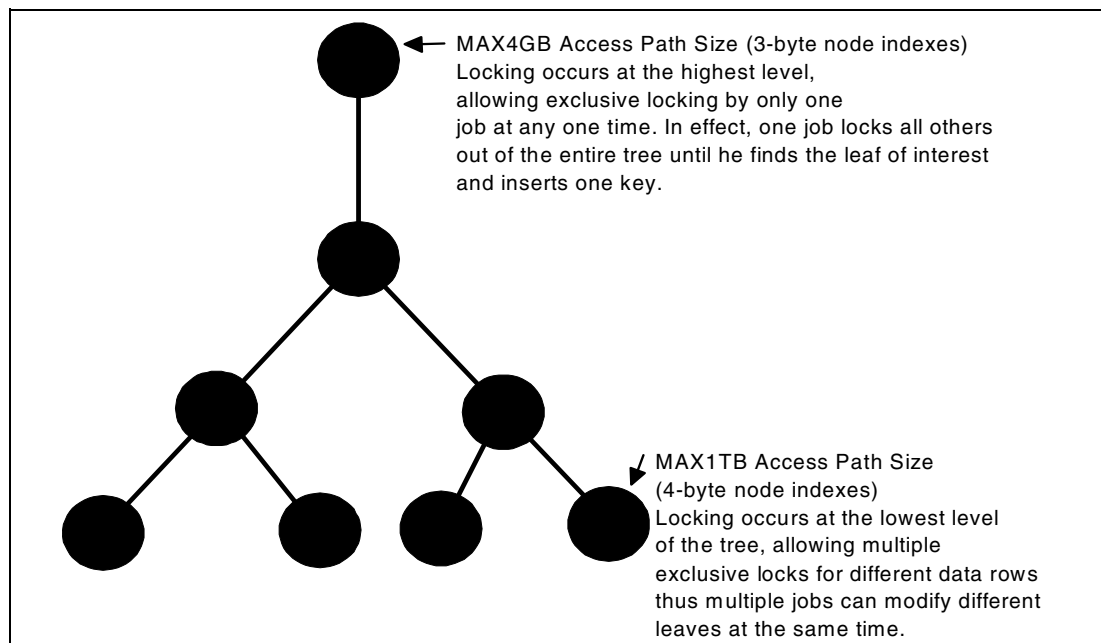


Figure 5-1 MAX1TB access path size

As you can see from Figure 5-1, indexes with access path sizes set to MAX1TB would allow multiple jobs to have control of different portions of the index at the same time. This significantly reduces locking contention and thereby improves processing throughput.

5.3.3 Minimize and optimize database table extensions

Extending the file size in order to add rows to a database table contributes substantially to a job's overhead in high-volume processing environments, especially when multiple threads are attempting to populate the same table at the same time.

By pre-allocating the disk space your application is going to populate, which you can do by use of the `ALLOCATE(*YES)` parameter of the `CRTPF` command, you can significantly reduce the performance impact of repeatedly extending a database table when writing new rows to the database table. The following is a basic example of how to specify this parameter:

```
CRTPF FILE(<library_name>/<database_table>) SRCFILE(<library_name>/<sourcefile_name>)  
SRCMBR(<source_member>) ACCPTHISIZ(*MAX1TB) ALLOCATE(*YES)
```

This technique is most effective for environments in which you have a long running batch job which fills a formerly empty table and knows in advance approximately how many rows are going to be added to this table. It's even more effective when you've split this batch job into parallel streams. Without pre-allocation of space, one job repeatedly extends the table in small increments while all other jobs sit idle. With pre-allocation of disk space for physical files and SQL tables, none of the jobs sit idle.

Figure 5-2 illustrates the technique of pre-allocating space for your tables.

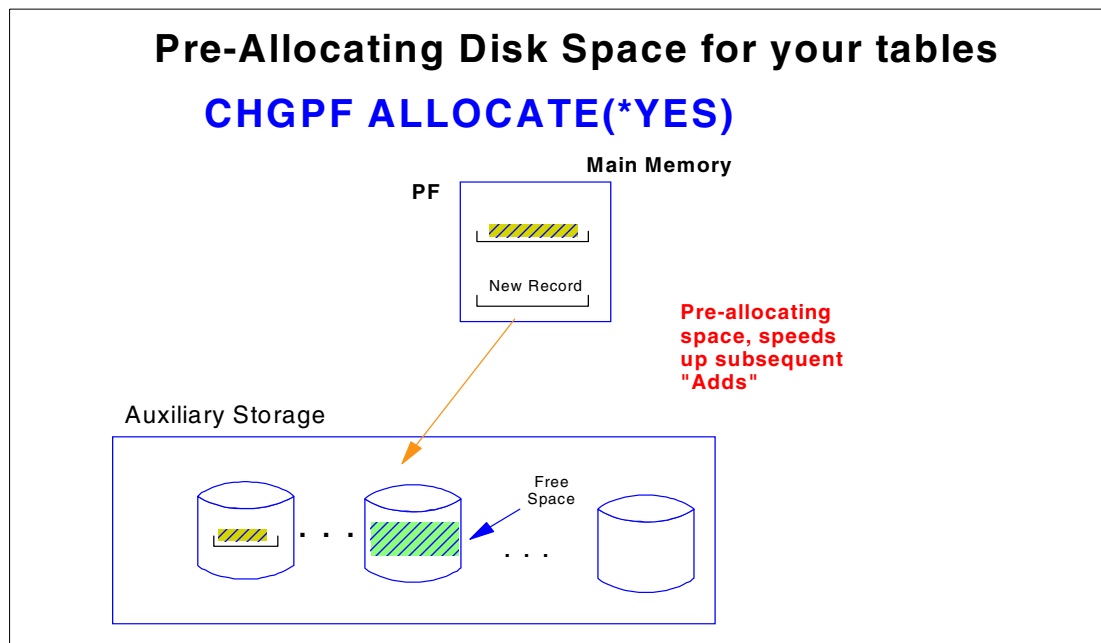


Figure 5-2 Pre-allocating space for your tables

When multiple concurrent jobs add rows to a database table, the space allocation functions of the system's Auxiliary Storage Management (ASM) microcode spend a large amount of time repeatedly extending the database table's allocated space on the disks to accommodate the new rows. While the required space is being allocated, any requests to add new rows to the database table are queued until the space allocation is completed before the rows can be added to the database table. This could significantly affect the overall runtime, as well as the throughput.

You cannot directly detect this onerous type of wait condition by using the OS/400 Performance Collector tools. You will only be able to see that a particular job encountered a high seize wait time. The lock report of the trace function will however show the database table on which the seize wait condition occurred.

To try and reduce this type of overhead, you should specify the maximum number of rows that your database table will ultimately contain (if this number is known to you) and then specify the `ALLOCATE(*YES)` parameter when you create the database table. You need not know the precise number of rows. A rough approximation will suffice. For example:

```
CRTPF FILE(<library_name>/<database_table>) SRCFILE(<library_name>/<sourcefile_name>)
SRCMBR(<source_member>) ACCPTHISIZ(*MAX1TB) SIZE(100000000 10000 5) ALLOCATE(*YES)
```

The system will then at the time of creating the database table pre-allocate sufficient disk space to accommodate the maximum amount of data that the database table is expected to eventually contain, thereby eliminating the time spent to repeatedly extend the disk space in small increments during job processing. Don't worry if you guess too low. Tables can grow well beyond the initial size allocation.

5.3.4 Managing your open data path buffer

In addition to pre-allocating disk space for new rows you may also want to improve the caching and blocking behavior of those applications which add, rather than update, lots of new rows to your tables. The secret is to give such jobs a private add-only view of your physical file and then to fill the corresponding open data path (ODP) buffer to its maximum capacity. Figure 1-5 on page 22 illustrates the presence and use of this ODP buffer. You will need to create a second file description for your database table and open this database table for output in your application program. Then, before running your application program, use the Override Database File CL command (`OVRDBF`) to point the application at this alternative file description, specifying the `SEQONLY(*YES nnnn)` parameter. The following is an example of how to specify this CL command:

```
OVRDBF FILE(<table_name>) TOFILE(<library_name>/<filedescription_name>) MBR(<member_name>)
SEQONLY(*YES 30000)
```

The value specified (30000 in the above example) represents the output blocking factor, which is calculated by dividing the length of your database row into 128000 (128 kilobyte block). You should try to get your blocking factor as close to 128kb as possible, without exceeding it.

5.3.5 Symmetric Multiprocessing (SMP)

SMP is an optional operating system feature (DB2 Symmetric Multiprocessing for OS/400 - 5722-SS1 Option 26) that significantly improves multiple concurrent processing of new rows and keys added to your database, especially if there are a large number of adds of database rows and the database table has many indexes defined over it.

Each time a database table is updated, the associated indexes are maintained as well. The index is in fact updated prior to the physical database row. This sequence of updates is maintained irrespective of whether SMP is used or not.

When you activate SMP, the maintenance of multiple indexes is overlapped such that you can have a different CPU servicing each index. Any required index pages that are not in main memory at the time are paged into main memory, resulting in a page fault, after which the new key is inserted. After all the indexes are processed, the physical database table is processed. The logical page size of indexes vary from 4KB to 64KB for keyed logical files, SQL indexes, and SQL referential constraints.

With SMP activated, the updating of the indexes is handled in the background in parallel by OS/400 System Licensed Internal Code (SLIC) tasks. Each index is processed by a separate SLIC task. Therefore, the more indexes defined over your table the more effective this technique. The user job waits until each subordinate SLIC task has completed processing its allocated block of indexes before continuing processing and calling the database functions to process the physical database rows.

The parallel index maintenance activity ensues *only* if your application is caching added rows via the ODP along with SEQONLY(*YES).

SMP does not reduce the amount of CPU utilization but rather reduces elapsed time by utilizing multiple CPUs in parallel and by servicing page faults in parallel. Whether you activate SMP or not, the same amount of CPU is being used. However, with SMP activated, the CPU utilization per second is higher, as the same amount of work is carried out in less elapsed time.

Note: DB2 SMP is only helpful when there are extra CPU cycles available. DB2 SMP will most likely increase the CPU utilization, if your CPU utilization is already around 90% during the execution of your batch job, the DB2 SMP has a good chance of hurting performance instead of improving performance.

Rule of thumb: If you add lots of new rows to a table which has lots of indexes defined over it, be sure to capitalize on both SMP parallel index maintenance and SEQONLY(*YES).

To enable SMP on your system, you must purchase and install the SMP feature and then set the OS/400 system value **QRYDEGREE** to ***OPTIMIZE** or you could instead use the CL command **CHGQRYA DEGREE(*MAX)** in the calling CL program of the jobs that update your database tables. This will allow your system to use as many SLIC tasks as required to update your indexes. Enabling SMP via the system value approach makes all jobs behave in this fashion. Enabling SMP selectively via the CL command allows you to be more selective and have parallel index maintenance only for the most critical batch jobs.

Note: You may also want to use **CHGQRYA DEGREE(*OPTIMIZE)** in jobs that run SQL statements, OPNQRYF and AS/400 Query.

5.3.6 Enable Concurrent Writes (ECW)

As each new row is added to a database table, the database functions need to validate if the new row may in fact be added to the table for example checking for the presence of a duplicate key. In an environment where the ODP blocking factor is set high and many jobs concurrently add rows to the same database table, this validation process could become a performance bottleneck. This occurs as a result of SLIC database processing serialization across jobs during the creation of the output rows.

ECW, also referred to as “holey inserts”, is a database function that overcomes this contention caused by database processing serialization when multiple concurrent jobs add rows to the same database table.

If you elect to break your original long-running batch job into parallel streams as outlined earlier, you'll end up with multiple simultaneous jobs trying to add large blocks of new rows to the same table and hence find this concurrent write support beneficial.

The ECW database function can be controlled with the system API *QDBENCWT*, for example, **CALL QDBENCWT '1'** to activate and **CALL QDBENCWT '0'** to deactivate the function. You can only benefit from this database function if your database table allows the reuse of deleted records. Refer to the CRTPF and CHGPF commands for the parameter REUSEDLT(*YES).

Note: Activating ECW enables concurrent adds for all database tables in your system.

ECW allows multiple jobs to add a large block of rows to the same database table concurrently without delay, even if another job is already adding rows to the same table. The database determines the position of each new row in the database table for any subsequent jobs and allows these jobs to continue processing. If a row fails the validation process, a deleted row is inserted into the database table at that failed position and the reason for the failure (for example, duplicate key) is signalled in the normal fashion.

Without ECW activated, invalid rows are not written as deleted rows and subsequent rows are written in strict sequential order in the database table. Due to this inability to pre-calculate the position of insertion of a new row into the database table, subsequent jobs in a non ECW environment are not allowed to insert any rows until the previous job has completed validating and adding all of its rows successfully.

Obviously, that means without ECW only one job can perform validation at a time, and all the other jobs remain idle, waiting their turn. With ECW, your jobs do not need to remain idle.

ECW also allows multiple jobs to use the CPU concurrently. This is especially advantageous if your system is equipped with *n*-way processors. The ECW approach is most helpful and effective for environments in which you have lots of jobs adding large blocks of rows to the same table at the same time and the table has lots of indexes defined over it.

Figure 5-3 illustrates how the concurrent holey adds support works on the system.

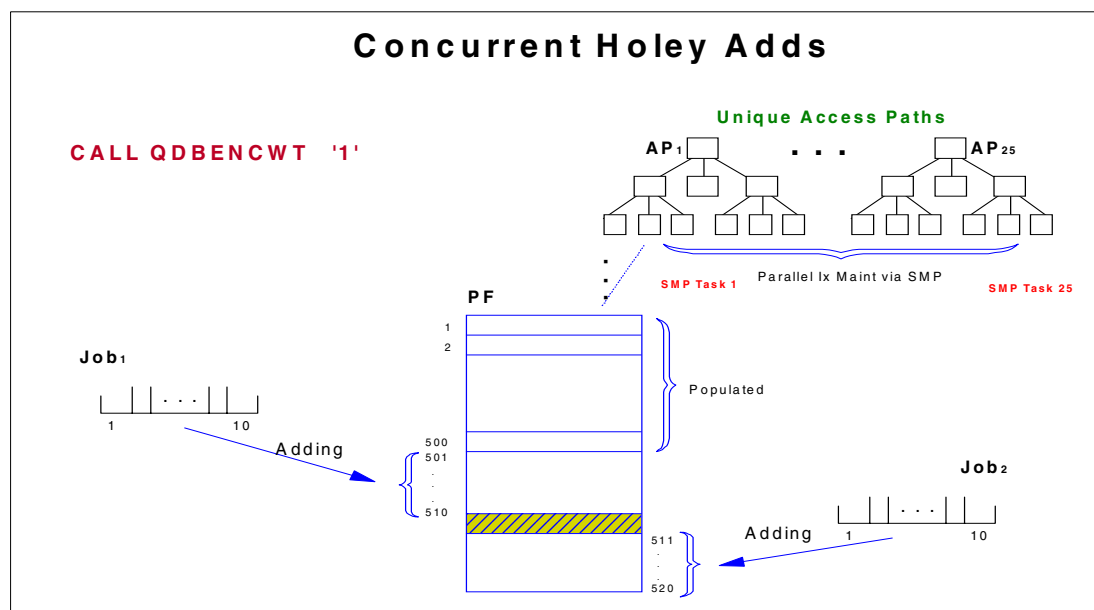


Figure 5-3 Concurrent Holey adds

5.3.7 Be journal friendly

You can often get the same work accomplished with less database and especially less journal overhead by selecting a wiser application coding technique. For example, `DELETE FROM Table_Name` finds and deletes each row individually and produces a separate journal entry for each row discarded. By contrast, if you intend to delete all the rows, the equivalent statement, `CLRPFM`, produces only one journal entry and is obviously much more efficient. Both achieve the same outcome, and both leave your table in the same final state, but one is far more journal friendly.

