

Parallel Sysplex Application Considerations

Introducing architecture mindset to
leverage z/OS sysplex applications

Designing applications to benefit
from Parallel Sysplex

DB2, DFSMSdfs, CICS, IMS,
WebSphere MQ applications



Jordi Alastrué i Soler
Mak Anders, Karl Bender
Amardeep Bhattal, Paolo Bruni
Andy Clifton, John Iczkovits
Franck Injey, Pirooz Joodi
Vasilis Karras, Andy Mitchelmore
Robert Queen, Mark Rader
Mayur Raja, Pete Siddall
Jens Erik Wendelboe



International Technical Support Organization

Parallel Sysplex Application Considerations

October 2004

Archived

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

Archived

First Edition (August 2004)

This edition applies to IBM Parallel Sysplex technology used with operating systems z/OS (program number 5694-A01) or OS/390 (program number 5647-A01.)

© Copyright International Business Machines Corporation 2004. All rights reserved.

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

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this redbook.	ix
Become a published author	xi
Comments welcome.	xii
Chapter 1. Introduction to Parallel Sysplex	1
1.1 What is a sysplex?	2
1.1.1 Why use a sysplex?	3
1.2 Data sharing in a sysplex	5
1.3 Parallel processing in a sysplex	8
1.4 Managing work in a sysplex	14
Chapter 2. Application design	17
2.1 Applications in a sysplex	18
2.2 Enabling for continuous availability	19
2.2.1 Affinities	21
2.2.2 Locking	24
2.2.3 Batch window	25
2.2.4 Single points of failure.	25
2.2.5 Performance	26
2.3 Step-wise deployment/versioning	27
2.4 Other considerations	29
2.4.1 Granularity, modularity, and generally reused modules	29
2.4.2 Fault tolerance	30
2.4.3 Debugging and tracing	30
2.4.4 Testing environment	31
Chapter 3. DB2 application considerations	33
3.1 Introduction to DB2 data sharing.	34
3.1.1 Advantages of data sharing	34
3.1.2 How data sharing works	38
3.2 Application considerations for developers.	40
3.2.1 Locking	40
3.2.2 Lock avoidance	41
3.2.3 Commit frequency	45
3.2.4 Locking recommendations	46
3.2.5 DB2 database design	47
3.2.6 Batch considerations.	51
3.2.7 Managing commit frequency in batch	52
3.2.8 Distributed access to a data sharing group.	53
3.2.9 Programming guidelines	54
3.2.10 Additional DB2 considerations	55
3.3 Application considerations for DB2 system administrators	56
3.3.1 BIND options	57
3.3.2 DDL options	58
3.3.3 DB2 installation options	59

3.3.4	Deadlock and time-out detection	61
3.3.5	Backup and recovery	62
3.4	Example application	62
3.4.1	Batch processing	63
3.4.2	Parallelizing sequential processes	64
3.4.3	Extraordinary processes	65
Chapter 4.	DFSMStvs application considerations	67
4.1	Introduction to DFSMStvs	68
4.2	Batch application considerations.	69
4.2.1	Unit of recovery.	69
4.2.2	Sharing files in the batch job structure	70
4.3	Transactional recovery	71
4.3.1	Using DFSMStvs: unique input	73
4.3.2	Using DFSMStvs: shared input	75
4.4	Programming changes	77
Chapter 5.	CICS application considerations.	81
5.1	Introduction to CICS TS	82
5.2	How to exploit sysplex in CICS TS applications	82
5.2.1	Affinities through use of CICS commands	85
5.2.2	CICS Transaction Affinities Utility	87
5.2.3	Programing facilities and techniques and their implications on affinities	89
5.2.4	Affinity summary	92
5.2.5	What happens if you have to create an affinity?	92
5.3	CICS TS application checklist.	92
Chapter 6.	IMS application considerations.	95
6.1	IMS Transaction Manager overview	96
6.1.1	IMS basics	96
6.1.2	IMS in Parallel Sysplex architecture	97
6.1.3	Related improvements	99
6.2	IMS Database Manager	103
6.2.1	Supported IMS databases	103
6.2.2	Data sharing integrity overview	106
6.2.3	Locking recommendations	107
6.2.4	Single points of failure.	112
6.2.5	Batch considerations.	113
6.2.6	Converting MSDBs	117
6.3	IMS Transaction Manager.	119
6.3.1	Affinities	119
6.3.2	Transaction workload balancing	121
6.3.3	Transaction simplification	122
6.3.4	General IMS TM application performance considerations	123
6.3.5	Connectivity	124
6.3.6	IMS configuration considerations	128
Chapter 7.	WebSphere MQ application considerations.	131
7.1	A brief introduction to WebSphere MQ	132
7.2	A standalone queue manager.	133
7.3	Queue managers in a queue-sharing group	134
7.4	Coupling Facility list structures	135
7.5	Channels in a queue-sharing group	137
7.5.1	Channel initiators	137

7.5.2 Shared inbound channels	138
7.5.3 Shared outbound channels	139
7.6 Intra-group queuing	141
7.7 Benefits of using queue sharing groups	143
7.8 Initial considerations related to the use of QSGs	144
7.8.1 Technical setup	144
7.8.2 Application development	146
7.9 How applications can exploit WebSphere MQ shared queues	147
7.9.1 Multiple cloned servers	148
7.9.2 Replicated local queues with shared inbound channels	149
7.9.3 IMS Bridge	150
7.9.4 Queue partitions	151
7.9.5 Serialized applications	151
7.9.6 Shared queues and clustering	152
7.9.7 Inbound channels and target queues	152
7.9.8 Shared queue triggering	154
7.10 Limitations and restrictions	158
7.11 Sharing and clustering	159
7.12 Some final considerations	161
Chapter 8. Implementation and migration	163
8.1 Migration planning	164
8.2 CICS	164
8.2.1 Create the CICS sysplex environment	164
8.2.2 Exploit the sysplex environment with the applications	166
8.3 DB2 data sharing implementation	166
8.3.1 Plan a naming convention	166
8.3.2 Planning for availability	167
8.3.3 Migrating DB2 applications	168
8.3.4 Monitoring DB2 data sharing	169
8.4 WebSphere MQ	170
8.4.1 Migrating your existing applications to use shared queues	170
8.4.2 Migration scenario	172
8.4.3 Recommendations	178
Glossary	179
Related publications	185
IBM Redbooks	185
Other publications	185
Online resources	186
How to get IBM Redbooks	186
Help from IBM	186
Index	187

Archived

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:

AnyNet®
AIX®
CICS/ESA®
CICS/VSE®
CICS®
CICSplex®
Domino®
DB2 Connect™
DB2®
@server®
Extended Services®
ESCON®
FICON®

Geographically Dispersed Parallel
Sysplex™
GDPS®
IBM®
IMS™
Language Environment®
MQSeries®
MVS™
MVS/ESA™
OS/390®
Parallel Sysplex®
RAA®
RACF®

Redbooks™
Redbooks (logo) ™
RMF™
S/390®
SupportPac™
Sysplex Timer®
System/390®
VisualAge®
VTAM®
WebSphere®
z/OS®
zSeries®

The following terms are trademarks of other companies:

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.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

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

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

Preface

This IBM® Redbook introduces a top-down architectural mindset that extends considerations of IBM z/OS® Parallel Sysplex® to the application level, and provides a broad understanding of the application development and migration considerations for IMS™, DB2®, Transactional VSAM, CICS®, and MQ applications.

Parallel computing and data sharing technologies are playing a major role in e-business computing. Despite their compelling capabilities, however, these technologies are virtually unknown within the application architect and developer communities and are deployed more like operational enhancement technologies, with limited application involvement. Parallel Sysplex deployment models often expressly attempt to segregate the application environment from the underlying environment.

But today's software engineering principles recommend that you view applications to be designed as the drivers for availability and data sharing requirements for any technology. This publication explains how to achieve this objective by designing applications to exploit the capabilities of Parallel Sysplex.

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, Poughkeepsie Center.

Jordi Alastrué i Soler is an IT Architect at la Caixa in Barcelona, Spain. He holds a degree in Computer Science from the Universitat Politècnica de Catalunya, and has 16 years of experience in application development tools and IMS and DB2 database administration. He is currently a member of an application architecture support team.

Mark Anders is a Consulting IT Specialist with IBM Advanced Technical Support, Americas, located in Dallas, Texas. He has more than 30 years of IBM experience supporting DB2 and IMS as a Systems Engineer, Database Administrator, and IT Specialist. For the past nine years, he has provided DB2 for OS/390® and z/OS technical support and consulting services. His areas of expertise include DB2 data sharing, performance, and availability.

Karl Bender is an Advisory Specialist at the z/OS EMEA Technical Support Center, Mainz, Germany. He supports Language Environment®, COBOL, PL/1, C/C++, and Java™, and teaches IBM classes worldwide on Language Environment. He has 25 years of experience in application development support for mainframe MVS™ OS/390 and z/OS.

Amardeep Bhattal is a Software Engineer at IBM Hursley Park, England, with four years of experience in IBM systems programming, working on WebSphere® MQ for z/OS, including the design and development of the shared channel architecture. He is currently a member of the WebSphere MQ for z/OS Development Team.