5.3.8 Commitment control

Contrary to most database writes, write operations to journal receivers are usually viewed as synchronous. Technically, journal write operations are neither pure sync nor pure async but rather a special hybrid. The application job which initiated the corresponding database change does indeed wait for the journal entry to be written from main memory and hence in that sense they do indeed behave as true sync writes. Under the covers however, the SLIC code passes responsibility for writing the page frames housing the journal entries to a storage management SLIC task and thus the write is scheduled async to your application job. For this reason the performance tools actually report such journal writes as async. While the storage management slic task is writing the journal entries to disk your application performs some overlapped housekeeping chores such as preallocation of space on disk for additional future journal entries. When it completes those housekeeping chores, it waits for the async disk writes to complete, thereby behaving in a sync fashion anew. This overlapped implementation technique provides improved performance over pure sync. This causes the job that requests the write operation to wait until the write operation is successfully completed before it continues processing. Obviously, the more times your job must pause and wait, the less efficient it becomes and the greater the elapsed execution time. You can however, cause these journal receiver write operations to be delayed and bundled into a single write to disk by employing commitment control.

The database write functions are aware of the fact that data survivability has to be maintained only at a commit boundary when commitment control is active, instead of at each tentative database update operation. This allows the write operations of both the database rows and the journal receivers to be delayed and bundled. All outstanding journal IO operations are written to disk when a commit boundary is reached (but not before), resulting in less total IO wait situations. The wider your commit cycles, the fewer pauses your job will experience.

Therefore, widening your commit cycles is an effective technique for improving application performance as well as enhancing journal efficiency.

Implementing commitment control deep within existing applications can be a lengthy and demanding exercise, although the benefits could be substantial. However, to lessen that effort, you could, instead, achieve nearly the same performance benefits for existing applications by electing to start and end the commit cycle only in the CL program that calls your main application programs.

If you decide to follow this approach, you would typically change the file definition in your application program to specify that commitment control is being used. Then, you would start the commit cycle in your calling CL program, call the application program and end the commit cycle in the calling CL program when control returns to this CL program. Ending the commit cycle in the calling CL program will force all outstanding journal IO operations and bundles to be forced to auxiliary storage. The performance benefit is that you've ceased issuing a synchronous disk write to the journal for each database row you modify and instead perform one journal related disk write only once each 128KB. This simple change can provide a substantial performance advantage. One customer we worked with recently reduced his

elapsed runtime by nearly 85% from this technique alone. It's generally most effective in environments that update or delete (rather than predominantly add) lots of rows. Obviously this approach should not be carried to extremes. If your batch jobs are going to add and update 100,000 rows, this single commit cycle approach probably makes sense. If your batch job is going to add and update 40 million rows placing all of those into a single commit cycle would be foolish, since the working set of locks resident in main memory employed in order to keep track of so many tentatively changed rows will surely degrade rather than enhance your ultimate performance.

5.4 Performance test scenario

We elected to put some of the application and database environment tuning principles to the test. In this test, we used two 11-way dedicated iSeries systems configured as LPAR. For further information about the configuration, please refer to Chapter 3, "Hardware choices" on page 57.

Customer's Focus Program

In this test, we used a real customer's application that we called the 'Focus Program'. In this customer's end of day run, there's a program that calculates the interest earned on a daily basis for all the accounts (for example, savings, checking, CDs). It also updates the balance of all these types of accounts. The customer reports that two years ago, this program took four hours to run on a smaller AS/400 configuration.

Since then, not only has the customer upgraded the hardware configuration, but they have also tuned their Focus Program to reduce the elapsed time of this program using several approaches we've outlined in this chapter plus:

- ▶ Elimination of unnecessary indexes
- ▶ Changing the physical files FRCRATIO value to a value *NONE (It used to be FRCRATIO of 1)
- ▶ Parallelized this batch job into three concurrent jobs

Of these three application tuning techniques, the most effective from an elapsed time perspective was the third one. During our tests we had two versions of the Focus Program. One was the non-optimized version which did not have the three application changes described above. We will call this version the *older version* of the Focus Program. The second version of the program is the one that is currently running in this customer's shop. We will call this version the *new version*. We measured the elapsed time of both programs with ordinary default journaling and no extra options and then with journaling and all options and finally with the journal batch PRPQ activated. Table 5-1 illustrates the four scenarios.

Table 5-1 Focus Program test scenarios

Case number description	Version of Focus Program
1. Journaling with no extra options	Older version
2. Journaling with all options and PRPQ	Older version
3. Journaling with no extra options	New version
4. Journaling with PRPQ and all options	New version

Note: The phrase *all options* means that we specified all of the following journal tuning values when changing journal receivers:

- ▶ *RMVINTENT in RCVSIZOPT parameter to help reduce implicit journal disk traffic

- ▶ *MINFIXLEN in RCVSIZOPT parameter to help reduce the sign of each journal entry
- ▶ *MAXOPT2 in RCVSIZOPT parameter to help broaden the disk bandwidth
- ▶ *FILE in MINENTDTA parameter to help reduce the number of bytes written to disk

5.5 Results and conclusions

Table 5-2 shows the results for all tests involving the Focus Program.

Table 5-2 The results

Case number	Results (elapsed time)
0. Original program (Older hardware)	240 minutes
1. Journaling with no performance options (OLD)	83 minutes
2. Journaling with PRPQ and All options (OLD)	56 minutes
3. Journaling with no performance options (NEW)	36 minutes
4. Journaling with PRPQ and all options (NEW)	23 minutes

Figure 5-4 illustrated the elapsed time differences (in minutes) for each test scenario with the Focus Program.

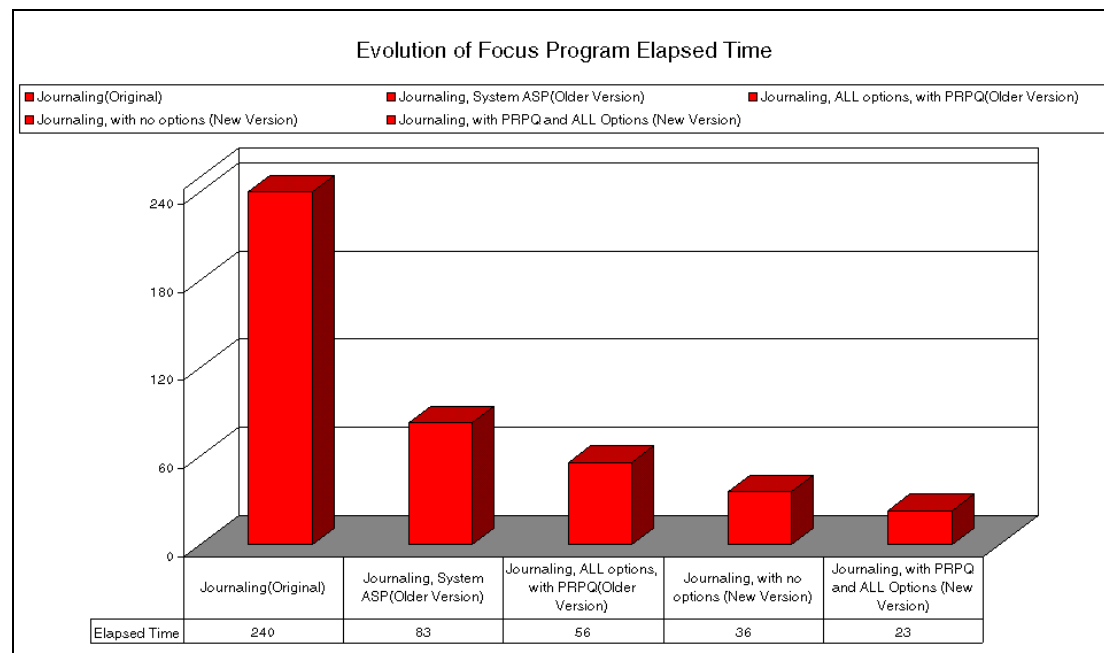


Figure 5-4 Evolution of the Focus Program

Note: The first bar to the left represents the elapsed time of the original single threaded program on slower original hardware (240 minutes). The hardware of the first bar is a 9406 Model 650 - 12 way with a CPW of 4550 and 12GB of main storage. With hardware upgrades the elapsed time is taken down to (83 minutes). The upgrades took this system to a 9406 Model 840 - 12 way with a CPW of 12000 and 40GB of main storage, with faster disk drives and more IOA write cache. Using some journal options and activating the Batch PRPQ it is taken down to 56 minutes. Then by making some application changes such as parallel jobs

without any journal options it is taken down to 36 minutes. Finally, applying the best journal tuning options and activating the batch journal PRPQ, it is taken down to 23 minutes, which for the customer is a very reasonable amount of time and a whopping 90% reduction from where he started!

Conclusions

From this specific application environment we have concluded the following:

- ▶ You can expect significant performance improvement by employing Journal Caching PRPQ. This was also evident from examples shown on the previous chapter.
- ▶ Specifying selected journal performance parameters on the CHGJRN command also contributes to performance improvement.
- ▶ You can also improve the entire batch throughput by *not* maintaining and journaling indexes which are not needed.
- ▶ Having multiple concurrent jobs each performing part of an application's work on a system can decrease elapsed time.
- ▶ The increase in IOA write cache was helpful.



Remote journaling performance

In this chapter we describe the tests we conducted and the conclusions that we reached regarding remote journaling (RJ) performance. Among the various tests that we conducted we investigated:

- ▶ Different communication infrastructure comparisons
- ▶ Comparisons among various journal receiver size option parameters
- ▶ Comparison between various remote journaling modes such as asynchronous and synchronous
- ▶ Comparison between asynchronous remote journal and using an HA BP style approach for transporting journal entries
- ▶ Comparison between using the Batch caching PRPQ
- ▶ Some comparisons made with the re-sync catch-up mode

6.1 Introduction

The remote journal function on the OS/400 offers you a reliable and fast method to transfer journal entries to a remote AS/400 or iSeries server. Remote journal allows you to establish journals and journal receivers on the target system that are associated with specific journals and journal receivers on the source system. Once the remote journal function is activated, the source system continuously replicates journal entries to the target system.

The remote journal function is a part of the base OS/400 system and is not a separate product or feature. It is implemented at the Licensed Internal Code layer. The benefits of the remote journal function include:

- ▶ It lowers the CPU consumption by as much as 10% on the source machine by shifting the processing required to harvest the journal entries from the source system to the target system.
- ▶ It eliminates the need to buffer copies of harvested journal entries to a temporary area before transmitting them from the source system. This translates into less disk writes and greater DASD efficiency on the source system.
- ▶ Since it is implemented in microcode, it significantly improves the replication performance and transmission efficiency of sending of journal entries to the target system in real-time. This real-time operation is called remote journal's *synchronous delivery mode*. If the synchronous delivery mode is used, the journal entries are guaranteed to be in main storage on the target system prior to control being returned to the application on the source system.
- ▶ It allows SAVOBJ operations executed against the journal receiver to be moved to the target system, further reducing resource utilization on the source system.

6.2 Remote journal basics

When the remote journal function is activated, the source system primes the matching remote journal receivers on the target system by replicating pre-existing journal entries as quickly as possible. This is referred to as *catch-up priming mode*. Once the specified journal receivers are transmitted to the target system, the source starts continuously sending subsequent new journal entries either synchronously or asynchronously. The mode of operation depends on what was specified when the remote journal function was activated. The different delivery modes are discussed in the following sections.

You can consider the journal receivers on the target system as a replica of the production system's journal receivers. It is as if you saved the production system's journal receivers and restored them on the target system. The timestamps, system name, and qualified journal receiver names in the associated remote journal's journal entries are exactly the same as in the local journal's journal entries on the source system. In addition, the attach and detach times of the local journal receivers on the source system are the same as the attach and detach times residing within the remote journal receivers on the target system. In short, the local and remote instance of the journal receivers are identical in nearly every detail. However, you may see a modest discrepancy in size between the local receiver and the associated remote receiver. This arises in part from the fact that you are using two different systems, which pre-allocate space for the journal receivers in different operating system environments. This may result in slightly different sizes, but the data in the associated receivers is always the same with one exception: the purely internal system-generated

so-called “hidden” journal entries which are employed only during IPL of the source system and have no relevance on the target system will be deliberately absent from the target system, provided you specified `RCVSIZOPT(*RMVINTENT)` via the `CHGJRN` command on the source.

6.2.1 Asynchronously maintained remote journals

Sending journal entries asynchronously means that each journal entry is sent to the target system at some time after control is returned to the end user application that deposited the journal entry on the source system. From a recovery standpoint, asynchronous mode is slightly less desirable than sync mode. The reason is that the source system may occasionally have a few journal entries which have not yet been transmitted to the target system. Using this method allows for recovery that may have to proceed on the target system without a few recent journal entries given a failure on the source system. Asynchronous transmission however tends to have minimal performance impact to the local system when compared to the synchronous delivery mode.

The main advantage of the asynchronous delivery mode is the minimal performance impact on the production system.

The disadvantages of the asynchronous delivery mode include:

- ▶ A risk that a few of the final database transactions may still be trapped on the source system when that system goes down.
- ▶ In case of production system’s failure, the process of re-synchronizing the primary and replica databases can be somewhat more difficult when compared to the synchronous delivery mode.

However, asynchronous mode still remains a valid choice for some who intend to implement the remote journal function. You should consider asynchronous mode when:

- ▶ Your shop can tolerate the delayed arrival of a few recent journal entries, such as in data warehousing environments.
- ▶ Your production system is heavily utilized and you do not have resources to compensate for even the very moderate extra overhead of synchronous mode.
- ▶ You have slower communication gear between the source and target systems such that you cannot easily keep up with the volume of journal traffic generated on your production machine.
- ▶ The distance between your source and target systems makes waiting for synchronous transmission followed by synchronous acknowledgement response from the distant target system impractical.