Paolo Bruni is a DB2 Information Management Project Leader at the International Technical Support Organization, San Jose Center. During his many years with IBM in development and in the field, Paolo's work has mainly been related to database systems.

Andy Clifton is a CICS System Tester at IBM Hursley, England, with 15 years experience with CICS. He holds a degree in Mathematics and Computing from Bath University, England. He is currently working with the CICS Regression Test Team at Hursley and as a CICS

expert on the Consolidated Service Test Team, helping to develop new applications for these environments. He previously worked on the Parallel Sysplex Trainer, including the CICS exercise workloads.

John Iczkovits is a Consulting IT Specialist with IBM Advanced Technical Support, Americas, located in Dallas, Texas. He provides DB2 for z/OS and OS/390 technical support and consulting services. His areas of expertise include DB2 data sharing, performance, and availability. He has more than six years of IBM experience supporting DB2 as a Systems Engineer, Database Administrator, IT Specialist, Consultant, and project lead. He has an operations background and more than 20 years of IT experience, ranging from database products such as DB2 and IMS to MVS systems support, including storage management.

Franck Inje is a Project Leader at the International Technical Support Organization, Poughkeepsie. He has 25 years of experience working on S/390® hardware and system performance. Before joining the ITSO, Franck was a Consulting IT Architect in France.

Pirooz Joodi is a Senior IT Architect for the National Architecture COE in US Business Innovation Services at IBM. He is IBM Certified with 20 years of experience in defining enterprise architectures and migration strategies, and has completed a Doctoral program at Columbia University, New York. He has published technical material on data warehousing, B2B, and application integration for IEEE, US DOD, and IBM. His current assignment includes building enterprise architecture for multi-industry solutions.

Vasilis Karras is a Project Leader at the International Technical Support Organization, Poughkeepsie. His areas of expertise include the zOS systems support area, database products, and vendor software including Oracle and PeopleSoft.

Andy Mitchelmore is a Consulting IT Architect for IBM UK. He is an IBM Certified Architect with 30 years of experience in operations, application development, infrastructure design, and IT architecture definition. His latest assignment involved implementing a sysplex with data sharing for a large CICS and DB2 customer service application at a major UK utility.

Robert Queen is an IT Specialist in the UK. He has more than 15 years of experience in IBM systems programming. He currently supports an IBM Global Services customer that has implemented Parallel Sysplex and data sharing.

Mark Rader is a DB2 Specialist at ATS, Dallas. He has 20 years of technical experience with IBM, including large systems, communications, and database. He has worked primarily with DB2 for MVS, OS/390, and z/OS for more than 15 years, and has specialized in data sharing, performance, and related topics for the past eight years.

Mayur Raja is a Software Engineer at IBM Hursley Park, England, with 15 years of experience in IBM systems programming. He has specialized in WebSphere MQ for z/OS for the past nine years, during which time he has worked with MQSeries® customers in the US and Asia Pacific, and more recently on the development of IGQ and shared channels. He is currently a member of the WebSphere MQ for z/OS Development Team.

Pete Siddall is a Software Engineer at IBM Hursley Park, England, with 17 years of experience in IBM systems programming. For the past six years he has specialized in WebSphere MQ for z/OS as the lead developer for the implementation of shared queues. He is currently a member of the WebSphere MQ for z/OS Development Team.

Jens Erik Wendelboe is an IT Specialist at Danske Bank in Denmark. He has 30 years of experience in different aspects of application development, development tools, and systems software. For the past five years, his responsibilities have included the setup and usage of WebSphere MQ at Danske Bank.

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

Adem Arslan
Garanti Technology

Richmond Roberts
Bank of Montreal

John Tilger
The Vanguard Corporation

Roy Kinney
Verizon

Yucel Senturk
Pamuk Bank

John McNeilly
East Cliff Consulting Ltd, UK

Mike Bredenkamp
IBM Dallas Support Center

John Campbell
IBM Silicon Valley Lab

Maida Snapper
IBM Silicon Valley Lab

Thanks also to:

Paul Kettley, lead architect for shared queues and WebSphere MQ for z/OS; Steve Hobson, architect for shared queues and WebSphere MQ for z/OS; Neil Johnston, developer for WebSphere MQ for z/OS; and Richard Harran, developer for WebSphere MQ for z/OS for their assistance with writing the WebSphere MQ sections.

Anthony O'Dowd, lead architect for shared channels and now architect for WebSphere MQSI for z/OS, for supplying recommendations for migration, and for allowing us to use some of his presentation material.

Steve Hobson and Morag Hughson, lead developers for distributed queuing on WebSphere MQ for z/OS, for allowing us to use some of their presentation material.

Dave Fisher, lead tester in the WebSphere MQ for z/OS Performance Test Team, for shared queue performance data.

Thanks also to all members of the WebSphere MQ for z/OS Development, Test, and Service Teams whose vast contributions over the years have undoubtedly helped to make WebSphere MQ for z/OS a real success.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners, and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us because we want our 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 e-mail to:

redbook@us.ibm.com

- Mail your comments to:

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

Introduction to Parallel Sysplex

The word *sysplex* might not be new to you. The MVS/ESA™ Sysplex has been available since 1990, when it was announced as a platform for an evolving large system computing environment.

Since then, Parallel Sysplex has become the large system computing environment that offers improved price and performance through cost-effective processor technology and enhanced software. This technology builds on existing data processing skills and runs existing applications—an additional cost saver. A sysplex can also increase your system's availability while increasing your potential to handle more work.

In this chapter, we begin by defining what constitutes a sysplex, and then we go on to describe its capabilities and explain why it might be suitable for your environment.

We cover the following topics:

- ▶ “What is a sysplex?” on page 2
- ▶ “Data sharing in a sysplex” on page 5
- ▶ “Parallel processing in a sysplex” on page 8
- ▶ “Managing work in a sysplex” on page 14

1.1 What is a sysplex?

A *sysplex* is a collection of z/OS systems that cooperate, using certain hardware and software products, to process work. A conventional large computer system also uses hardware and software products that cooperate to process work. A major difference between a sysplex and a conventional large computer system is the improved growth potential and level of availability in a sysplex.

The sysplex increases the number of processing units and z/OS operating systems that can cooperate, which in turn increases the amount of work that can be processed. To facilitate this cooperation, new products were created and existing products were enhanced.

Sysplex hardware

The following hardware components participate in a sysplex:

Coupling Facility

The Coupling Facility enables high-performance multisystem data sharing. You can use standalone Coupling Facility models, or define logical partitions in zSeries® servers to be used as Coupling Facilities.

Coupling Facility Channel

These channels provide high-speed connectivity between the Coupling Facility and the servers that use it.

Coupling Facility Control Code

The Coupling Facility equivalent of an operating system is the Coupling Facility Control Code (CFCC), which is implemented only as licensed code in the CF and runs only in LPAR mode. No software runs in the CF processor. Different releases of the CFCC are referred to as *levels*. For example, the latest CFCC level (as of this writing) is CFCC Level 13.

Sysplex Timer®

The Sysplex Timer synchronizes the time-of-day (TOD) clocks in multiple zSeries servers and Coupling Facilities in a Sysplex.

Processors

Selected models of S/390 or zSeries servers can take advantage of sysplex.

Channels and Directors

ESCON® and FICON® channels enhance data access and communication in a sysplex. The ESCON and FICON Directors add dynamic switching capability for channels.

Control units and I/O devices

ESCON and FICON control units and I/O devices in a sysplex provide the increased connectivity necessary among a greater number of systems.

Sysplex software

The following software works with the hardware to enable interaction and cooperation among systems in a sysplex:

► System software

Base system software that is enhanced to support a sysplex includes the z/OS operating system, JES2 and JES3, and DFSMS).

► Networking software

VTAM® and TCP/IP support the attachment of a sysplex to a network.

- ▶ Data management software

The IBM data managers that support data sharing in a sysplex are Information Management System Database Manager (IMS DB), DATABASE 2 (DB2), and Virtual Storage Access Method (VSAM).

- ▶ Transaction management software

CICS Transaction Server, Information Management System Transaction Manager (IMS TM), WebSphere Application Server support transaction management in a sysplex.

- ▶ Systems management software

A number of software products are enhanced to run in a sysplex and exploit its capabilities. The products manage accounting, workload, operations, performance, security, and configuration, and they make a sysplex easier to manage by providing a single point of control.

1.1.1 Why use a sysplex?

Now that you have been introduced to the pieces that make up a sysplex, you might be wondering what a sysplex could do for you. If your data center is responsible for even *one* of the following types of work, then you could benefit from using a sysplex:

- ▶ Large business problems: ones that involve hundreds or thousands of end users, or deal with a large volume of work that can be counted in the millions of transactions per day
- ▶ Work that consists of small work units (such as online transactions), or large work units that can be subdivided into smaller work units (such as queries)
- ▶ Concurrent applications on different systems that must directly access and update a single database without jeopardizing data integrity and security

A sysplex shares the processing of work across multiple z/OS systems, and as a result offers the following benefits.

- ▶ Reduced cost through:
 - Cost-effective processor technology
 - Continued use of large-system data processing skills without re-education
 - Protection of application investments
 - The ability to manage a large number of systems more easily than other, comparably performing, multisystem environments
- ▶ Continuous availability so that applications can be available 24 hours a day, 7 days a week, 365 days a year (or close to it)
- ▶ Ability to do more work:
 - Greater capacity
 - Improved ability to manage response time
 - Platform for further capacity and response time advances
- ▶ Greater flexibility:
 - Ability to mix levels of hardware and software
 - Ability to dynamically add systems
 - An easy path for incremental growth
- ▶ Varied platforms for applications, including parallel, open, and client/server.

Depending on your data center's goals and needs, some of these benefits might be more attractive to you than others. In this IBM Redbook, we focus on improved continuous availability and greater flexibility.

Parallel Sysplex

In 1990, IBM announced the sysplex as the strategic direction for large-systems computing environments and described it as a collection of MVS systems that cooperate, using a combination of hardware and software components to process work. At the time of this announcement there was no Parallel Sysplex; only a base sysplex existed that provided improvements in inter-processor communications between systems and any subsystems wishing to exploit those services, but no data sharing services.

The Parallel Sysplex was introduced later in the 1990s, and it added hardware and software components to provide for sysplex data sharing. In this context, data sharing means the ability for sysplex member systems and subsystems to store data into, and retrieve data from, a common area. The Parallel Sysplex supports a greater number of systems and significantly improves communication and data sharing among those systems.

With the Parallel Sysplex, data sharing through a new coupling technology, the Coupling Facility, gives high-performance, multisystem data sharing capability to authorized applications, such as z/OS subsystems. Use of the Coupling Facility by database management and transaction subsystems such as IMS, CICS, and DB2 ensures the integrity and consistency of data throughout the entire sysplex.

The capability of linking together many systems and providing multisystem data sharing makes the sysplex platform ideal for parallel processing, particularly for online transaction processing and decision support.

In short, a Parallel Sysplex builds on the base sysplex capability, and allows you to increase the number of servers and z/OS images that can directly share work. The Coupling Facility enables high-performance, multisystem data sharing across all the systems. In addition, workloads can be dynamically balanced across systems with the help of new workload management functions.

Figure 1-1 on page 5 illustrates the hardware components of a Parallel Sysplex.

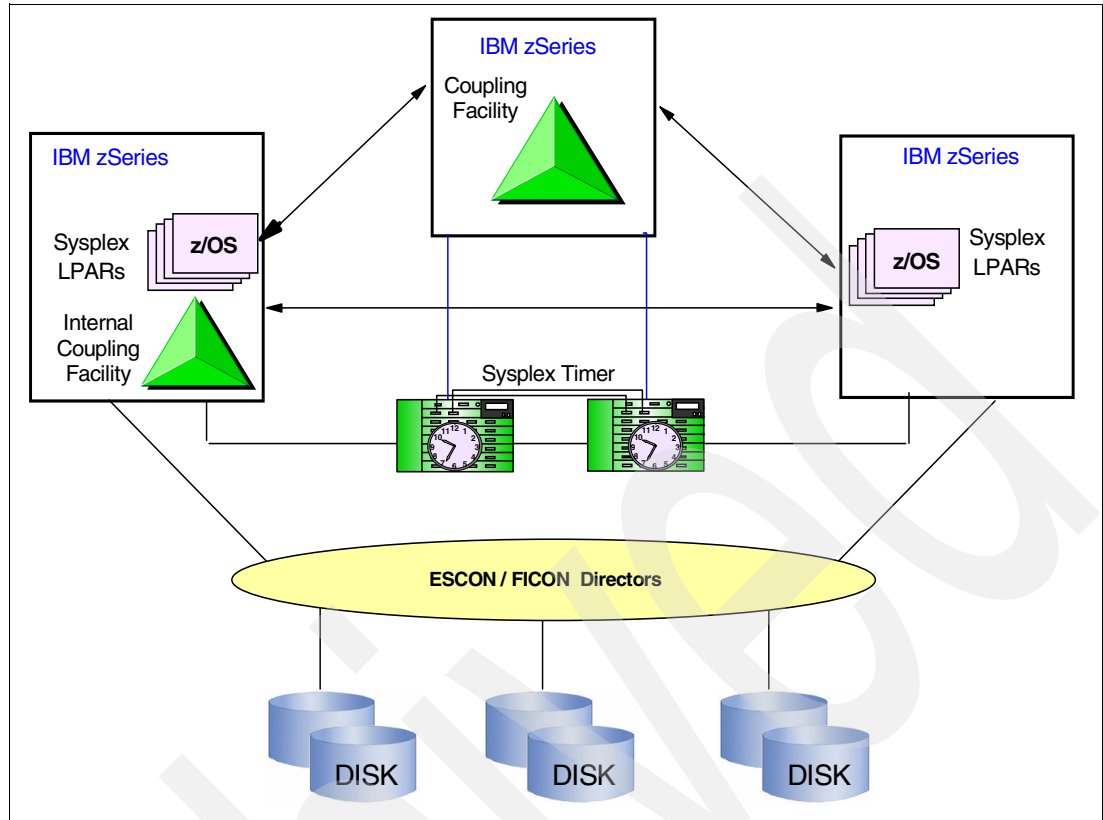


Figure 1-1 Parallel Sysplex hardware components

The Parallel Sysplex environment can scale up to 32 z/OS system images. There can be a mix of any servers that support the Parallel Sysplex environment.

Summary

The unique characteristics of a Parallel Sysplex can enable you to reduce your total cost of computing over prior offerings of comparable function and performance. Sysplex design characteristics mean that you can run your business continuously, even when it is growing or changing. You can dynamically add and change systems in a sysplex and configure them for no single points of failure. If your revenue depends on continuously available systems, a sysplex can protect your revenue. If you need to grow beyond the limits imposed by today's technology, a sysplex lets you go beyond those limits, and helps you avoid the complex and expensive splitting and rejoining of data centers.

The innate robustness and reliability of the z/OS operating system and zSeries servers are the foundation of a sysplex. That robustness and reliability are extended to all systems in a sysplex through cross-system workload balancing and data sharing using the coupling technologies. Therefore, applications on multiple systems can be available continuously to end users, yet the applications are shielded behind a single-system view.

1.2 Data sharing in a sysplex

Connecting a large number of systems brings with it special considerations, such as how the large number of systems communicate and how they cooperate to share resources. These considerations affect the overall operation of z/OS systems.

A sysplex significantly changes the way z/OS systems can share data. As the number of systems increase, it is essential to have an efficient way to share data across systems. The Coupling Facility enables centrally accessible, high-performance data sharing for authorized applications, such as subsystems and z/OS components, that are running in a sysplex. These subsystems and components then transparently extend the benefits of data sharing to their applications.

Use of the Coupling Facility substantially improves the feasibility of connecting many z/OS systems together in a sysplex to process work in parallel.

Data sharing

The concept of *data sharing* is not new. Different applications within a system often need to access the same information, sometimes to read it and other times to update it. Sometimes several copies of the data exist, and with that comes the requirement of keeping all of the copies identical. If the system fails, you need a way to preserve the data with the most recent changes.

Data validity is controlled by a data management system. Within a single z/OS system, the data management system keeps track of which piece of data is being accessed or changed by which application in the system. It is the data management system's responsibility to capture and preserve the most recent changes to the data, in case of system failure.

When two or more z/OS systems share data, each system has its own copy of a data management system. Communication between the data management systems is essential.

Therefore, multisystem data sharing hinges on high-performance communication to ensure data validity among multiple data management systems, and it requires high-speed data accessing methods.

Data validity

When many applications share data, within the same z/OS system or among several z/OS systems, they need mechanisms to guarantee data validity. Two such mechanisms are serialization and data consistency (buffer invalidation).

Serialization is a mechanism that allows control over the access and update of data. Through serialization, only one application can access and change a given piece of data at a time. As the number of applications increases, contention over a piece of data becomes more likely, and performance slows. One way to enable more applications on multiple systems to access more data at the same time is to allow serialization on smaller pieces of data. Generally, serialization at a lower level of granularity increases overhead, which slows performance. But, by using the Coupling Facility, data management systems can afford to serialize at a low level with acceptable performance.

Another requirement for maintaining data validity is a mechanism that ensures consistency among local copies of data. One such mechanism is buffer invalidation. This mechanism indicates to each of the data sharing systems whether their local data reflects the most recent changes.

Database managers need high-speed serialization and data consistency mechanisms across systems. Not until data sharing based on the Coupling Facility technology was developed has it been possible to efficiently share data using these mechanisms across more than two z/OS systems.

Sharing data using Coupling Facility technology

In contrast to other design options that use no sharing, data sharing based on the Coupling Facility makes it practical to have read/write data sharing among more than two z/OS systems. The Coupling Facility enables data management systems to communicate so that they can share data directly. There is no single system creating a bottleneck, the data does not have to be partitioned or re-partitioned when you add another system, and there is no longer a two-system limitation.