We were pleasantly surprised to discover during our testing of this function that asynchronous remote journal support over a sufficiently fast communication line never fell behind nor left recent journal entries trapped on the source system no matter how hard we pushed it. In fact in every instance we never saw more than 5 ms transpire between the time we produced the journal entry on the source system and the time it was sent down the communication wire. This leads us to conclude that the better performing async mode may be a satisfactory choice for many shops who have sufficiently fast communication lines. We also noticed, as you’ll see later that it appears to be a superior performance choice over any approach which harvests journal entries on the source system via software products installed on top of the source journal.

Note: If it's not practical or cost-effective to upgrade the speed and capacity of the line, there are others choices that include:

- ▶ Capitalize on the journal filtering support provided by most HA BP products, especially if the HA BP product you select is able to perform filtering on the source machine. This is an especially attractive choice if you happen to journal thousands of files to the same journal but only need to replicate a small percentage of them.
- ▶ Consider splitting your files into two groups: those that need to be replicated and those that do not. Your critical files would be routed to one journal and this journal could employ the remote journal support while your less-critical files could be routed to a separate journal. This has the advantage of allowing the HA BP software to more efficiently read/harvest only the journal entries related to the interesting/critical files.

Figure 6-1 illustrates a simplified diagram of data flow involved in the asynchronous mode.

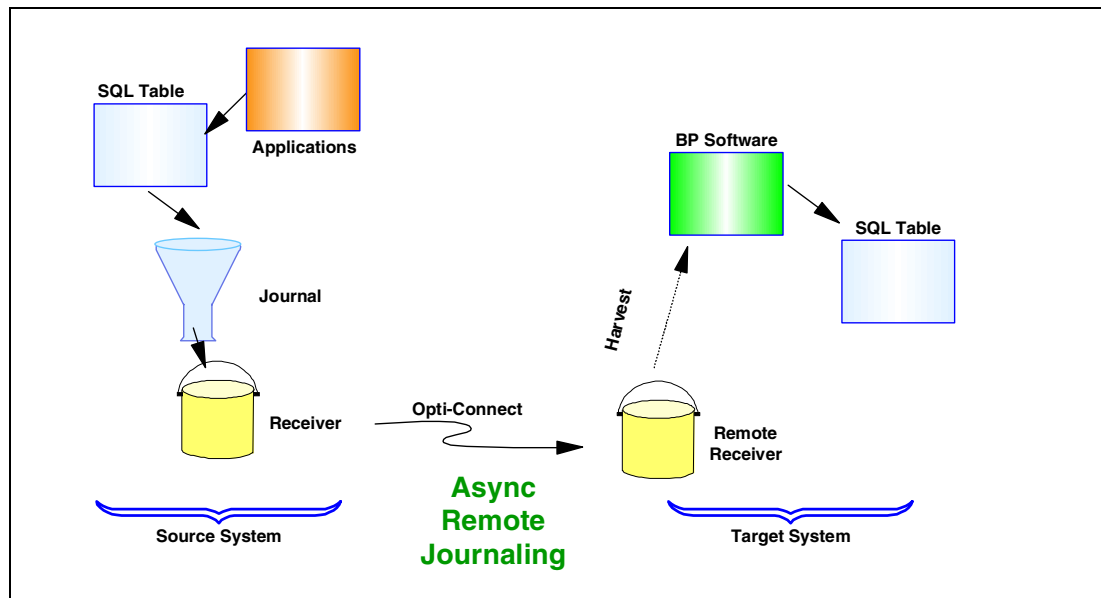


Figure 6-1 Remote journaling: asynchronous mode

6.2.2 Synchronously maintained remote journals

In synchronous mode, journal entries are replicated to the main memory on the remote system before the corresponding database rows are updated on the source system. After an arrival confirmation message is returned to the source system, the journal entry is deposited to the local journal receiver. Next, the actual database update on the source system is made, thus, the target system is updated in real-time with all of the journal entries as they are generated by a user application on the source system. Synchronous journaling assures recovery processing that is missing no journal entries on the target system when an outage is experienced on the source system. Synchronous remote journaling assures that the target remote journal houses a copy of the new journal entry before the database on the source system is modified. Thus, no crash of the source system can result in having the source database up-level and more modern than the remote journal. Sending journal entries synchronously to a target system modestly impacts the journaling throughput on the source system.

Our experiments showed async mode having an application performance benefit over sync mode ranging from 5% to 18%.

The main advantages of the synchronous delivery mode include:

- ▶ There will be no trapped transactions on the production system. The synchronous mode should definitely be your choice if it is intolerable to switchover to the target system with a few of the final database transactions trapped on the source system because these transactions were not sent to the target in real-time.
- ▶ The journal image reaches the target system before reaching the disk of the source system. Therefore, there is no delayed arrival of journal entries.

Caution: While using both remote journal synchronous mode and the Journal Caching PRPQ on the same journal at the same time is a legal combination, it probably is an inconsistent choice. Synchronous mode implies that your database changes are so critical that you're willing to slow down your application and make it wait for transmission to the target machine of the most recent database changes. On the other hand, the caching behavior of the PRPQ suggests that row by row recovery is not your primary objective and that you're willing to accumulate multiple rows in a main memory buffer before transmitting them if by doing so you can improve your application performance. When the operating system is presented with these conflicting goals, only one can win. The winner is performance.

Figure 6-2 illustrates a simplified diagram of the behavior in synchronous mode.

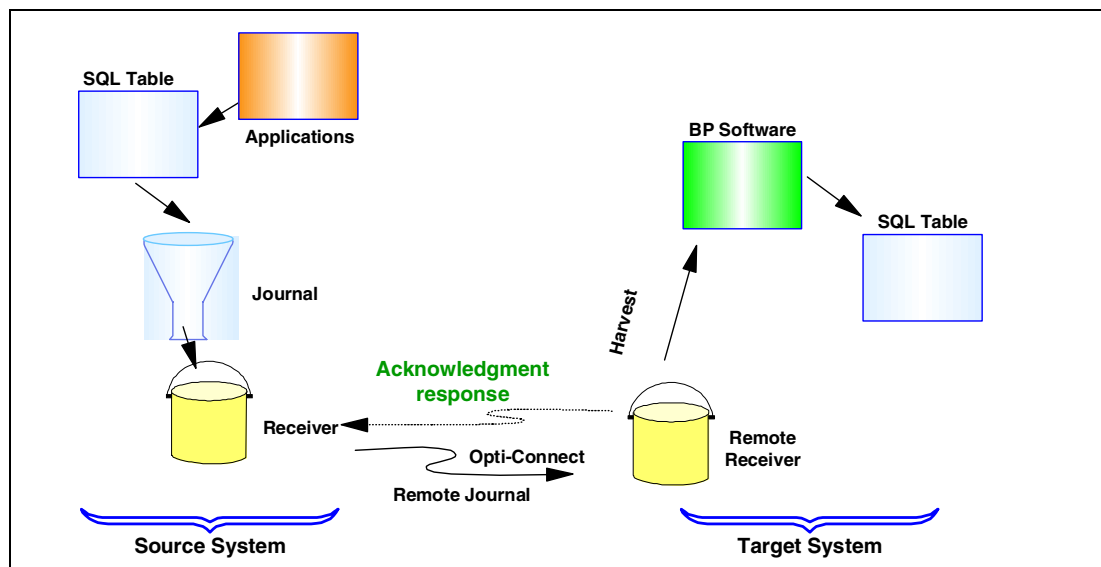


Figure 6-2 Remote journaling: synchronous mode

6.2.3 The potential communications bottleneck

As with any software that must communicate with other systems, there's a cost associated with the communication layer. This cost can range from extreme, when transmitting over a communications layer lacking sufficient capacity, to minimal when the properly sized communications hardware is used. The communications layer can be divided into two parts:

- ▶ Communications software and hardware protocols
- ▶ Communications line (hardware)

The communications software and hardware protocols is what pushes the data down the communications line. For the most part, the difference between software and hardware protocols is how much data it can push down the communications line at one time. The difference between communication lines is the speed at which data can travel from point A to point B.

Communications hardware bandwidth (capacity) is measured in megabits per second (Mbps). This means how many megabits (millions of bits) can be transferred across the communications hardware per second. This is often referred to as the size of the *communications pipe*. One can visualize this effect by thinking of a pipe with water passing through it. The bigger the pipe, the larger the volume of water that can pass through it in any given time.

As far as performance is concerned, the communication pipe is critical because of the impact it can have on remote journal. As explained in this chapter, when using remote journal with undersized communications hardware in either synchronous or asynchronous delivery mode, a noticeable performance impact can be expected. However, when properly sized communications hardware is employed, the performance overhead of remote journal is quite modest.

6.2.4 High Availability Business Partners (HA BP) solutions

Anyone who's serious about achieving high availability will want to look beyond the mere transport of journal entries from a production machine to a backup machine. That's where the software provided by HA BP's comes into play. Such products typically monitor the health of your HA replication approach as well as ensure that they reach beyond traditional journaled objects and also replicate changes made to critical system objects such as user profiles, job descriptions, and spooled files. A total HA solution generally mandates such support.

In V5R1 the underlying operating system broadened the list of objects which can be journaled to include data queues, data areas, and IFS files. As this list continues to grow the volume of journal traffic sent from your production system to your backup system will increase as well. As a consequence, you'll want to ensure that you're using the most efficient underlying software technology to drive the transport of the corresponding journal entries between your production and backup machine. Historically, the HA BP software has provided this support and has employed an asynchronous approach. In keeping with that tradition, the new remote journal support similarly offers an asynchronous transport option. In fact, we performed side-by-side measurements to compare the remote journal transport technology with the traditional HA BP transport technology. You'll see the results of that comparison on the ensuing charts.

For folks who favor keeping the replication and transport overhead at the lowest possible level, an asynchronous approach is clearly best. For environments which tend to journal lots of objects to the same journal but only want to replicate a small subset to the backup machine, a mechanism for filtering out the unneeded journal entries may be even more attractive. Some HA BP products provide such filtering facilities. However, if nearly all of your journal entries need to be transported to the backup machine, use of asynchronous remote journal support seems to be more efficient than any other choice.

When the asynchronous transport is managed by the remote journal facility, it occurs in real time deep in the machine at the SLIC microcode level. This not only makes it a remarkably efficient transport mechanism but also affords the opportunity to actually send fewer bytes down the wire. In addition, as our measurements reveal, there's very little delay in getting such journal entries placed on the wire. Provided that your communication gear is fast enough, such remote journal managed asynchronous transports are generally on the wire before control is returned to your application. Technically this asynchronous transport is

managed by a separate SLIC task which runs in parallel with your application and so could theoretically have some slight delay in responding to the presence of the new journal entries. However, in practice, we witnessed no such actual delay within the granularity of our ability to measure such delay with our tools. You'll see charts later showing how remarkably fast this so-called asynchronous activity actually is. That no doubt stems in part from the fact that this underlying async remote journal task runs at such a high priority. In short, we found few instances in which we would lose much sleep over employing a remote journal-enabled asynchronous transport approach.

However, if you absolutely need to squeeze out even the last 5 milliseconds (or less) of potential delay in order to sleep well at night, the remote journal support has yet another mode you may want to select: synchronous remote journal transport. Unlike the asynchronous approach, this technology doesn't farm the transport mission out to a separate SLIC task but rather enlists your own application job to put the new journal entry images on the wire. Hence, by the time control returns to your application you know not only that the related database change has been made to your SQL table but also that the matching journal entry has arrived in the main memory of the backup machine. If your two machines are within a reasonable distance of each other and your communication line strung between them is sufficiently fast, waiting for this synchronous confirmation may be the right choice. For most shops, however, the asynchronous remote journal support is so darn fast in practice that it may suffice.

The good news is that nearly all of the HA BP products now offer a version which allows you to employ either the asynchronous remote journal transport technology or the synchronous remote journal transport technology in addition to their more traditional non-SLIC application level asynchronous approach.

In short, most HA BP providers have embraced the remote journal technology and are ready to put it to work for your shop.

6.3 Test environment

In this scenario, we used two 11-way model 840 iSeries systems each configured as an LPAR system. We didn't use a user ASP to house our journal receivers and our system ASP was protected by RAID5. Our test environment is described in more detail in Table 6-1. For a complete description of the environment used please refer to 2.1.1, "Teraplex Center" on page 29.

Table 6-1 Benchmark test systems

System parameters	System1- SourceSystem	System2- Target System
Model	840	840
Processors	11 ways	11 ways
Main memory	40GB	40GB
System ASP size	684.4GB	684.4GB
Number of disk arms	44 arms	44 arms
OS/400 Version	V5R1M0	V5R1M0

Since one of the main purposes of our benchmark was to measure the efficiency of different communications scenarios, we set up a range of communications hardware connecting target and source systems. The infrastructure we tested is summarized in Table 6-2.

Table 6-2 Benchmark communications links

Hardware	Speed	Protocol
Token Ring	16 Mbps	TCP/IP
OptiConnect	1 Gbps	TCP/IP - MTU(1496)
Ethernet	100 Mbps	TCP/IP

To measure the transmission speed difference, we also configured TCP/IP over OptiConnect and changed the MTU size to 1496. The default value is 8991.

6.4 Performance benchmark scenario

To benchmark performance, we elected to simulate a realistic user environment with regard to system load. In this scenario, we used the retail-store application. For a complete description of this environment please refer to 2.2.2, "Retail store application" on page 53. It simulates interactive transactions. The transactions are simulated by multiple instances of the same program running against a set of database tables, indexes and a shared data area. The program is written in C with embedded SQL statements which produce the database activity. The database tables include transaction tables which are populated throughout the run and other inventory tables which are updated as the transactions take place. Data areas are used to maintain a transaction ID shared between all of the jobs using a set of the data.

In order to collect a variety of data points, we ran this retail store environment at increasing levels of activity, ramping up the number of retail stores being serviced.

For the ramp-up scenarios, a number of the environments are set up with multiple jobs producing transactions against each set of data. The jobs are allowed to run for a fixed period of time before being ended. Data is collected after the jobs have ended to determine how many transactions were completed for an interval. The data sets are completely destroyed and recreated between each iteration. The number of sets of data as well as the number of jobs producing transactions increases with each iteration in order to produce a more stressful workload for each iteration.