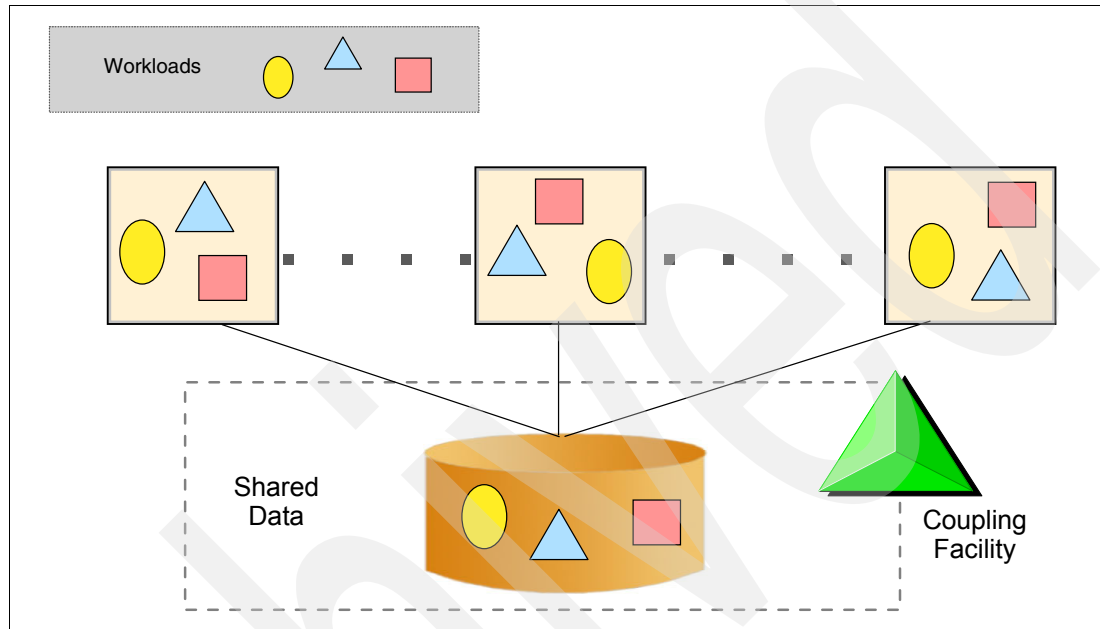


Figure 1-2 Sysplex data sharing using the Coupling Facility

A closer look at Coupling Facility technology

The technology that makes high-performance sysplex data sharing possible is a combination of hardware and software services available in the supporting z/OS releases.

A Coupling Facility can be a zSeries Coupling Facility standalone model or a logical partition of an IBM zSeries server. High-bandwidth fiber optic links known as Coupling Facility channels provide high-speed connectivity between the Coupling Facility and systems directly connected to it.

Within the Coupling Facility, storage is dynamically partitioned into structures. z/OS services manipulate data within the structures. Each of the following structures has a unique function:

- ▶ **Cache structure:** Supplies a mechanism called *buffer invalidation* to ensure consistency of cached data. The cache structure can also be used as a high-speed buffer for storing shared data with common read/write access.
- ▶ **List structure:** Enables authorized applications to share data that is organized in a set of lists, for implementing functions such as shared work queues and shared status information.
- ▶ **Lock structure:** Supplies shared and exclusive locking capability for serialization of shared resources down to a very small unit of data.

Exploiters of the Coupling Facility

Authorized applications, such as subsystems and z/OS components in the sysplex, use the Coupling Facility services to cache data, share queues and status, and access sysplex lock

structures in order to implement high-performance data sharing and rapid recovery from failures. The subsystems and components transparently provide the data sharing and recovery benefits to their applications.

Some IBM data management systems that use the Coupling Facility include database managers and a data access method.

Information Management System Database Manager (IMS DB)

IMS DB is the strategic hierarchical database manager from IBM. It is used for numerous applications that depend on its high performance, availability, and reliability. A hierarchical database has data organized in the form of a hierarchy (pyramid). Data at each level of the hierarchy is related to, and in some way dependent on, data at the higher level of the hierarchy.

IMS database managers on different z/OS systems can access data at the same time. By using the Coupling Facility in a sysplex, IMS DB can efficiently provide data sharing for multiple z/OS systems and thereby extends the benefits of IMS DB data sharing. IMS DB uses the Coupling Facility to centrally keep track of when shared data is changed.

DATABASE 2 (DB2)

DB2 is the IBM strategic relational database manager. A relational database has the data organized in tables with rows and columns. DB2 data sharing support enables multiple DB2 subsystems within a sysplex to concurrently access and update shared databases. DB2 data sharing uses the Coupling Facility to efficiently lock, to ensure consistency, and to buffer shared data. Similar to IMS, DB2 serializes data access across the sysplex through locking. DB2 uses Coupling Facility cache structures to manage the consistency of the shared data. DB2 cache structures are also used to buffer shared data within a sysplex for improved sysplex efficiency.

Virtual Storage Access Method (VSAM)

VSAM, a component of DFSMS, is an access method that gives CICS and other application programs access to data stored in VSAM-managed data sets. DFSMS supports a VSAM data set accessing mode called *Record Level Sharing* (RLS). RLS uses the Coupling Facility to provide sysplex-wide data sharing for CICS and the other applications that use the new accessing mode. By controlling access to data at the record level, VSAM enables CICS application programs running in different CICS address space, and in different z/OS images, to share VSAM data with complete integrity. The Coupling Facility provides the high-performance data sharing capability needed to handle requests from multiple CICS regions.

In addition to data management systems, there are other exploiters of the Coupling Facility, such as the z/OS Security Server (RACF®), JES2, WebSphere MQ, and so on. Transaction management systems also exploit the Coupling Facility to enhance parallelism.

1.3 Parallel processing in a sysplex

Parallel processing is a technology that increases application availability. It gives you more servers for enhanced throughput; and if the enhanced throughput relieves system overload, parallel processing will visibly reduce end-user response time.

One reason for setting up a sysplex is to enable parallel processing. Special-purpose parallel systems for engineering and scientific applications have been available for years. The sysplex, however, can provide parallel processing for commercial online transactions.

The sysplex brings together parallel capability on many levels: in the applications, in the supporting sysplex software, and in the hardware. Applications can be replicated to run in parallel on a single tightly coupled multiprocessing system. When you run parallel applications in a sysplex with subsystems that support enhanced communication across systems, you have a greater degree of parallelism. And when you add hardware to that greater degree of parallelism, hardware with the capability for modular, incremental growth, you have the broad-based parallelism that is available in a sysplex.

What is parallel processing?

Parallel processing in a sysplex is the ability to simultaneously process a particular workload on multiple servers, each of which may have multiple CPs. A CP can generally process one unit of work at a time. Typically the more CPs in a system (as in a tightly coupled multiprocessor), the more work can be processed simultaneously. When you have multiple servers processing the same workload, you can significantly reduce elapsed processing time.

Parallel processing is a type of multiprocessing. For the purposes of this publication, the distinction between multiprocessing and parallel processing is in whether the same kind of work is being processed. While multiprocessing is the simultaneous processing of work, it is not necessarily the same kind of work; parallel processing in a sysplex is the simultaneous processing of one type of work, whether it is many small but distinct units of work, or a larger unit of work that is broken down to run in parallel.

In a multiprocessing environment, at a particular moment in time, one CP might be running a payroll batch application, while another CP processes online transactions. The different work runs at the same time. If the payroll batch application takes hours to complete, and if the application is structured as a single task, it can only use the power of one CP. The fact that there is more than one CP available does not split up the long-running batch application. But if the batch application is structured as multiple tasks, it can be processed simultaneously on many CPs—in other words, in parallel.

Similarly, if there are more online transactions than one CP can handle, adding more CPs enables the online transactions to run in parallel.

Therefore, parallel processing is the ability to use many CPs to simultaneously process a particular workload; and by running a workload in parallel, it is possible to make a noticeable improvement in the response time for the end user.

In a sysplex, it is possible to have one workload running on many tightly coupled multiprocessors that are sharing data using the Coupling Facility. How many CPs you have depends on the configuration. For example, a configuration of 16 zSeries servers with 6 CPs per server gives you a total of 96 CPs that can process a workload in parallel.

Problems solved by parallel processing

Parallel processing in a sysplex can be used for two different programming problem situations:

- ▶ When the number of transactions noticeably slows response time
- ▶ When a long-running application takes too long to complete

Both situations can adversely affect end-user response time.

Increased number of transactions

When the number of incoming transactions exceeds the processing capability, transactions are delayed and response time increases. A large number of transactions can build up when they exceed the processing capability.

By adding servers and running transactions in parallel, you can reduce end-user response time by increasing throughput. Figure 1-3 shows how adding processing capability increases throughput.

Similarly in your environment, when an increased number of transactions slows throughput, you can use a sysplex and the Coupling Facility to increase the number of CPs running in parallel.

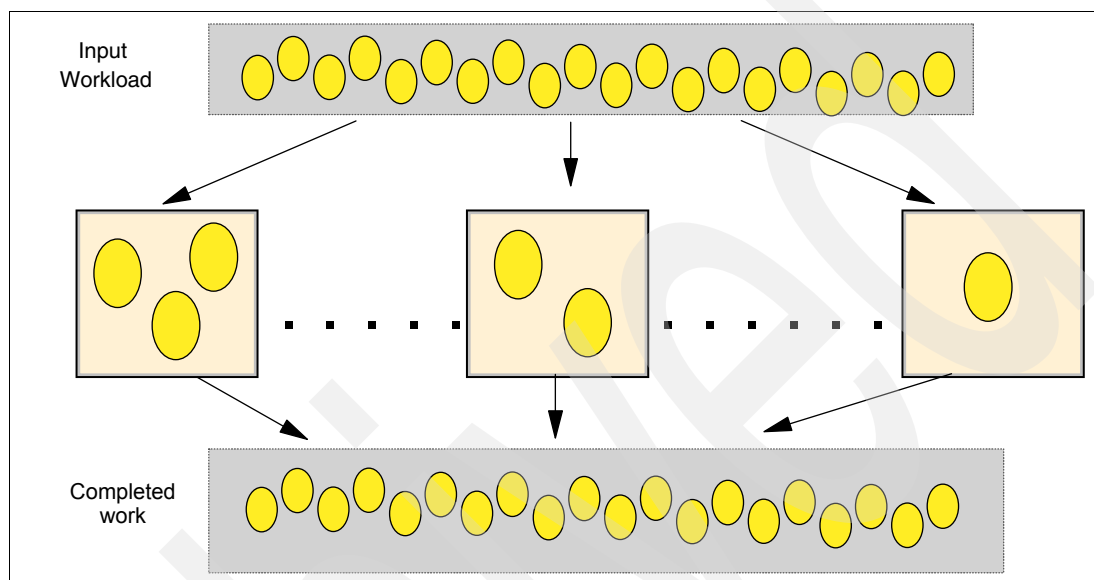


Figure 1-3 Increasing throughput for a workload

Lengthy application

To shorten the processing time of a single, long-running application, subdivide it into smaller units of work, if possible. These smaller units can then run in parallel. Figure 1-4 on page 11 shows how a lengthy application is processed one piece at a time when there is only one CP.

When the lengthy application is divided into smaller units of work and there are many parallel servers, the smaller units of work can run simultaneously and in parallel. Figure 1-4 on page 11 shows how a lengthy application can be processed in parallel by dividing it among many parallel servers.

You can use this same approach in a sysplex to speed up the processing of a long-running application, such as a complex query.

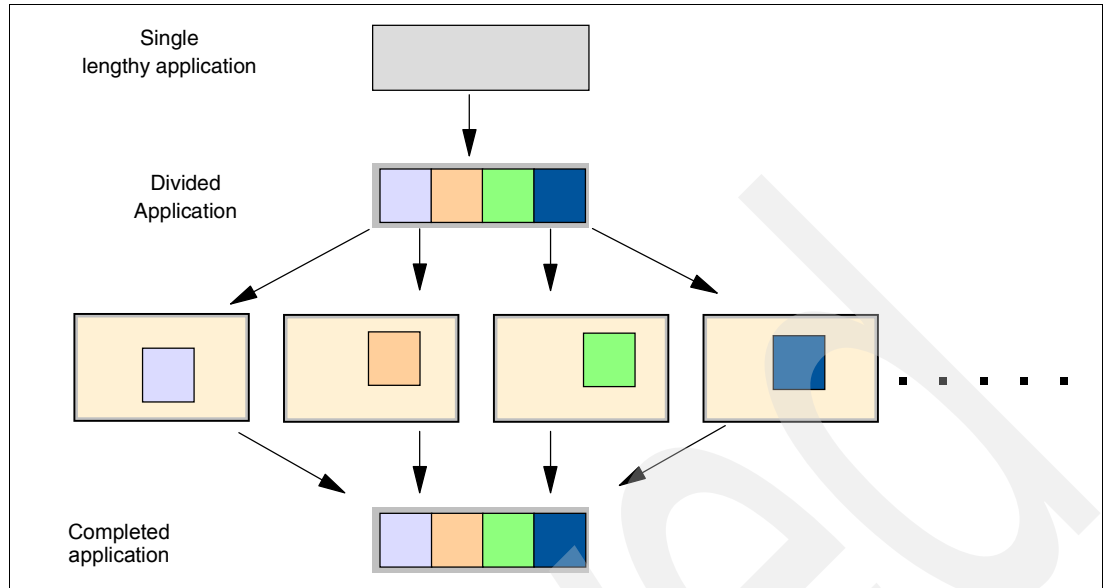


Figure 1-4 Decreasing the processing time of a lengthy application

Running work in parallel

z/OS is known for its strength and dependability in processing applications that solve large business problems. These are some characteristics of a business problem, each of which makes it a large problem:

- ▶ Volume of work, up to millions of transactions per day
- ▶ Number of end users, up to thousands
- ▶ Amount of storage required, such as terabytes of data
- ▶ Bandwidth requirements for sending data in a network, such as megabytes of data in seconds
- ▶ Stringent requirements for continuous availability (24 hours a day, 7 days a week, 365 days a year)

Some common z/OS applications that deal with large business problems and can be made to run in parallel are batch, query, and online transaction processing (OLTP).

A batch workload consists of multiple job streams, which can be unrelated and can therefore run in parallel. A single job stream, however, runs in sequence and usually cannot take advantage of parallelism. When a single job stream is lengthy, it can take considerable time to complete.

Queries are a diverse type of workload and usually rely on partitioned data when running in parallel. To help you process queries in parallel, IBM offers a parallel query server. The *parallel query server* is a self-contained system made up of hardware, software, and database services. The system is fully installed, customized, and serviced by IBM. The parallel query server is used to process large queries in a shared relational database. For more information about the parallel query server, see your IBM marketing representative.

Online transaction applications are a common z/OS application type that can take advantage of parallelism. They are the focus of the following topics, because there is extensive software support for OLTP in a sysplex, and the number of online transaction applications are on the increase.

Online transaction processing

Online transaction applications are used by banks, airlines, insurance companies, and other businesses that give online users direct access to information. The applications process units of work, called *transactions*. A single transaction might request a bank balance; another might update that balance to reflect a deposit.

An application can be replicated to run in parallel on a tightly coupled multiprocessor so that it can simultaneously process multiple transactions. The degree of parallelism depends on the system software, the transaction manager, and the hardware that the application runs on.

Online transaction applications use system software services provided by transaction managers, and they access data controlled by data management systems. To balance the transactions dynamically across systems, z/OS Workload Manager, a component of z/OS, works with the transaction managers. In this section, we first discuss transaction managers, and then describe workload management.

Transaction managers

From the time a transaction manager obtains the transaction from the network until it returns the completed transaction to the network, it is responsible for:

- ▶ Sending the transaction to the appropriate application
- ▶ Converting transaction data to a form that can be used
- ▶ Reconverting results to a form that can be displayed on an output device

If the transaction manager can simultaneously process copies of an application in multiple address spaces within a single system, an online transaction application can run in parallel.

If the transaction manager can take advantage of sysplex support and simultaneously process copies of an application across different z/OS systems, the OLTP application can run with a greater degree of parallelism. (See Figure 1-5.)

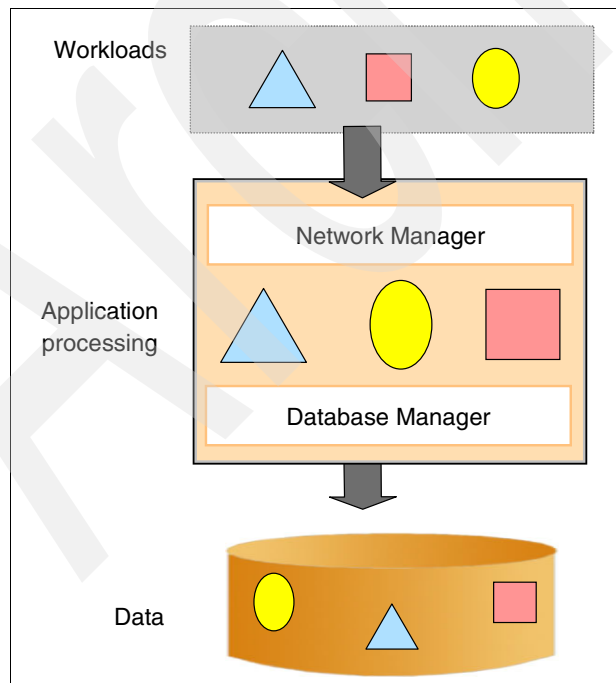


Figure 1-5 Parallel OLTP applications in a single system

Customer Information Control System (CICS) and Information Management System Transaction Manager (IMS TM) provide an additional degree of parallel support in a sysplex. Let's look at these in more detail.

Customer Information Control System

CICS enables online transactions entered at terminals to be processed concurrently by user-written application programs. Originally CICS ran in a single z/OS address space that was responsible for terminals, applications, and files.

Using CICS multi-region operation, you can separate function into individual regions, namely terminal-owning regions (TORs), application-owning regions (AORs), and file-owning regions (FORs). These individual resource-owning regions can then be linked together and managed in what is called a *CICSplex*.