We developed 5 categories of tests to evaluate various kinds of journal environment measurements. The scenarios are summarized as follows:

Table 6-3 The test scenarios

Category number	Description
1	Compare the network performance among the different communications infrastructure using the original non-remote journal HA BP sending approach: - Token Ring - Ethernet - OptiConnect
2	Remote journaling
3	Add each of the journal options one by one: - RMVINTENT - MINENTDTA - PRPQ(5799-BJC)

Category number	Description
4	Remote journal asynchronous mode Compare performance among the following: <ul style="list-style-type: none"> - With no High Availability(HA) Solution - With an HA Solution - With an HA Solution with Journal Caching PRPQ (5799-BJC)
5	Remote journal synchronous mode Compare performance among the following: <ul style="list-style-type: none"> - With no HA Solution - With an HA Solution - With an HA Solution with Journal Caching PRPQ (5799-BJC)

For tests number 4 and 5 we wanted to investigate the impact of having a journal harvesting job plus a non-remote journal transmission job running on the source system as would be typical of the approach used by customers who have an original non-remote journal HA BP technology replication product installed and compare it to the performance characteristics of a more modern HA BP product which capitalizes on use of remote journal. Two of the iSeries HA BP's graciously agreed to work with us while developing the tests documented in this chapter. Both have incorporated remote journal support into recent versions of their products. The results documented in this chapter reveal that they've made a wise choice and that the move to recent versions of their products capitalize on the advantages of remote journal technology. Figure 6-3 illustrates the HA harvesting approach without the use of remote journaling.

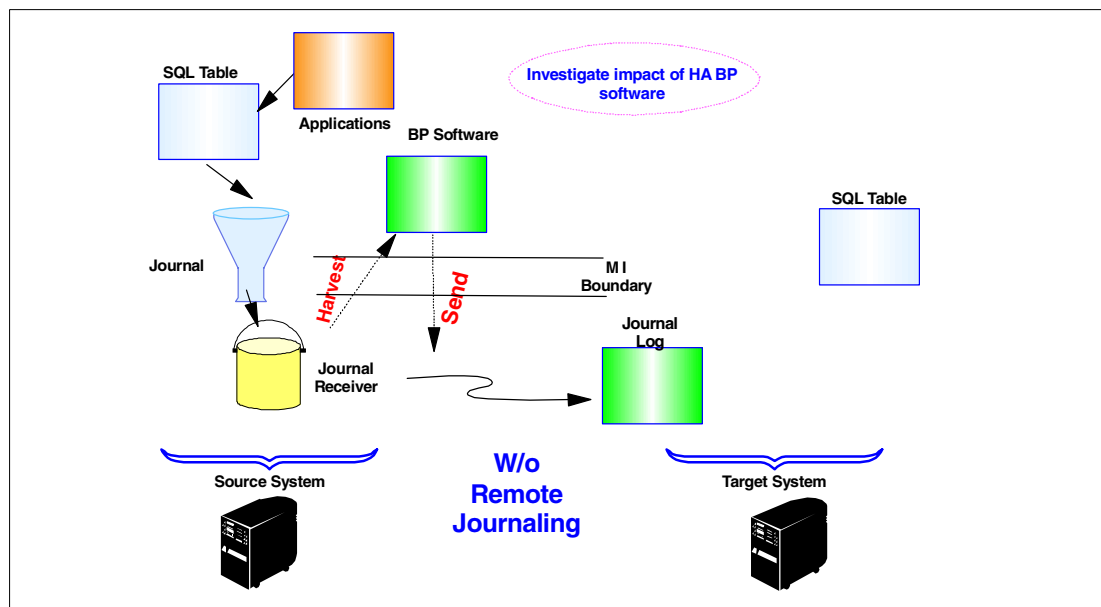


Figure 6-3 HA BP data harvesting concept without remote journaling

Figure 6-4 illustrates the environment for a HA solution using remote journaling.

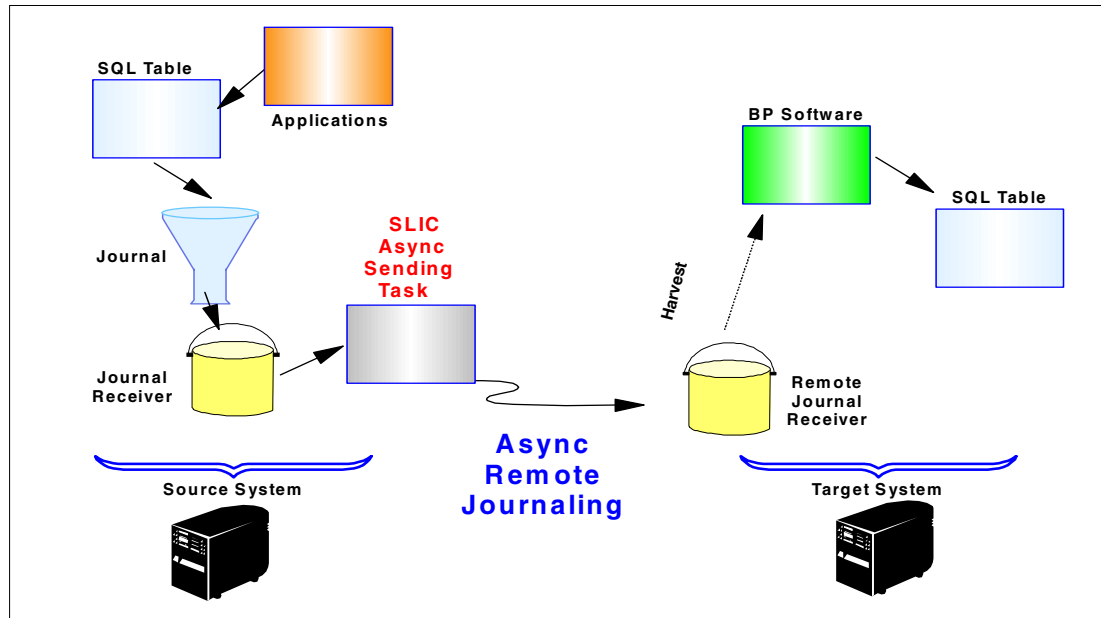


Figure 6-4 Asynchronous remote journal for HA environment

6.5 Conclusions and findings

In this section we have summarized our conclusions and findings regarding the performance factors affecting Remote journaling. These conclusions were derived from the following tests:

- ▶ Communication Infrastructure Comparisons
- ▶ Comparison between using the 5799-BJC PRPQ on the target system during HA BP replay and not using it
- ▶ Comparisons among various journal receiver size option parameters
- ▶ Comparison between asynchronous remote journal and using an HA BP style approach for transporting journal entries
- ▶ Comparison between two remote journal sending modes, both asynchronous and synchronous

6.5.1 Communication infrastructure comparison

Figure 6-5 shows the results of the retail store application that was tested over communication hardware comparing Opticonnect, 16 Mb token ring adapters and 100Mb Ethernet.

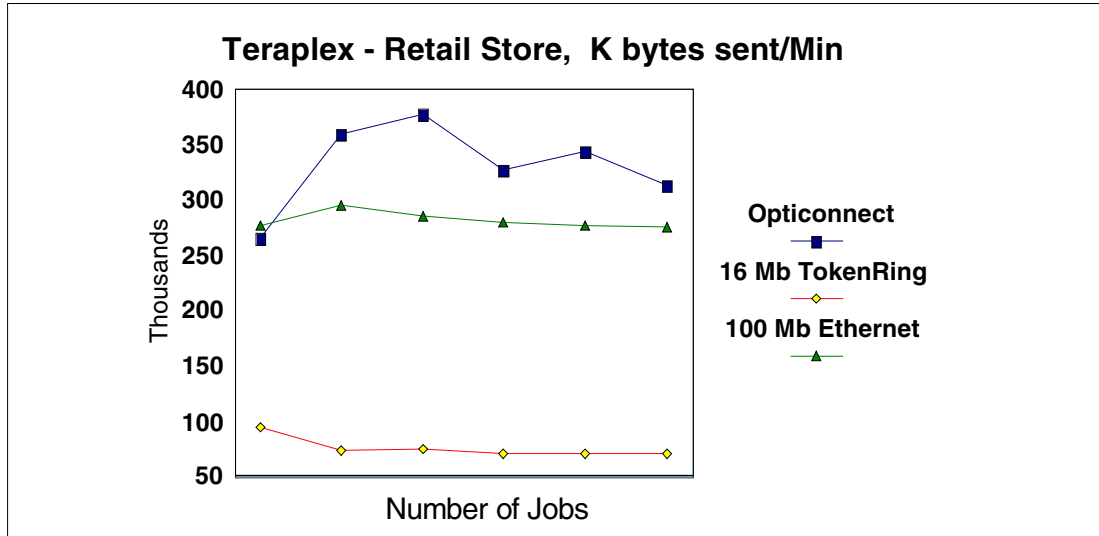


Figure 6-5 Average sustained KB sent per minute by each communication infrastructure

Conclusions

- ▶ The 16Mb token ring pipe was a constraint for this environment even at low volumes (Opticonnect over fiber handled five fold move volume at peak than 16Mb token ring).
- ▶ Unless your journal volume is very low, a 16Mb pipe is probably going to be too small to handle your journal traffic between systems.
- ▶ OptiConnect beats 100Mb Ethernet at peak (the third point on the graph) by up to 33 percent.
- ▶ While 100Mb pipes are good, our 11-CPU configuration needed even more capacity. It wasn't until we stepped up to 1000 Mb that we had excess capacity for this environment.

6.5.2 Comparison between synchronous and asynchronous remote journal

Figure 6-6 shows a comparison between synchronous and asynchronous remote journaling.

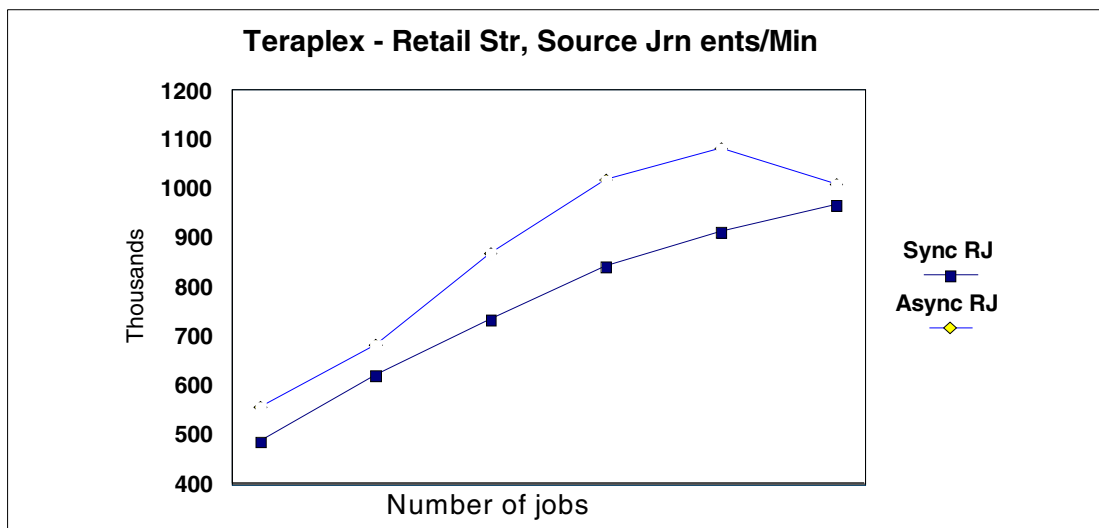


Figure 6-6 Source journal entries sent per minute without the PRPQ

Conclusion

- ▶ Synchronous remote journal gives nice consistent ramp up of application behavior on the source system, but can hurt throughput by as much as 18% at peak. This represents the apparent cost of having assurance that no transactions will be lost in the event of crash.
- ▶ The good news was that this overhead never exceeded 18% and was even more modest at lower journal traffic volumes.

6.5.3 Comparison regarding the (5799-BJC) PRPQ

Figure 6-7 shows the volume of journal entries per minute that the source system is able to send to the target system when the source system has the 5799-BJC PRPQ installed. It also illustrates the difference when the PRPQ is not installed.

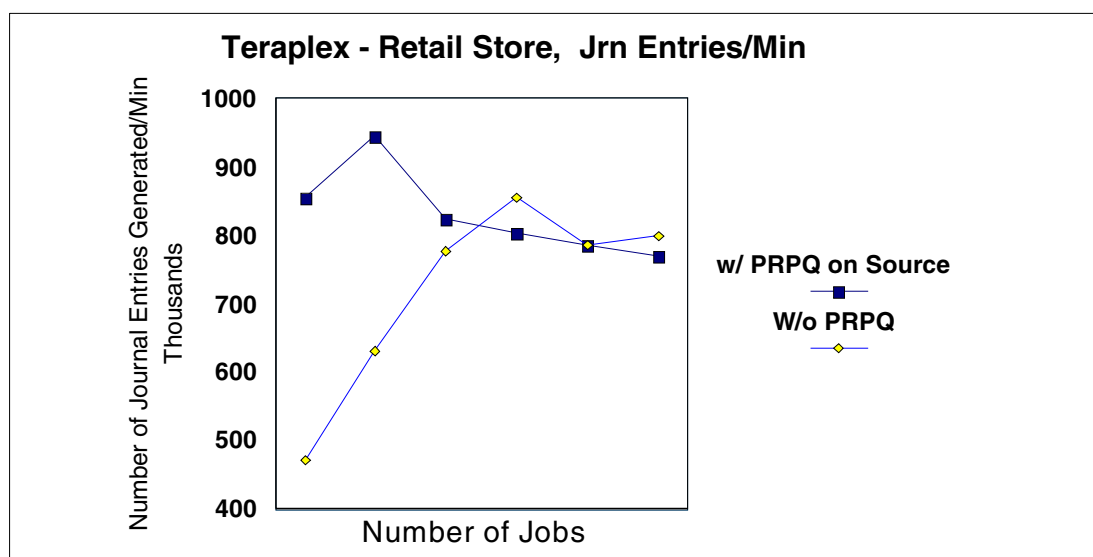


Figure 6-7 Number of journal entries generated per minute - PRPQ benefit

Conclusions

- ▶ The PRPQ improves the source system throughput by up to 50% at peak. (It doesn't take as many jobs to reach the peak with the PRPQ installed and enabled since each job has such fast response time. Hence each job can cycle anew and generate many journal entries per minute.)
- ▶ As the quantity of jobs increases, the natural time-based bundling behavior of journal is almost as beneficial as the extra PRPQ caching for interactive environments like this one. The biggest PRPQ caching benefits ensues well before the peak volume is achieved.
- ▶ This suggests that one need not reserve use of the caching PRPQ only for the most aggressive journal environments but rather that the caching PRPQ is very effective even for less aggressive environments.

6.5.4 Comparison synchronous RJ: send volume versus acknowledge volume

Figure 6-8 illustrates the sent kilobytes/minute flowing from source to target system versus the acknowledged bytes flowing back to the source system confirming arrival of sync remote journal images.

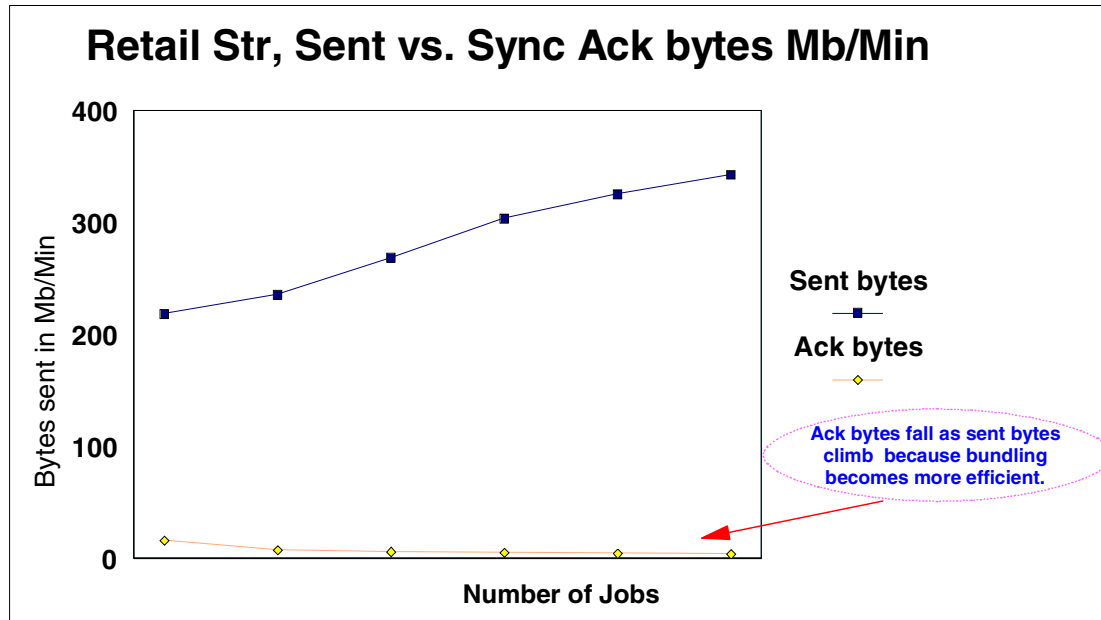


Figure 6-8 Synchronous RJ: send volume versus acknowledge volume

Conclusions

- Remote journal sync response packets flowing back to the source system seem to fill the pipe with no more than 3% or less of the bytes outbound from source to target. They shrink in absolute volume as we ramp up because average bundle size increases and hence less total bundles are sent and acknowledged. This illustrates that adaptive journal bundling is working well and that the journal actually performs better (more efficiently) as you load it up, hence, provided they have sufficient disk arms and IOA write cache, customers ought to be encouraged to journal lots of objects to the same journal.

6.5.5 Comparison among various journal receiver size option parameters

Figure 6-9 shows the number of transactions generated per minute when both the *RMVINTENT and MINENTDTA journal options are activated. Activating both options in a remote journal environment is desirable since both help reduce the quantity of bytes that need to be sent down the pipe. Our investigation was aimed at determining whether the source system (which has to work harder to strip out these extra bytes) would be noticeably harmed by shouldering this extra burden. The good news is that such filtering is a wise choice in a remote journal environment, especially if your in danger of overrunning the capacity of your communication lines.

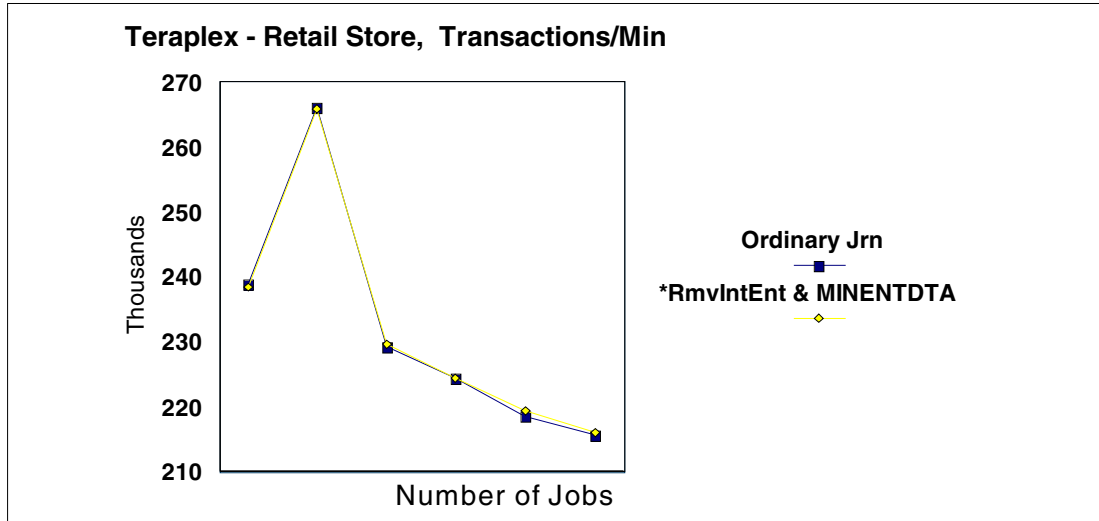


Figure 6-9 Number of transactions generated and sent per minute

Conclusions

- Use of the MINENTDTA journal attribute (when there are no BLOBS to compress) on the source system left the transaction rate almost unaffected, 0.2%, so you might as well turn it on so as to help reduce both the save media for journal receivers and reduce communication line traffic for a remote journal environment.
- Use of the *RMVINTENT setting for the RCVSIZOPT parameter has only modest impact on source throughput (less than 0.1% difference). So you might as well turn it on so as to help reduce both save file size and communication line traffic without risking any noticeable performance harm.

6.5.6 Comparison between remote journal asynchronous and HA solution

Figure 6-10 shows the number of journal entries we were able to generate per minute when using asynchronous remote journaling as our sending/transport mechanism versus using an HA BP like transport mechanism.

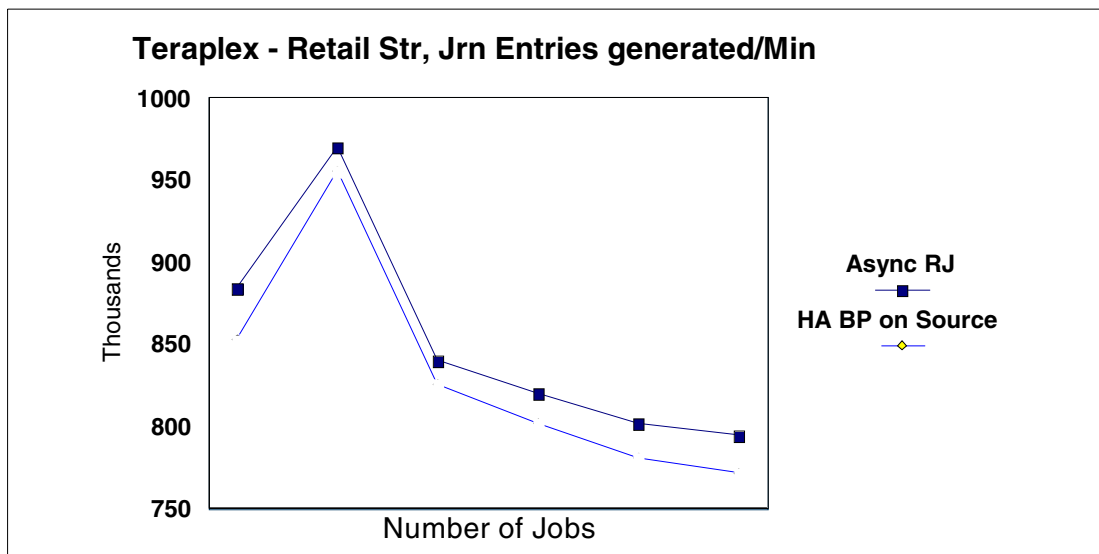


Figure 6-10 Journal entries generated per minute

Conclusions

- ▶ The actual quantity of HA BP overhead experienced on the source system varies depending on which software technologies are employed by your BP. Some elect to perform more aggressive filtering on the source side so as to transport fewer bytes. Others elect to transport the journal entries as rapidly as possible and hence shift the filtering decisions to the target system but end up sending more bytes over the wire. Both approaches have merit depending on your application characteristics. For the particular product we measured in this test, the HA BP harvesting operation on the source system slows down our jobs by only 1.5% at peak. (Async remote journal holds a slight performance benefit over the filtered HA BP approach we employed, but not by much.) Unless your HA BP filters on the source side and unless you also qualify for a substantial reduction in communication line traffic as a result of this filtering and also have a rather constrained communication line, you're probably going to be more satisfied with a new vintage HA BP approach which can tie into the asynchronous remote journal support since this newer approach simply has less chance for excessive latency in the pipe.
- ▶ From a CPU cost point of view, the HA BP load on the target machine employed in order to both harvest from the remote journal and replay to the database files when fed by asynchronous remote journal was about 7.0%, while the remote journal CPU overhead on the target is low at about 0.5 percent.
- ▶ The new vintage remote journal - enabled HA BP products do not incur this source-side overhead.
- ▶ While the particular Retail Store environment we measured in this instance produced primarily simple database adds, updates, and delete operations, we realize that full fledged real-world applications often generate critical changes to lots of non-journaled and non-database objects which need to be replicated by the HA BP software. Lacking direct journal support for some of these objects, the HA BP approach is often to monitor the audit journal and/or sample the object contents itself for evidence of change. Upon detecting such change, they must refresh the replica instance of this non-journaled object. Doing so can consume substantial amounts of resources on the source machine and put a strain on the communication line traffic. We simply did not attempt to replicate a high density of those non-journal HA BP actions during our testing. Hence, while we show HA BP CPU consumption values as low as 1.5% on the source machine for pure database and journal changes, you should recognize that real world examples involving replication of non-database changes often drive CPU consumption much higher. V5R1 introduced the opportunity to start journaling a few popular replicated objects: IFS files, data areas, data queues. Capitalizing on this kind of new journal support should help drive down the source side cost of replication. If additional objects join this select group and become journal-eligible in the future, this HA BP overhead may end up being reduced as well.

Figure 6-11 illustrates the number of bytes sent across the communication line by an HA like solution as the number of jobs increases.

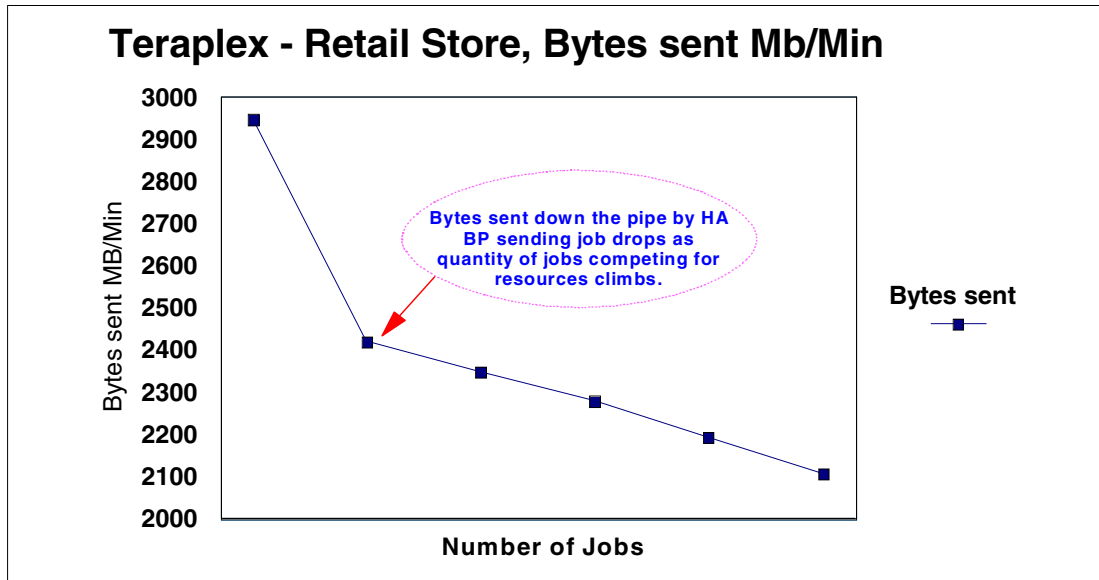


Figure 6-11 HA solution sending environment

Conclusion

- As the number of application jobs ramps up, HA BP jobs can get starved for CPU and can't send as many bytes as are being generated — hence they can fall behind on the source side (which is why use of remote journal is probably a wiser choice to get the journal entries sent promptly).

6.5.7 Unsent journal entries comparing old and new HA BP approach

Another of the comparisons that we performed was to monitor the quantity of unsent database changes that were piling up on the source when remote journaling was being used versus using an HA solution. Figure 6-12 illustrates the environment we measured.

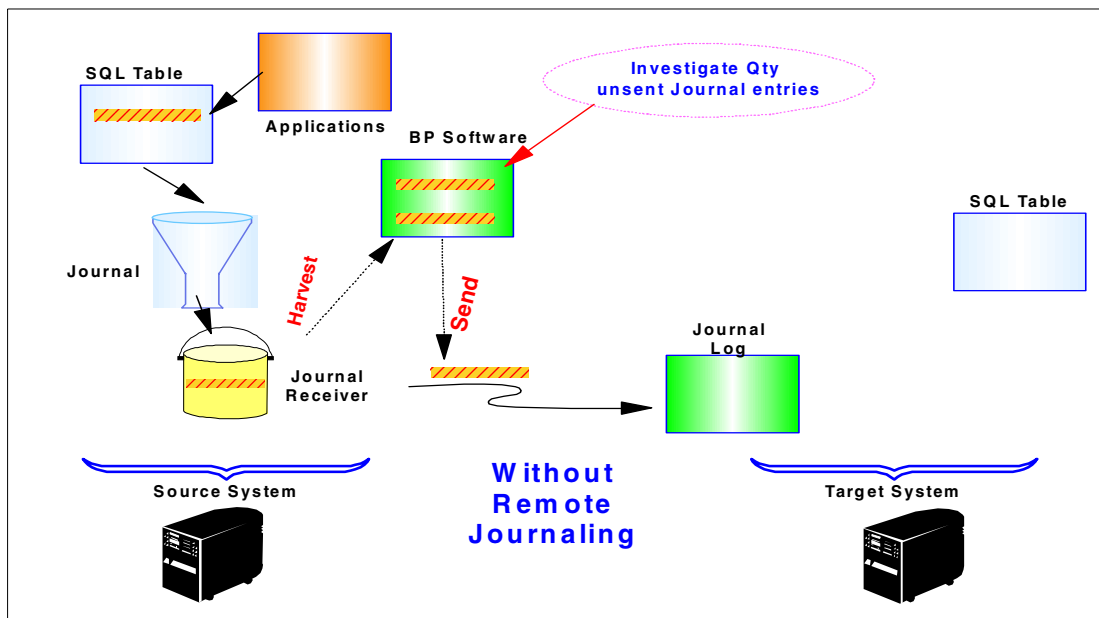


Figure 6-12 Original non-remote journal HA BP solution without remote journaling

Figure 6-13 illustrates the results of this comparison.