One of the benefits of separating the CICS functions into a number of resource-owning regions is to take advantage of multiprocessor systems, as each region (address space) can run on a different CP. You can structure a CICS environment such that the application-owning regions can process transactions in parallel.

Replicating regions and running multiple regions on multiple systems in a sysplex further increases the parallelism. The greater the number of regions, the greater the CICS availability in the event of a system failure.

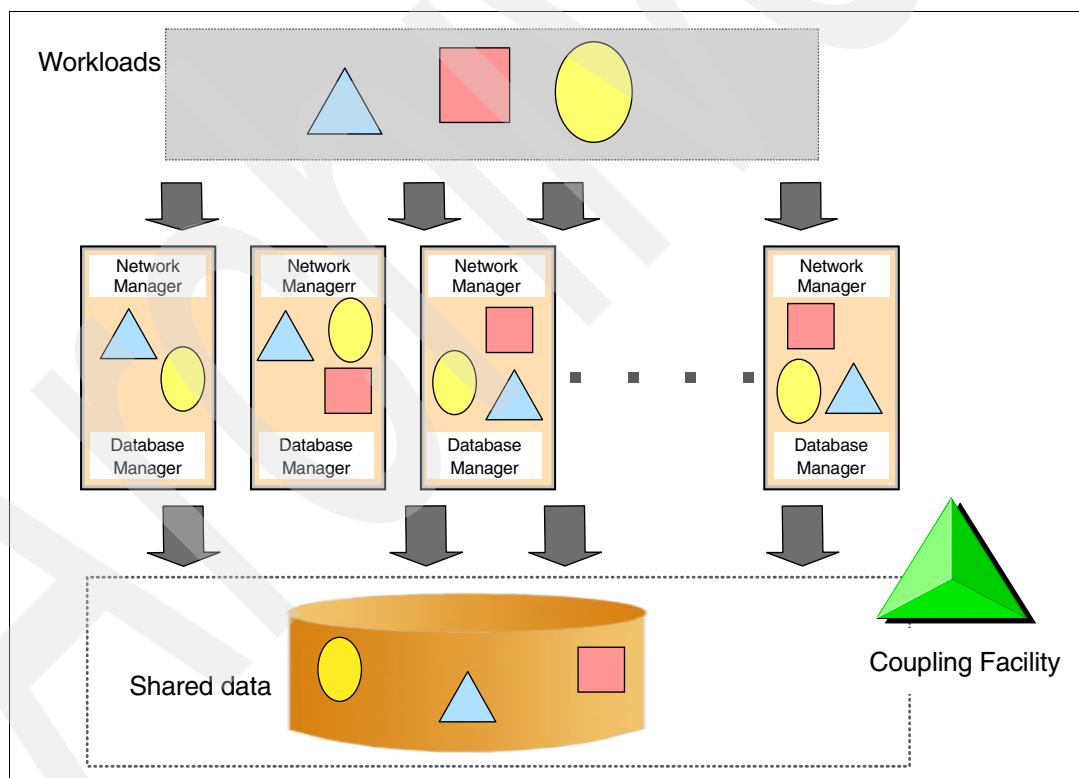


Figure 1-6 Online transaction applications in a Parallel Sysplex

IMS TM also takes advantage of the sysplex. Several IMS TMs can coordinate work in a sysplex. Transaction message traffic workloads can be balanced. IMS continues to enhance and unify management of IMS systems in a sysplex.

Data management systems

When an online transaction application needs to access data, the data management system controls the data access and update. Data management systems are responsible for validity of the data, providing a way to recover data in case of failure, and a way to secure data from unauthorized access.

If the data management system takes advantage of the Coupling Facility, the online transaction applications can avoid shipping data back and forth between systems. By using the Coupling Facility for data sharing and for queuing and status control, data management systems and transaction managers can perform their duties more quickly and therefore manage more systems efficiently. This increase in the number of systems that can be managed increases the degree of parallelism even further.

IBM data management systems, such as IMS DB, DB2, and VSAM, take advantage of the Coupling Facility.

1.4 Managing work in a sysplex

Along with parallelism comes the need for simple and dynamic multisystem workload balancing. When you are in an environment with multiple systems, the set of performance issues changes. Existing mechanisms for managing system performance are complex and single-system-oriented.

To reduce the complexity of managing a sysplex, z/OS workload management provides dynamic sysplex-wide management of system resources. z/OS workload management is the combined cooperation of various subsystems (such as CICS, IMS, and VTAM) with the z/OS Workload Manager (WLM) component. An installation defines performance goals and a business importance to workloads through WLM. Workload management focuses on attaining these goals through dynamic resource distribution.

This type of workload management is different from the way workloads were previously managed. The new emphasis is on defining performance goals for work, and having z/OS and the subsystems adapt to meet the goals.

Defining performance goals

An installation defines performance goals in a *service policy*. Service policies are defined through an ISPF application, and they set goals for all types of z/OS-managed work. An installation can create multiple service policies to adjust performance goals for different periods of time.

The scope of a service policy is the sysplex. Each service policy has a name, and can be activated by the ISPF application, an operator command, or an automation package. Only *one* policy can be active at a time. When it is activated, all systems in the sysplex process toward the goals defined in the policy.

Workload management coordinates and shares performance information across the sysplex. Each z/OS system handles its own system resource management and dynamically matches resources to work according to the goals defined in the service policy. During processing, the system monitors how well the goals are being met and adapts accordingly as the environment changes. If there is contention for resources, each system makes the appropriate trade-offs based on the importance of the work and how well the goals are being met. This way, all systems can cooperate to protect work that is critical to your installation.

Reporting

Resource Measurement Facility (RMF™) combines z/OS system management facilities (SMF) data for the sysplex, and reports how well the sysplex is doing to achieve the goals defined in the service policy. In addition, execution delay information is available in SMF records that show where delays are occurring. If there is a problem, you can use this information to help adjust the performance goals, focus on specific subsystems having a problem, or make work scheduling adjustments.

Archived



Application design

In this chapter we provide general considerations related to application design in Parallel Sysplex environments.

The chapter contains the following:

- ▶ “Applications in a sysplex” on page 18
- ▶ “Enabling for continuous availability” on page 19
- ▶ “Step-wise deployment/versioning” on page 27
- ▶ “Other considerations” on page 29

2.1 Applications in a sysplex

Most S/390 applications can run “unmodified” on a Parallel Sysplex cluster. This is true in the sense that not all applications have to take advantage of the cluster capabilities. This is illustrated in Figure 2-1, which shows a three-system-images Parallel Sysplex:

- ▶ Appl 3, Appl 5, and Appl 6 each run on only one image. Each application is “supported” in the Parallel Sysplex environment but does not exploit its capabilities.

In this case, although management of all applications may be improved from the operations point of view, continuous availability and scalability characteristics of each individual application are unchanged compared to a single system image solution.

- ▶ Appl 1, Appl 2, Appl 4, and Appl 7 each run in multi-systems-capable mode with data sharing enabled and are exploiting, to various degrees, the Parallel Sysplex.

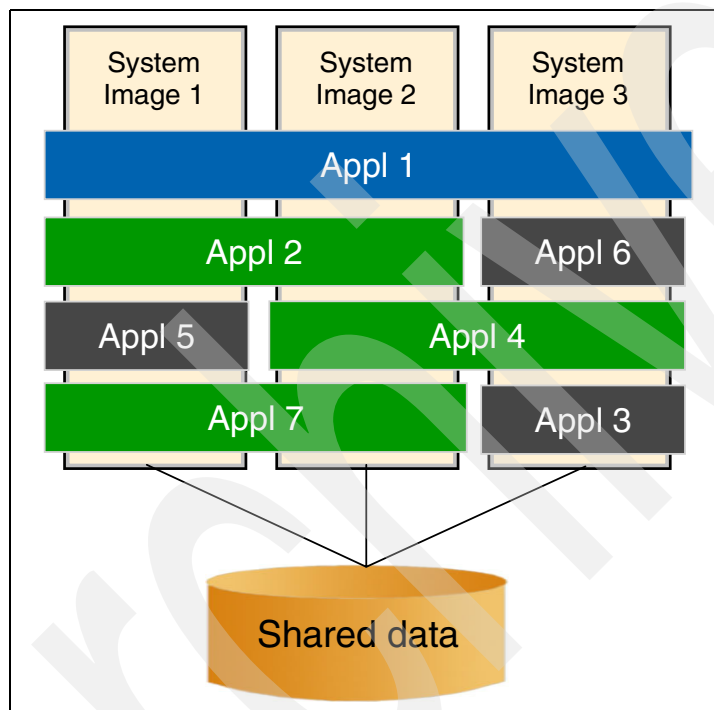


Figure 2-1 Applications in a Parallel Sysplex

From the continuous availability point of view, this provides immediate benefit:

- In case of failure, only a subset of users are affected: either one-third (Appl 1) or half (Appl 2, Appl 4, and Appl 7) of the active transactions in this example.

For connected users that do not have active transactions running, the failure is mostly transparent because the incoming application workload is immediately routed to the surviving system image or images.