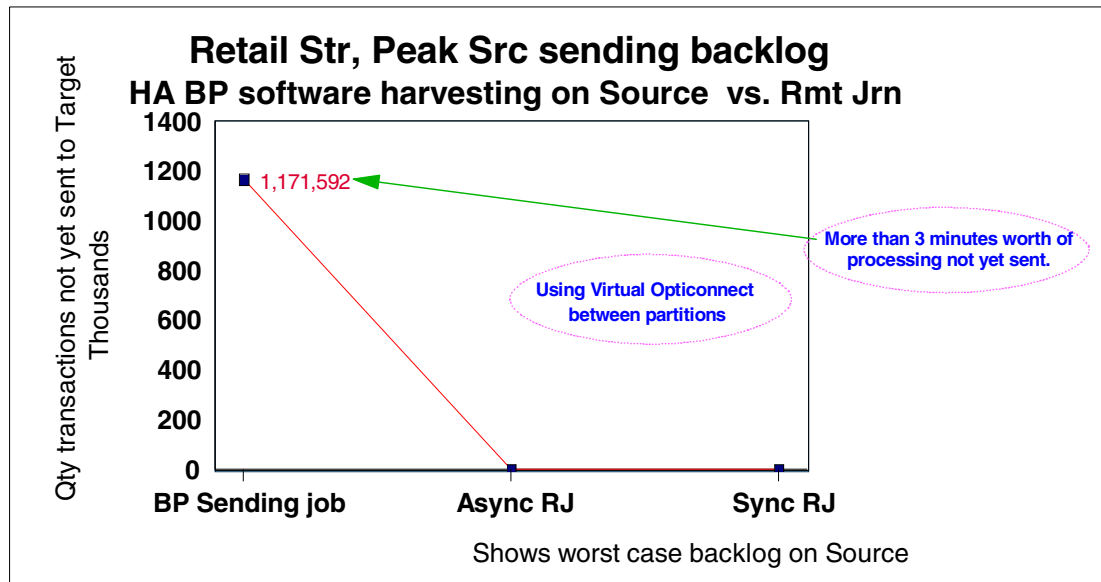


Figure 6-13 Unsent journal entries backed up on the source system

Conclusion

The async remote journal sending task on the source machine with a fast virtual opti-connect communications path between the partitions keeps up and doesn't fall behind even when the transaction rate on the source peaks at 382,998 entries/minute, by contrast the HA BP sending software falls behind as much as 3 minutes worth after a 10 minutes run.

Note: This is not a criticism of the former HA BP harvest and send approach, but rather an illustration of how the HA BP software products which have been enhanced recently to incorporate better plumbing in the form of remote journal support are able to capitalize on the low level microcode level efficiencies inherent in remote journal. Be sure to ask your HA BP provider for a version of their product which is equipped to use the remote journal transport mode.

Both the async and sync varieties of remote journal support have an inherent advantage over any HA BP like traditional approach. The remote journal transport technology doesn't have to incur the overhead to convert the journal entries from internal SLIC representation to external mapped representation, this obviously saves CPU cycles and time. In addition the internal representation takes up fewer bytes per journal entry and hence not as many bytes need to flow down the wire.

To determine how the behavior of async remote journaling related to un-sent bytes, we decided to do the following special test run:

- ▶ 140 jobs running the retail store environment without exhausting the CPU
- ▶ Using an 11-CPU partition in the Teraplex center
- ▶ Async remote journal over 100 Megabit Ethernet line
- ▶ 22,280,812 transactions generated in 10 minutes:
 - 18,167 journal entries per second
 - 91 journal entries per bundle (bundle is the unit sent across the communication line)
 - 1.65 Gig of journal receiver populated during this run
 - 200 journal bundles put on the communication line per second

Conclusion

Not a single one of the journal bundles fell “behind” (each former bundle had been sent by the time the next bundle was populated and handed to the SLIC async sending task 5 Msec later).

These remarkable results illustrate how effective and efficient the remote journal approach can be.

Rule of thumb: If you’re currently using an original non-remote journal HA BP product which still uses the less efficient plumbing, you’ll want to ask your HA BP about a version of their software which supports remote journaling.

6.5.8 Conclusions for HA BP solution

In Figure 6-14 we summarize several measurements affecting HA solutions. Figure 6-14 also illustrates the observations that we were able to establish from our runs.

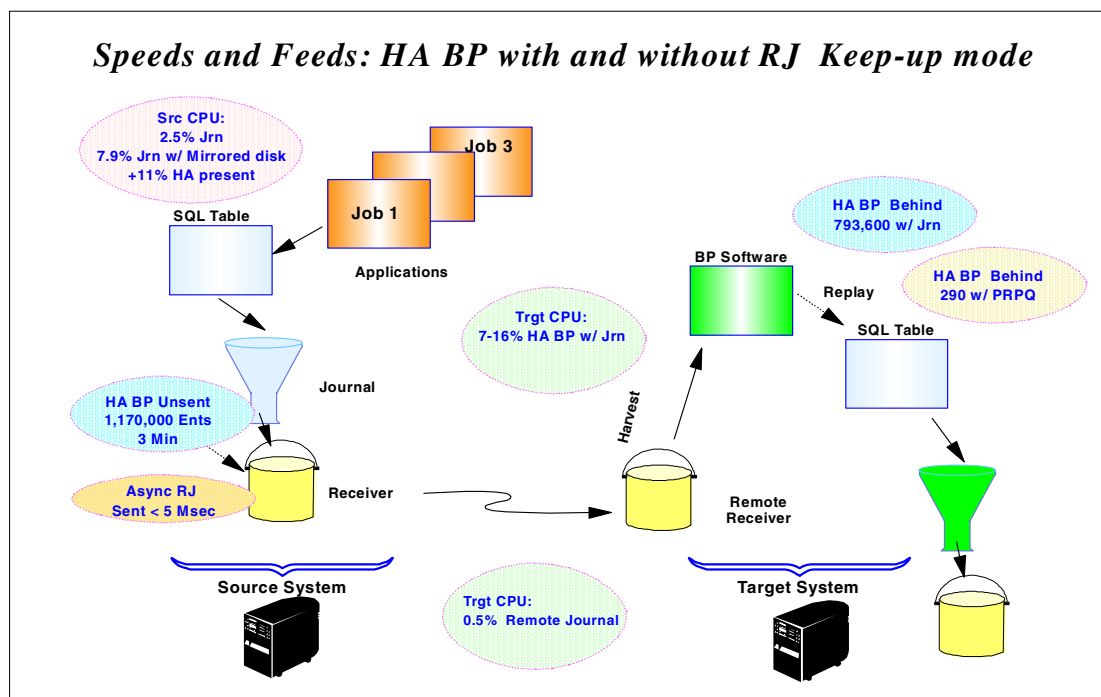


Figure 6-14 HA BP versus remote journal conclusions

- In a high-bandwidth non-filtered environment, the newer remote journal enabled HA BP approaches can send more transactions per minute than the original non-remote journal HA BP software. Remote journal is simply better optimized. In an environment with significant communications bandwidth, remote journal can move more data with less impact on the communications path. It should be noted that by using the optimized remote journal approach on average less than 2% of the virtual opti-connect bandwidth was being used during these tests even when 11 CPUs were busy generating journal entries. In an environment with limited communications bandwidth and a higher filtering requirement of journal transactions, it is possible that an HA BP approach which provides filtering support can similarly reduce the communications bandwidth requirement.
- With the PRPQ installed on the target system, the HA BP database replay phase was able to stay current with the source system. The CPU utilization rate on the target system of 7% - 16% was consistent whether the CPU utilization rate on the source was in the 40% range

or 95% or greater range. Therefore, for environments that have heavy transactions volumes, the use of remote journal along with the Journal Caching PRPQ on the target journal will probably give you the greatest throughput capability.

- ▶ The use of the Journal Caching PRPQ increased overall throughput on the both source and target system but used slightly more CPU in the process. In an environment that experiences performance bottlenecks due to the disk write intensive journaling overhead, the Journal Caching PRPQ can be used to improve overall throughput capabilities. In an HA environment, the use of the Journal Caching PRPQ on the target system is an excellent combination of data protection, improved fail-over time and higher throughput capability.
- ▶ It is important to point out that while we measured only a modest 11% or less extra CPU overhead on the source system due to the presence of the HA BP software for this particular *simple* Database replication environment, that more realistic and challenging real world replication environments using HA BP software must also consume cycles monitoring, sampling, copying, saving, and sending non-Database objects many of which can be numerous and substantial in size. As a consequence, far *more* than 11% of the CPU can be consumed performing such services. Recent versions of most HA BP products now support replication of Stream File objects via journaling which is a more modern and efficient approach than the former practice of cloning such objects each time they are changed. We suggest that customers who are utilizing an HA BP product to replicate Stream Files within the IFS, cease using the old less efficient technology and instead ask their HA BP for a version of the product which capitalizes on the opportunity to journal such objects. This change alone saves both communication bandwidth and CPU pathlength.
- ▶ It is important also to point out that folks probably can't rely upon a pure remote journal approach (because it only records the changes to database, stream files, data areas, and data queues) and thus may need to enlist the help of an HA BP product if they place critical application data into objects which are *not* journal eligible.
- ▶ The benefits derived by jointly using *both* the remote journal support and an RJ-enabled version of an HA BP product in harmony are that by having an HA BP product present to monitor the RJ environment and the health/status of the communication line the automated HA BP software can *quickly* respond to any communication line failures, vary the lines back on, even switch over to use of alternative lines if they exist and automatically restart the RJ environment. Hence, remote journal and HA BP are not competing technologies but rather complimentary technologies designed to work well together.

6.5.9 Catch-up mode tests

Once a failed source system has been repaired the remote journal function can be employed to transmit the journal delta data needed to re-sync the source tables (which now can be down level) with the more modern replica tables from the target system. During this critical phase the system replicates large quantities of pent-up pre-existing journal entries as quickly as possible. This is referred to as *catch-up mode*. Figure 6-15 illustrates catch-up mode. Once the source system has been repaired the remote journal catch-up mode will help speed up the swap back to the production server.

Remote Jrn: Swap Back to Production (**Catchup mode**)

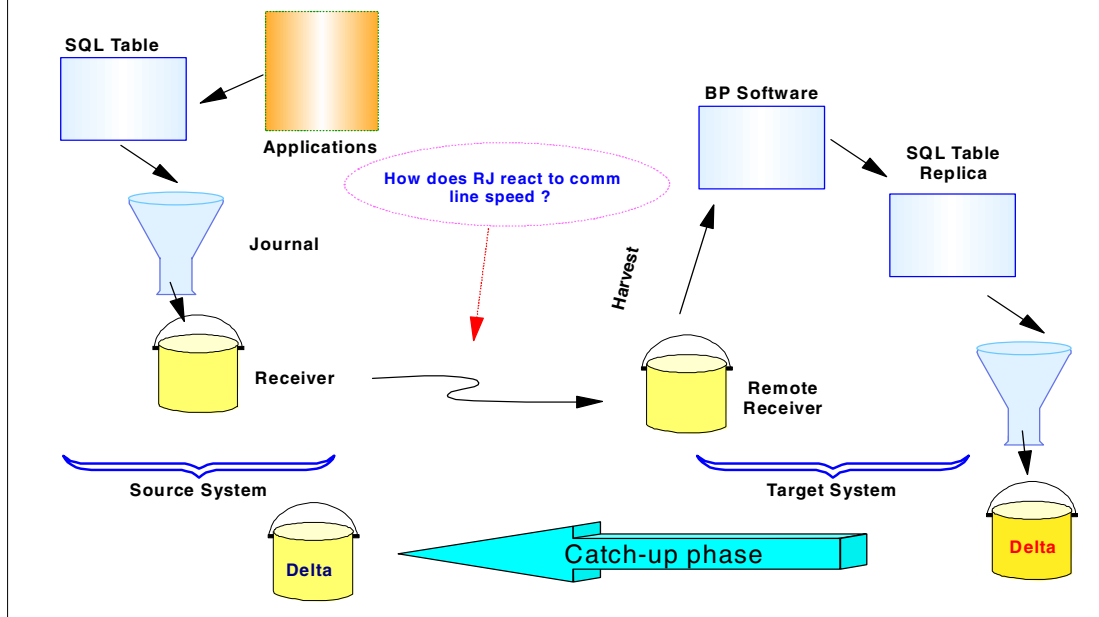


Figure 6-15 Remote journal catch-up mode

We elected to investigate the speed of this critical catch-up phase. These tests were done using a variety of communication hardware speeds. Figure 6-16 illustrates the number of MB/minute transmitted that we were able to achieve during remote journal catch-up mode using various different communication speeds.

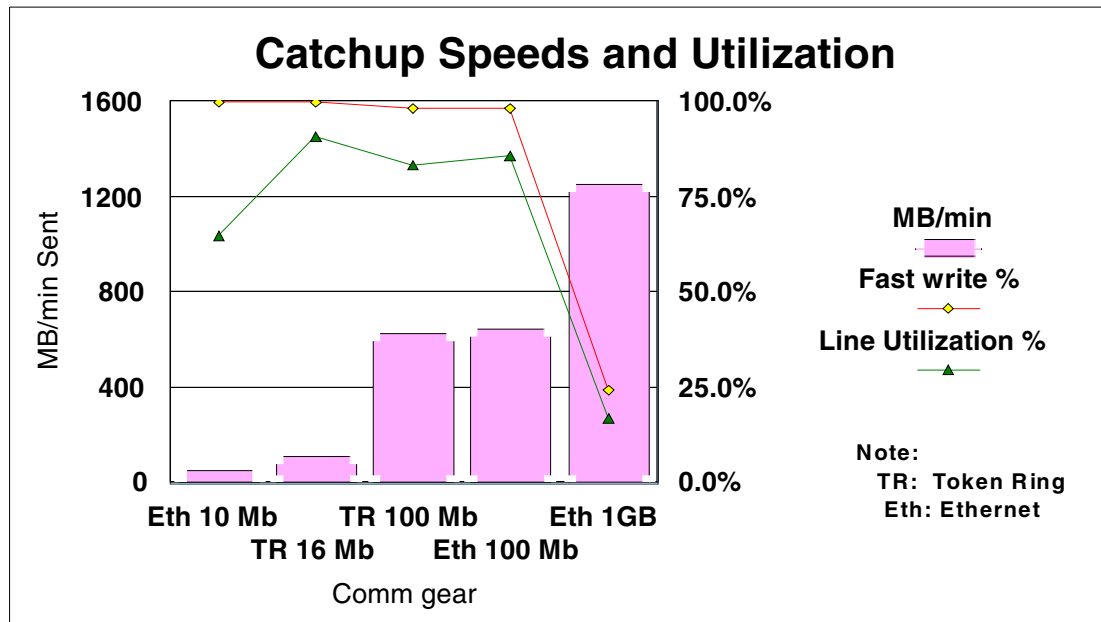


Figure 6-16 Catch-up speeds and utilization

Conclusion

As evidenced by the sharp drop in fast write percentage, once we employ the highest speed lines, the communication gear ceases to be the bottleneck and the rate at which disk writes can be serviced on the source machine we're trying to refresh becomes the limiting factor. Communication rates of 1,000 Megabits per second overrun the write cache suggesting that you'd be wise to relieve this bottleneck by configuring fewer disk drives per IOA thereby increasing the quantity of write cache available to service each disk arm during aggressive write operations such as those which ensue for catch-up mode.