- In case of scheduled zSeries server maintenance, z/OS system maintenance, or subsystem maintenance, only one system image is affected. Because there is a surviving application image, the application service is not interrupted.

In case of application maintenance, there is still a need to bring down the whole application. Although there is no easy way to resolve the application maintenance issue, you can now take advantage of multiple instances provided by the Parallel Sysplex design. Using the redundancy of application images, you may now think about a more granular application maintenance policy by application instance, and investigate the approaches described in 2.3, “Step-wise deployment/versioning” on page 27. This what you should be working toward.

Parallel Sysplex is a cluster-type solution that differs from Shared Memory Processor (SMP) design. As a consequence, you may have to rethink some application design choices. Most considerations deal with the following areas.

- ▶ Affinities

Either inter-transaction or system image affinity may have consequences for the scalability and backup capabilities of the cloned applications in a Parallel Sysplex:

- Logical (for example, image name)
- Physical (for example, hardware resource: cryptographic device, special printer)

- ▶ In Memory Counter

This was formerly applicable application-wide, but it is no longer applicable sysplex-wide. The server memory affinity issue is described further in 2.2.1, “Affinities” on page 21.

- ▶ Processor local timestamp is not unique.

Use the sysplex timestamp interface (ETOD) for a sysplex-unique timestamp. The timer affinity is further described in “Affinities” on page 21.

- ▶ Single Record Database

This may cause a “hot spot” due to increased activity. This issue is further explained in 2.2.2, “Locking” on page 24.

- ▶ Parallelism of execution may have important consequences.

Do *not* assume that subsequent executions of a transaction will run on the same system image as the previous transaction.

Do *not* assume that order of execution for a unit of work will be the same as order of arrival.

- ▶ Do *not* design an application based on an outdated software level. Parallel Sysplex is very much alive, and improvements are continuously made to system functions and subsystem functions.

So, when designing new applications, ensure that you are informed of the latest sysplex and data sharing facilities.

In the following sections, we expand on these design considerations, focusing on continuous availability and scalability.

2.2 Enabling for continuous availability

Enabling the application for continuous availability gives you the opportunity to keep it running and stable even if one or more servers should become unavailable as a result of:

- ▶ Unplanned failures on one or some (but not all) of the servers
- ▶ Planned outages in the operations of one or some of the servers in connection with reconfiguration, upgrades, maintenance

Make sure you correctly understand your expectations: Parallel Sysplex provides better availability through redundancy of servers, z/OS system images, and synchronized data management across participating system images.

You may improve continuous availability by designing the application in such a way that it supports (or even better, exploits) parallelism, as described in this chapter. Thus, when a server is not available because of a planned or unplanned outage, the other servers can take over, possibly with new system images being added.

However, even if you have designed the application to support parallelism, you may still not be completely unaffected by outages. In case of server failure, the infrastructure may not have enough capacity to run all of the applications in the remaining capacity. Although a planned outage can be carried out at a point in time where the requirement for capacity is low enough, an unplanned outage can happen at any point in time (and will likely happen at the worst possible time).

To limit capacity problems during planned or unplanned server outages, consider implementing the following:

- ▶ A granularity as small as possible for execution elements in the application. It will then be easier and quicker for sysplex workload management to redistribute the workload from the failed instance to remaining servers.
- ▶ Prioritizing your transactions to support the possibility to temporarily close low-priority functionalities. Users may not be totally unaffected, but the main functionality will still be available.

You exploit parallelism by making the application able to run concurrently on different servers as clones (that is, replicates or exactly copies accessing common shared data).

In order to do so, you should:

- ▶ Identify affinities and decide what to do with them.
Affinities are application-specific dependencies tying your execution elements from the application to be executed on a specific server—or inhibiting them from using a specific server. Affinities are further explained on 2.2.1, “Affinities” on page 21.
- ▶ Identify and reduce constraints due to many clones updating the same, shared data.
This may affect locking and cause hot spots to appear. *Hot spots* are heavily referenced data points, subject to high frequencies of locking and contention. They may also imply single points of failure. In order to avoid the problem, it may be necessary to review the application data design and organize locking at a lower level.
- ▶ Consider your batch windows.
If batch processing requires data serialization, batch may prevent you from running multiple application instances in parallel around the clock.
- ▶ Consider whether the application code could be divided into small, granular, self-containing, and possibly reusable modules in order to eliminate serialization on the code itself during execution.
- ▶ Identify and try to avoid single points of failure (SPOF) in the application.
Identify and eliminate as much as possible unique points in the application that are needed by all of the clones and are vulnerable to errors bringing all clones down. Especially unique shared data implies the risk of single points of failure. Note that the code itself is an SPOF.

- Consider exploiting versioning.

If you develop or change the application in a way that allows different versions of the application to coexist, you have the opportunity to deploy new versions to separate instances, the same way it can be done for system software. This can minimize the impact of development or maintenance errors.

Note, however, that Parallel Sysplex does *not* provide redundancy of the data and in the context of this redbook we only address continuous availability through redundancy of the system images.

Ensuring that you provide redundancy for *all* components of an application also requires that you consider coherent duplication of the data. This can be accomplished by using dual copy functions from disk technologies or, preferably, Remote Copy Management Facility (RCMF) components of the Geographically Dispersed Parallel Sysplex™ (GDPS®) solution.

2.2.1 Affinities

The purpose of enabling an application for parallelism is to ensure that any instance of that application can run on any server in the Parallel Sysplex. That way, if your transaction experiences a failure on one server, other clones of your transaction can run on the other servers, thereby providing continuous availability.

We previously mentioned two important points: affinities are application-specific dependencies tying the application to be executed on specific servers (or inhibiting it from using one or more servers), and improved availability provided by the Parallel Sysplex is based on redundancy. Therefore, if you have affinities in the application that require it to run on a unique component (such as a cryptographic co-processor) and there is a failure on that element, you will not be able to exploit the automatic restart facilities.

If you have affinities that require an application to run entirely on one single server (no matter which one), you still may be able to exploit the automatic restart facilities provided by Parallel Sysplex. But you may not exploit the scalability and workload balancing facilities. Therefore, new applications being developed should contain no affinities or as few as possible.

Descriptions of the different types of affinities follow.

Access to non-shared data

This is one of the most common affinities. If the application uses local, non-shared data, then other applications or instances of the same application that happen to be executed on another server will not be able to access it. Non-shared data can be in-storage tables or data on a non-shared data management system.

This has the following impacts:

- If the non-shared data is dynamically created and only used internally by each instance, there may not be any problem.
- Otherwise, you must ensure that the other transactions accessing the same data also will be executed in a place where they can access this data—probably on the same server, maybe in the same subtask.
- You cannot exploit scalability and workload balancing, but the automatic restart facilities are still available.

Solution

Depending on the application requirements, you may decide to tolerate the affinity or to change the application to exploit shared data.

Access to non-shared resources

Non-shared resources can be services only implemented on a subset of the servers, for instance ATMs, connections to other companies, cryptographic features, and so on.

This has the following impacts:

- ▶ This type of affinity is usually originated from the infrastructure, and applications cannot do anything about it other than make a recommendation to change the infrastructure and make these features available on all servers.
- ▶ You cannot exploit scalability and workload balancing, but the backup/recovery capability may still be available.

Solution

You have to tolerate the affinity until the infrastructure is changed.

Sequencing

You may have a demand requiring the application to process transactions in a particular sequence. It may be of great importance that transactions are processed in the order they arrive.

This has the following impacts:

- ▶ You must execute this application on one instance, but it does not have to be a specific server.
- ▶ You cannot exploit scalability and workload balancing. However, the automatic restart facilities will still be available.

Solution

You probably have to tolerate this affinity. But you may consider whether you can change the application so that it is not dependent on sequencing.

Timer

When an application is running in multiple instances on different servers, two or more events can happen at the exact same time and get the same time stamp from each processor local timer.

This has the following impacts:

- ▶ If the application does not require a unique time stamp, then there is no problem.
- ▶ Some applications do not support duplicate time stamps.

Solution

Use the sysplex timestamp interface API (ETOD) to ensure that each event receives a sysplex-wide unique time stamp.

Single memory counter

You may have a requirement for acquiring numbers in exact sequence. Depending on the software you are using, you may not have been able to acquire a unique sequence number across the sysplex.

This has the following impacts:

- ▶ If the counter is to be sysplex-wide, you must ensure that the other transactions accessing the same data also will be executed in the same instance where they can access the data.

In such cases, you cannot exploit scalability and workload balancing, but the automatic restart facilities will still be available.

Solution

If you are using CICS or DB2, you can use the provided APIs (Named Counter Server in CICS/TS, or identity columns or sequence objects in DB2) to acquire numbers in growing sequence throughout the sysplex. Alternatively, you may decide to tolerate this affinity.

Connectivity and use of APIs

Depending on the subsystems the application is connecting to, you may use functions, commands or components that introduce affinities. They may include access to non-shared data or use other non-shared features in the subsystem.

This has the following impacts:

- ▶ You must ensure that dependencies can be solved. Other transactions may also be required to execute in the same server or subtask.
- ▶ This may limit scalability and workload balancing.