Rule of thumb: Calculate your fast write percentage. If you're achieving less than 90% fast writes, you'd probably benefit by configuring more write cache per disk arm.

Optimum disks attached to an IOA

We also wanted to determine the optimum number of disks that should be attached to an IOA. Figure 6-17 shows the results of the tests performed using remote journal in catch-up mode.

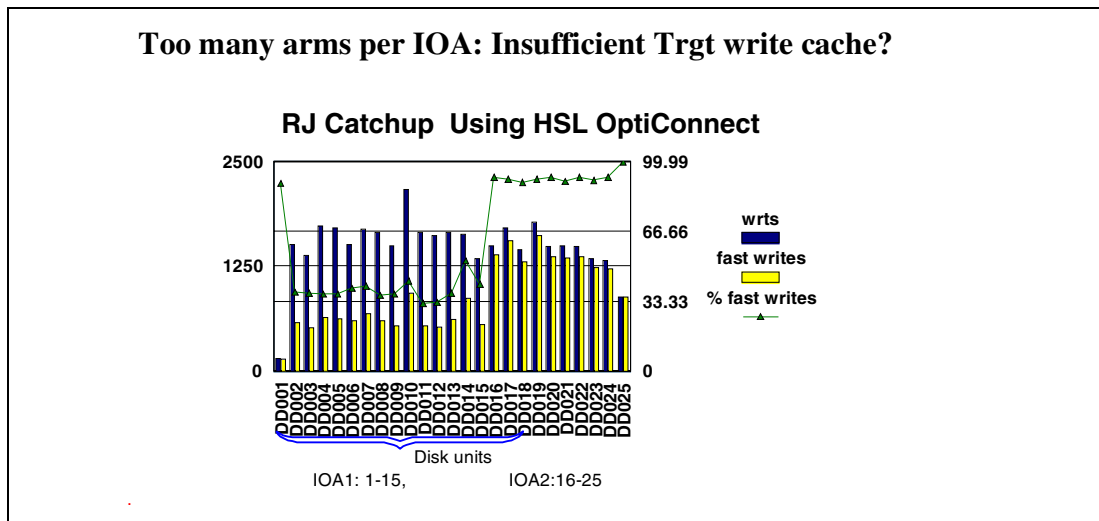


Figure 6-17 How many disks per IOA

Conclusion

The first 15 disk arms were all attached to the same IOP with 104 MB of write cache in its IOA. Clearly even 104 MB of write cache is not enough when all 15 disk drives are servicing heavy journal traffic during catch up mode. On the other hand, the remaining 10 disk drives attached to a separate IOA are achieving nearly 95% fast writes, so 10 arms are not too many. Our results would suggest that you may want to consider capping the number of journal arms per IOA at 10 and not step up to the full 15 an IOA lets you physically configure unless your journal rates are more modest.

Rule of thumb: Although the hardware allows you to attach up to 15 disk drives to the same IOP, high rates of journal activity can swamp the associated write cache in the IOA. Limiting yourself to no more than 10 such disk drives per IOA will help insure less likelihood of overrunning the write cache and will thereby give better journal performance.

Note: This rule of thumb applies only to the disks housing your journal receivers and only if the rate of arrival of journal entries is rapid enough to overrun your IOA write cache. For less aggressive disk write environments, more than 10 disk arms might be perfectly acceptable. The best way to make your optimal configuration selection is probably to collect some disk and IOA performance statistics during the most journal intensive part of your day and then employ the algorithm documented in the Appendix which shows you how to determine your percentage of so-called *fast* writes being service by the IOA write cache. (See Appendix A, “How to calculate the fast write percentage” on page 155). If your resulting percentage of fast writes is in excess of 90%, you probably have sufficient write cache configured. If below 90% you might want to either reduce the number of disk arms attached to this IOA, thereby distributing more write cache per arm, or step up to a newer model of IOA which provides a larger quantity of overall write cache.

Using ESS disks

We decided to conduct some tests with the IBM Enterprise Storage Server (ESS).

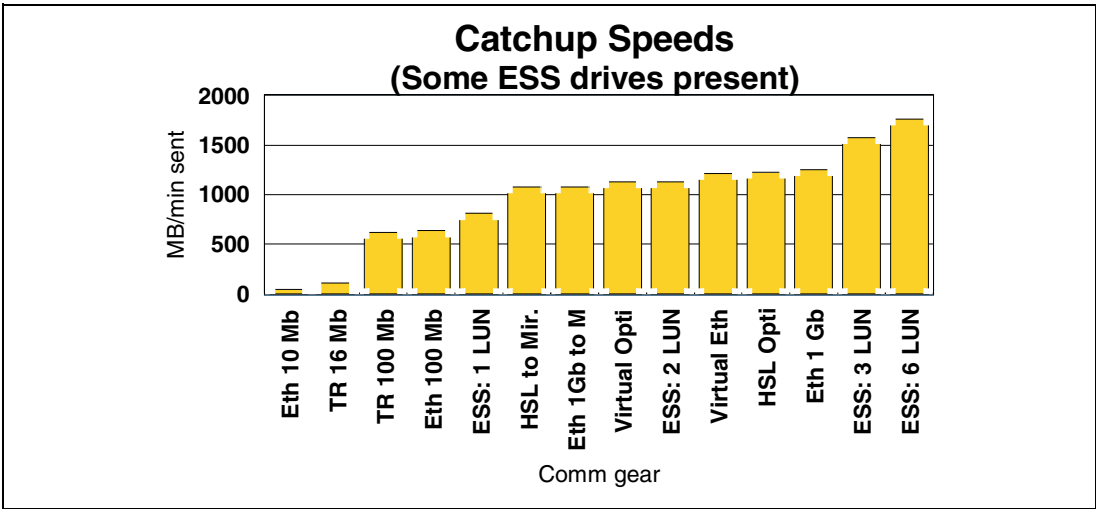


Figure 6-18 Catch-up speeds during catch-up mode

Conclusion

Remote journal catch-up mode responds well to the presence of a faster pipe between the source and target machines until the speed of the disks on the destination machine become the bottleneck.

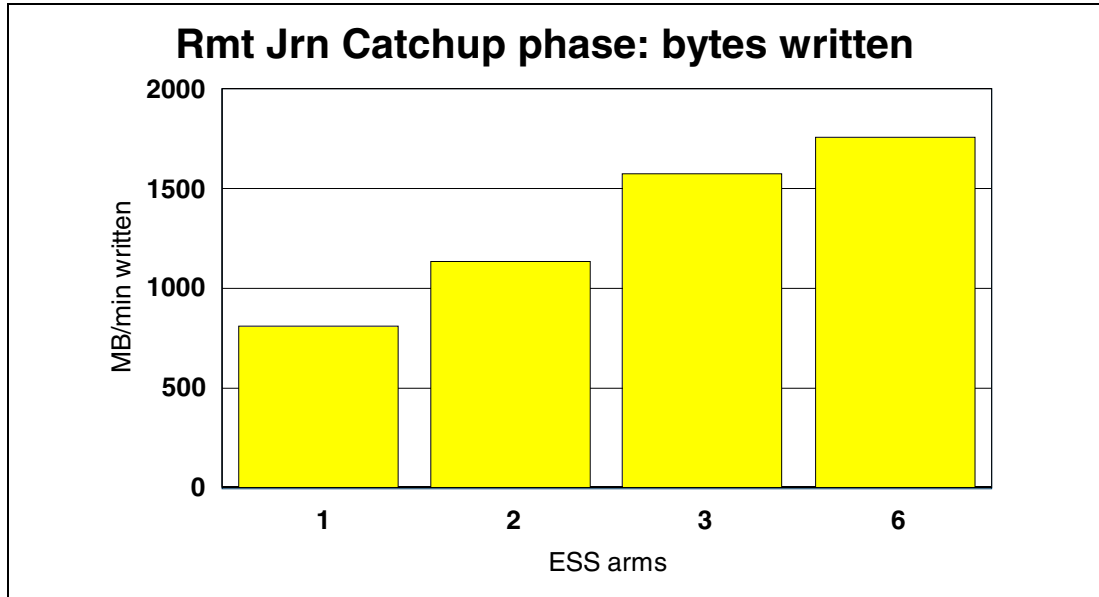


Figure 6-19 ESS arms on the target system

Conclusion

The number of logical ESS arms (LUNs) configured on the target machine during remote journal catch-up mode has a direct impact on the efficiency of the target machine's forcing task (the SLIC support which completes the catch up process by writing journal entries to disk). Increasing the quantity of virtual disk arms is definitely a helpful configuration approach when using ESS (probably because it similarly increases the quantity of write cache at our disposal).

Rule of thumb: If you elect to use ESS drives, configure as much write cache as practical to service the ASP housing your journal receiver even if this means configuring more LUNs.



Part 3

Conclusions and recommendations

In this part we describe the conclusions and recommendations of our findings from the tests that we conducted. It represents a summary of the best journaling practices.



Conclusion

Throughout the previous sections of this redbook, we have introduced you to a number of concepts and options that could potentially have an impact on the performance of your iSeries server when you activate journaling on your database tables, IFS files, and data queues or data areas.

We also showed the measured impact of some of these factors in a real customer environment and indicated a few of the reasons for our measured findings.

In this section, we will revisit some of these issues and also share with you a few general recommendations that could further assist you in reducing the impact of journaling on your iSeries server.

7.1 General performance considerations

As you will notice from the testing scenarios and findings described in the previous sections, journaling definitely can have an impact on the performance of your system. The quantity of the impact is up to you and directly influenced by the hardware choices, journal tuning choices, and application design principles you embrace. Journal overhead can be as high as 50% if you tune poorly and perhaps as low as 3-5% if you choose wisely.

The following is a brief summary of issues that you should consider in your attempts to reduce the impact of journaling on your iSeries. A number of these recommendations have previously been employed by customers who have participated in benchmark exercises. Each technique has successfully reduced the impact of journaling on their systems.

Hardware and ASPs

► IO Adapters and write cache

Having enough IOA write cache is more important than nearly any other tuning decision. Where possible, install the newest technology IO Adapters (IOAs) with large write cache. Tests illustrated that the new technology IOAs with its large write cache capabilities reduced most of the performance bottlenecks when compared to older IOA technology.

Also, unless you have these newest 26 MB technology IOAs installed, spreading the disks within your ASPs over multiple IO processors and limiting the number of disk arms to no more than five disk arms per IOA, will further assist in reducing the possibility of write cache overflow.

► User auxiliary storage pool (ASP):

Configuring private User ASPs to house journal receivers remains the preferred methodology for folks who have a high volume of journal traffic and want to assure optimal performance. Obviously, such a User ASP must have sufficient disk arms and IOA write cache at its disposal. Isolating a single disk drive as your User ASP is rarely sufficient. The higher your journal volume, the more disk arms you're going to need to provide sufficient bandwidth. The benefit of private disk arms set aside in this fashion to service journal receiver traffic and only journal receiver traffic is especially important if the disks back in the system ASP are already operating at near peak capacity due to other work taking place on your machine. Since we conducted our journal performance tests without such other non-journal disk traffic from unrelated applications unleashed simultaneously against the disk arms in the System ASP, our particular tests might lead you to conclude that the performance benefit associated with User ASPs are minimal. Obviously, that's not a realistic or proper conclusion in some shops. Instead, you'll want to look at your overall disk traffic pattern and disk write volume as well as your percentage of fast writes to your IOA as you weigh the performance trade-offs associated with the User ASP versus System ASP decision. Ultimately, what you should keep in mind is that journal performs best when given a sufficient quantity of disk arms across which to spread its traffic.

► Disk protection:

If your iSeries server is equipped with the new technology IO adapters equipped with a large write cache, parity protected ASPs no longer present much of a performance bottleneck and they are generally a better performing choice than mirrored ASPs.

Journal options

► Journal Caching PRPQ (5799-BJC)

You have been exposed to this feature and its performance advantages throughout this redbook. For batch jobs, most of the known journaling performance issues and bottlenecks can be reduced by installing this very effective feature.

► Number of journal receivers:

Creating multiple journal receivers in the same ASP could potentially contribute towards degraded performance. This is mainly due to potential disk arm contention, especially if your journaling traffic is very high.

► Omit journal entries:

The journal parameters *OMTJRNE(*OPNCLO)* and *IMAGES(*AFTER)* greatly assist you in reducing the number of journal entries written to your journal receivers, which in turn leads to smaller journal receivers, less IO operations and less storage requirements.

You should however verify that this information is not required by any of your other applications, for example auditing systems, before employing these journaling options.

The journal manager does however attempt to reduce the performance impact if both the before and after images are written to your journal receivers. It does this by scheduling a single write operation for both images at once, thereby reducing the number of IO operations.

► Manage journal receivers

You can reduce the impact of changing journal receivers by setting the journal parameter *MNGRCV(*SYSTEM)*. Also, if your journal receiver was created in a user ASP, this reduces the possibility of the journal receiver overflowing into the system ASP when the user ASP threshold had been reached, which in turn could potentially lead to performance degradation.

When this parameter is set to *MNGRCV(*SYSTEM)* instead of *MNGRCV(*USER)*, you can react more rapidly to the need to issue a CHGJRN operation, because the system initiates the CHGJRN operation for you rather than relying upon a human operator to react or a user program to respond.

Set the receiver threshold higher to increase the number of arms written in parallel and to reduce frequency of changing to a new attached receiver (with all the overhead that requires).

► Journal receiver size options:

The journal parameter *RCVSIZOPT(*RMVINTENT)* forces the system-generated IPL critical journal entries to be written to disk arms other than those used to write your user journal entries and also recycles disk space consumed by system-related journal entries that are no longer required for recovery purposes. This reduces the possibility of disk arm contention and also reduces the size of your journal receivers.

The journal parameter *RCVSIZOPT(*MINFIXLEN)* reduces the size of your journal entries, also resulting in improved performance. You should however verify that this information is not required by any of your other applications, for example high availability software or auditing systems, before employing this option.

Use *RCVSIZOPT(*MAXOPT2)* to enable the journal to spread across up to 100 arms and support a larger receiver threshold.

► Saving journal receivers

If possible, do not save journal receivers while still attached and being written to. Failure to honor this protocol will cause the applications writing to the journal receiver to wait until the save operation is completed or a save boundary has been reached before processing can continue, resulting in a decrease in throughput and an increase in overall runtime.

Application techniques

- ▶ **Force-write-ratio (FRCRATIO):**

The performance of your system could be severely impacted if you specify a value of 1 for the FRCRATIO parameter on a journalled database table. This parameter takes precedence over most of the journaling parameters and will therefore force a disk write operation, even if the journal manager is attempting to bundle the writes in memory.

Use of the FRCACCPH parameter for SQL indexes and keyed logical files should similarly be avoided if you seek optimal journal performance.

- ▶ **Parallelism:**

Split your long-running batch job into multiple parallel jobs streams if possible. This reduces overall runtime and could lead to more efficient CPU utilization as well as better journal bundling.

- ▶ **Specify Sequential Only on OVRDBF**

Where possible, use the CL command *OVRODF* to specify sequential-only processing when adding rows to your database tables. Also, consider using separate open data paths (ODPs) and open these for output when adding large amounts of rows to your database tables.

Also, specify the *NBRRCDS* parameter with a value as close to 128KB/row-width as possible to maximize memory buffer utilization. If you make use of commitment control, issue a *COMMIT* no more often than each 128KB boundary as well if practical.

- ▶ **Keep database tables open throughout your application.**

Opening and closing database tables repeatedly is an expensive operation from a performance point of view. Try to keep database tables open for further processing in subsequent steps.

- ▶ **Limited journaling**

One of the common mistakes first time users of HA BP software and first time journal users make is to go overboard. They blindly journal every file in sight without stopping to make prudent decisions. Frankly, it's probably unlikely that every file in your shop deserves to be journaled and replicated. Since journaling puts some performance burden on your application, it makes far more sense to pause and analyze the purpose and use of each physical file rather than blindly enabling journaling for all files in your application library. If you don't feel comfortable performing this analysis yourself, you may want to enlist the help of IBM Global Services folks or some of the HA BP field service advisers. Both can help walk you through this analysis process. As general guidelines, you'll want to refrain from journaling or replicating the files that your application wouldn't pick up and re-use if you needed to fail over to a backup machine. The files you probably don't really need to journal include: Transient tables, Work files, and Temporary files. Unless you have a very unusual and heroic recovery strategy built into your application, most of these kinds of files are neither the ones you'd use in an HA BP failover scenario, nor the ones you'd expect to find and employ even in a single-system recovery environment. Hence, why bother incurring journal overhead by protecting what you're probably going to toss away?

- ▶ **Access path recovery**

Set a realistic SMAPP value. In a high availability environment, consider setting a higher access path recovery time on the target system while this system is in stand-by mode. This will reduce the overhead while the journal entries are replicated and the transactions are being reproduced on this target server. You can always set this value lower when you do a role-swap.

- ▶ Parallel Access Path Maintenance

Also, use the CHGQRYA command to improve performance if you have database tables with a large number of access paths and use *SEQONLY(*YES)* processing where possible.

Remote journal

- ▶ Ask your HA BP provider for a version of their product which supports remote journaling and enable this support. Try synchronous remote journaling first. If the performance impact is too great, back off to asynchronous remote journal.
- ▶ Where practical, consider use of async remote journal during batch periods and sync remote journal during critical interactive periods.
- ▶ Keeping transmitted journal entries current

Our measurements demonstrated that both varieties of remote journal (even the async flavor) sent the journal entries across the communication line in a timely fashion. Neither experienced any measurable delay in getting the new journal images on the wire. This stems in part from the fact that the sending operation is managed at the microcode level in real time as soon as your application deposits new journal entries. By contrast, any approach which must wait until alerted that a new journal entry has arrived, then seek to be scheduled to use the CPU, then harvest the new journal entries, and finally filter out the interesting ones and place them on the wire, obviously will have more path length and a natural delay of some magnitude. The non-remote-journal-enabled versions of most HA BP products fall in this category and though they try heroically to keep current, it's difficult for them to achieve this objective as well as remote journal, especially when the volume of journal traffic is as high as we employed during these tests.

Similarly, most HA BP products try valiantly to stay current within their replay jobs on the target system. Here too they face some rather steep obstacles, often having one replay job attempting to replay what thousands of interactive jobs back on the source system are able to generate in a short burst of time. If you discover that your HA BP replay jobs on the target system are having a tough time keeping up, you may be able to help them out by enabling the Journal Caching PRPQ on the target system.



A

How to calculate the fast write percentage

In this appendix we describe the steps required to calculate the fast write percentage for the IOA write cache that was used throughout some of the graphs.

Instructions

Fast writes are recorded by collection services.

Notes:

- ▶ You must have V5R1 hardware or later for this analysis approach to work, otherwise all writes are erroneously reported as *fast writes* regardless of whether the cache could accept them immediately.
- ▶ You need the performance tools (5722PT1) to start collection services.

1. Start collection services, if it isn't already running (if it is running, skip to Step 2). Type the following command on a 5250 session.

GO PERFORM

Select Option 1 to start collection services.

2. Collect for *at least 5 minutes* (disk statistics are gathered at this interval).
3. Dump the statistics to a database table using the following CL commands:

CRTPFRDTA FROMMGTCOL(*ACTIVE) TOLIB(LIB) CGY(*DISK)

This should create a file called QAPMDISK.

4. Query the file. The data is in columns DSCCFW (fast writes) and DSWRTS (total writes) in table QAPMDISK. Type the following CL command:

STRSQL

SELECT 100*SUM(DSCCFW)/sum(DSWRTS) FastWrtPrct from lib/qapmdisk

The result is the percentage of writes that were immediately accepted into the write cache (they were fast).

5. Optionally stop collection services.

GO PERFORM, Option 2, Option 2

Related publications

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

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 158.

- ▶ *AS/400 Remote Journal Function for High Availability and Data Replication*, SG24-5189
- ▶ *IBM @server iSeries Handbook Version 5 Release 1*, GA19-5486-21
- ▶ *LPAR Configuration and Management with iSeries Logical Partitions*, SG24-6251
- ▶ *Clustering and IASPs for Higher Availability*, SG24-5194

Other resources

These publications are also relevant as further information sources:

- ▶ *iSeries Backup and Recovery*, SC41-5304

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ iSeries Backup and Recovery
<http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/books/c4153045.pdf>
- ▶ Batch Journal Caching PRPQ
<http://www-1.ibm.com/servers/eserver/iseries/ha/hajournal.htm>
- ▶ Teraplex Center
<http://www.as400.ibm.com/developer/bi/teraplex/>
- ▶ Clustering Redpapers
<http://www.redbooks.ibm.com/redpapers/pdfs/redp0111.pdf>
- ▶ IBM Software Knowledge Base
<http://www-912.ibm.com/supporthome.nsf/document/10000051>
- ▶ Teraplex Centers for Business Intelligence Solution Integration Test
<http://www.ibm.com/solutions/businessintelligence/teraplex>
- ▶ Clustering and IASPs for Higher Availability, SG24-5194
<http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245194.pdf>

How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

ibm.com/redbooks

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

Numerics

5799-BJC 97
5799-BJC PRPQ 23
 performance benefits 23

A

access 58
access path recovery 152
access paths 84
 MAINT(*REBLD) 84
ALLOCATE(*YES) 115
ALTER TABLE 26
application 52
application tuning 109
Apply Journal Changes Extended PRPQ 24
APYJRNCHG 20, 89
APYJRNCHGX 21, 25
 DDL journal entries 25
 DDL operations 25
ASP 6, 33, 58, 84, 129, 150
async 118
asynchronous delivery mode 125
asynchronously maintained remote journal 125
Atlanta 44, 47
auxiliary storage pool (ASP) 18, 58

B

batch 110
batch job parallelism 111
Batch Journal Caching PRPQ 5, 21
bundling 15

C

caching 22
catch-up mode 124
Change Logical File command 112
Change Physical File command 112
changing journal receivers 87
CHGJRN 8, 84–85
 MINENTDTA 9
 RCVSIZOPT 85
 RCVSIZOPT(*RMVINTENT) 10
CHGQRYA 116
CHGQRYA command 153
CLRPFM 118
coalescence 17
Combined Function I/O Processors (CFIOP) 31
COMMIT 152
commitment control 118
communication infrastructure 132
concurrent writes 116
configuration 64

CPU 68, 97
CPU consumption 68
Create Logical File command 112
Create Physical File command 112
CRTJRN 84

D

data areas 19
data queues 19
database 99
database indexes 112
database table extensions 114
database writes 69–70
DB2 Symmetric Multiprocessing for OS/400 60
delivery mode
 asynchronous 125
 catch-up 124
 synchronous 126
DFU 84
disk 69, 75, 98
disk protection 74
 mirrored ASP 74
 RAID-5 74
DSPRCYAP 84

E

edit 59
EDTRCYAP 82
 SMAPP settings 82
elapsed 66
elapsed runtime 65
encoded vector index 84
end journaling 20
ENDJRN 20
ENDJRNOBJ 20

F

failover scenario 152
filtering support 126
force-write-ratio (FRCRATIO) 152
FRCACCPH(*YES) 84

H

HA BP product 126
HA BP replay 132
HSL OptiConnect 42

I

I/O wait 70, 101
ICPW 110
IFS objects 5, 19
 Directories(*DIR) 19

- Stream Files(*STMF) 19
- Symbolic Links(*SYMLNK) 19
- IMAGES(*AFTER) option 99, 151
- increasing performance 111
- input/output adapter (IOA) 16
- input/output processor (IOP) 6
- Interactive Commercial Processing Workload 110
- IO adapters 150
- IOA 6, 58, 84, 120, 135, 150
- IOP 67, 143
- ITSO 51

J

- job processing types 110
- job thread 112
- JOECRA 87
- journal bundle 16
- Journal Caching PRPQ 97
- journal changes 137
- journal deposits 69
- journal overhead 103
- journal parameters 17
- journal receivers 8, 151
- journal threshold 7
- journaling
 - data areas 5
 - data queues 5
 - ending 19
 - IFS directories 5
 - non-database objects 19
 - starting 19

L

- Licensed Internal Code (LIC) 124
- logical 70
- LPAR 32

M

- Malmo 45
- managing journal receivers 87
- MAX1TB 113
- Mbps 128
- megabits 128
- millisecond (ms) 76
- MINENTDTA 8, 97
- MINENTDTA with BLOBs 105
- MINFIXLEN 8
- minimize entry-specific data 87
- mirroring 6
- MNGRCV 8
- MNGRCV(*SYSTEM) 87
- MNGRCV(*USER) 151
- Multiple Function I/O Processors (MFIOP) 45

N

- NBRRCDS 70

O

- ODP 12, 70, 115, 152
- OMTJRNE 9
- OMTJRNE(*OPNCLO) 99, 151
- open data path (ODP) 115
- Operations Navigator 29
- optical HSL 42
- OVRRBF 12, 70
 - NBRRCDS 12
 - SEQONLY(*YES 12
 - SEQONLY(*YES) 70
 - Sequential Only 152

P

- packets 135
- parallelism 111
- partition 35–36, 40
- Performance Collector 98
- Performance Tools 61

Q

- QBATCH subsystem 110
- QINTER subsystem 110
- QPFRAJ 61
- query 84

R

- Raid 6
- RAID-5 protection 38
- RCVSIZOPT 7, 84
 - *MAXOPT1 86
 - *MAXOPT2 86
 - *MINFIXLEN 85
 - *RMVINTENT 85
- recovering at a hot site 93
- recovering from a hot site 104
- recovery 59
- Redbooks Web site 158
 - Contact us xv
- remote journal 7, 126
 - async 153
 - asynchronous 123
 - catch-up mode 123
- remote journal (RJ) 123
- remote journal catch-up processing 7
- remote journaling 123
 - synchronous 123
- remove 86
- removing internal journal entries 10
- revolutions per minute (rpm) 6, 62
- RMVINTENT 84, 97
- runtime 66, 96

S

- saving journal receivers 151
- SAVOBJ 124
- secondary 46, 48

- secondary partitions 35
- SEQONLY 12
- SETOBJACC 110
 - *PURGE option 110
- SLIC xiv, 11, 69, 82, 98, 113, 128
- SMAPP xiii, 11, 59, 82, 84, 152
- SMP 60, 112, 115
- source system 135
- SQL 84
- start journaling 19
- STRJRN 19
- STRJRNAP 11, 84
- STRJRNOBJ 19
- STRJRNPF 9
- switchover 127
- symmetric 59–60
- symmetric multiprocessing (SMP) 90, 115
- sync 118
- synchronous delivery mode 124, 126
- synchronous write operation 18
- synchronously maintained remote journal 126
- system 59, 61, 90–91
- system ASP 7
- system licensed internal code (SLIC) 69
- System Managed Access Path Protection (SMAPP) 11, 82
- system value 116
 - QRYDEGREE 116

T

- total 102
- total writes 101

U

- user 37, 39, 41, 65
- user auxiliary storage pool (ASP) 7, 150

V

- virtual LAN 32

W

- write 76
- write cache 6, 16
- write-ahead journaling 5
- WRKACTJOB 110
- WRKHDWRSC 77



Redbooks

Striving for Optimal Journal Performance on DB2 Universal Database for iSeries

Discover which journal tuning parameters make the biggest difference

Learn the impact of journaling on batch processing

Discover the latest on remote journaling performance

Journaling is an inherent feature of OS/400 on the iSeries that has been around since the early releases of the System/38. The system employs journals to ensure data integrity and recoverability and this support can further be extended to reduce the duration of an abnormal system end. Journaling also serves as the cornerstone upon which many data replication and data warehouse refresh schemes are built.

The aim of this IBM Redbook is to provide you with an understanding of the factors which influence journal performance on the iSeries and to help you identify which hardware options and software settings you can employ to minimize this impact. We will also share with you some ideas on how to minimize the impact of journaling on your iSeries from an application point of view.

We also explain the remote journal function on the OS/400, which offers a reliable and fast method to transfer journal entries to a remote iSeries server. We performed tests concerning remote journal performance and tested this function in different environments.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6286-00

ISBN 073842319X