Solutions

- ▶ Change the application, if possible. Look for another API call or command that exploits shared data or resources.
- ▶ If the affinity is implemented intentionally, you should consider whether it is still needed.

To reduce or avoid problems arising from affinities, do the following:

- ▶ Analyze the application and identify the occurrences of affinities.
- ▶ For each occurrence, clarify the impact.
- ▶ For each occurrence, make a decision whether you can avoid the affinity, reduce the impact, or tolerate it as it is.

The decision of whether to remove or tolerate an affinity depends on the difficulty of removing it while still meeting the needs of the business, and the extent to which it interferes with the goals you are trying to achieve with respect to parallel processing, continuous availability, scalability, and workload balancing. A low level of affinities may not interfere with these goals and need not be removed.

In fact, you may sometimes deliberately implement an affinity in order to ensure that a specific application or specific version only executes on a specific system image. This could be the case, for example, if you are installing a major system or application change and wish to deploy it on a subset of the application (for example, one instance only) to minimize risk.

For further considerations related to each of the subsystems involved, see:

- ▶ Chapter 3, “DB2 application considerations” on page 33
- ▶ Chapter 4, “DFSMTvs application considerations” on page 67
- ▶ Chapter 5, “CICS application considerations” on page 81
- ▶ Chapter 6, “IMS application considerations” on page 95
- ▶ Chapter 7, “WebSphere MQ application considerations” on page 131

2.2.2 Locking

When a transaction accesses data in a database, *locking* occurs to ensure that the database is kept consistent at any point in time. As long as only one transaction accesses specific data, no problems occur. When multiple transactions access the same data within the same time frame, locking will serialize access to the data by these transactions.

When an application is cloned throughout the Parallel Sysplex, the probability that multiple transactions access the same data increases. If the access rate to the data is too high, it may lead to *locking contention*, and the execution time for transactions competing for the same data will be degraded.

It is important that you understand this locking mechanism when designing applications, in order to minimize the risk of introducing serialization issues.

Consider reducing the number of locks required, and the duration for each lock held, as follows:

- ▶ Issue commits explicitly as soon as possible after reaching points of consistency, even in read-only situations.
- ▶ Divide your transactions into smaller recovery units, issuing more frequent commits, especially in batch.
- ▶ Organize your code to reduce locking duration. Perform updates as late as possible within the unit of recovery.
- ▶ Issue rollback explicitly after a failure, thereby releasing the locks.
- ▶ Partition your data and organize access in a way that reduces the probability that several instances will access the same database partitions at the same time. In DB2, locks are minimized and performance improved if a partition is only accessed from one server at a time.

For more information about database partitioning, see 3.4, “Example application” on page 62.

- ▶ Access data in the same order from different servers to reduce the risk of deadlocks. A cloned transaction will do so—but you should also design different transactions to do so if they access the same set of data.
- ▶ Read without integrity; if the application can deal with this, you can avoid locks. However, be aware there is a risk that you may obtain uncommitted data that could possibly be backed out.
- ▶ Use the proper access level. If you only want to read, do not use the access level for update.
- ▶ Try to minimize the number of calls to the database manager.
- ▶ Database tuning can also help reduce the time locks are held.

See Chapter 3, “DB2 application considerations” on page 33 and Chapter 6, “IMS application considerations” on page 95 for detailed considerations concerning DB2 and IMS/DB.

Hot spots

A hot spot is typically a data object that is subject to extremely high access rates. As with locking problems, hot spots arise from the needs of parallel processing and the resulting multiplication of accesses. The consequence is reduced performance due to serialization of the shared data in the Parallel Sysplex. Hot spots must be handled with considerations similar to the ones given for locking.

2.2.3 Batch window

A *batch window* is a time frame in which online transactions are stopped in order to run batch jobs. In many companies, a daily and possibly a weekly batch window is implemented. The online transactions may be stopped in different ways, either by operational intervention supported by system automation mechanisms or by application means (for instance, by a field in a database).

Some applications may not present a problem if they are unavailable for an hour or two during nights or weekends. And even if your online transaction is not available 24x7, you can still enjoy sysplex features such as parallel processing, continuous availability, scalability, and workload balancing.

But your batch window may grow because of increasing amounts of data to process. And you may face the need to extend the operating hours of your application. More and more, applications require 24x7 availability.

In banks, for example, self-service applications are increasingly provided to customers, and they want to be able to access their accounts at any time of day or night. Other applications deal with several time zones around the world. Batch windows must be avoided in these applications.

To reduce or eliminate the impact of batch windows, you must consider the following issues:

- ▶ Can you do without batch? Or can your existing batch functionality be converted to online transactions?
- ▶ Is it possible to parallelize your batch jobs, thereby reducing the batch window size? If you choose to parallelize your batch, you may reduce locking and improve performance by partitioning the tables involved and letting each batch stream refer to its own partition.
- ▶ Can your batch jobs run in parallel with online transactions? Pay attention to the constraints arising from locking, especially when accessing common data with high frequency.
- ▶ Do you have batch logical end-of-day processing, during which you do not want additional online transactions to interfere? Logical end-of-day imposes a time dependency that requires you to establish a sysplex-wide synchronization point across all instances of an application. There may be a need to process data related to certain time intervals, such as state data of accounts and so on.

If your database design is event-based, by adding rows instead of updating rows you may be able to decide which data is related to the time period you want to process. As an alternative, you may also consider a messaging solution using WebSphere MQ.

2.2.4 Single points of failure

Fundamentally, a single point of failure (SPOF) is a point in the application where errors potentially could inhibit continuous availability for the application and any dependent. If a failure or corruption should occur, all application instances using this table will be affected, regardless of the number of clones you may have.

Shared data, especially, implies the risk of having single points of failure. Typically, it is the role of the infrastructure management to secure the shared physical data. But data errors may result from applications that are not functioning correctly; for example, a transaction on one instance may corrupt a data table shared with other instances, thereby creating a problem for other transactions on all instances in the sysplex.

You can avoid single points of failure by implementing the following:

- ▶ Replicating critical data and allowing applications to access different replicates, if possible.
- ▶ Partitioning your data and organizing access in a way that reduces the probability that multiple instances are accessing the same partition at the same time.
- ▶ Dividing your transactions into smaller units and isolating the SPOF (for example, inside separate data access modules).

2.2.5 Performance

A sysplex configuration provides multiple advantages, such as managing multiple servers as one single system image, improved availability through managed server redundancy, and more stable response time through workload distribution. However, since additional system code is required to keep the participating systems synchronized and maintain data integrity, a sysplex installation usually requires slightly more CPU than a comparable non-sysplex one.

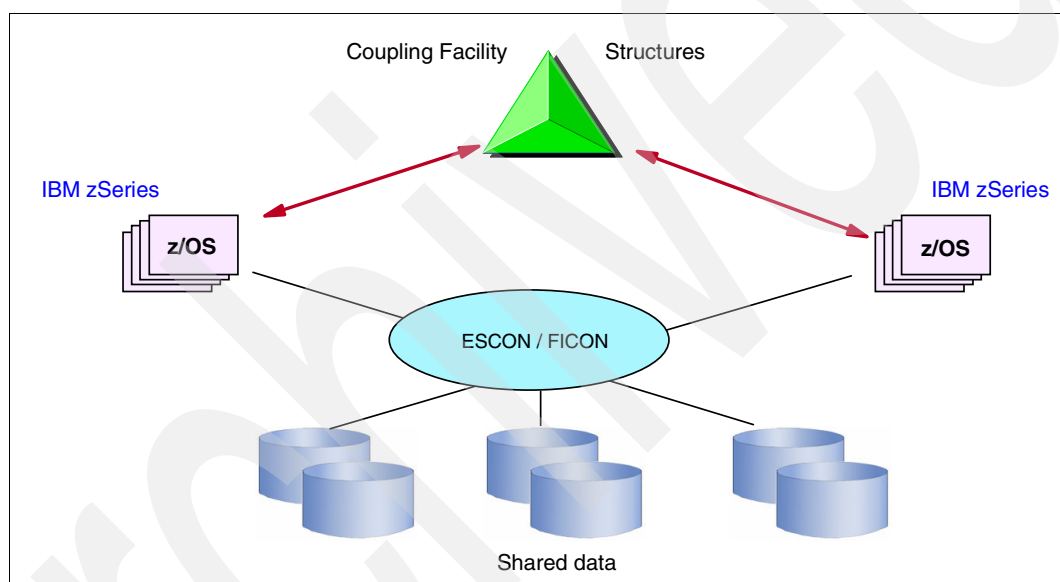


Figure 2-2 CPU overhead

Most of the CPU increase comes from the additional code that is required to handle communication between the participating z/OS system images and the Coupling Facility Control Code, to manage the structures contained in the Coupling Facility.

This overhead can be split into two parts:

- ▶ System management, which varies with the number of system images participating in the sysplex. It is mostly an infrastructure cost, and applications have little or no influence on it.
- ▶ The second part is related to data sharing and is a direct consequence of lock management required to manage the structures for the database subsystems using data sharing. It varies according to the data sharing activity from all applications. Recommendations on locking provided in this redbook will help you keep the CPU overhead to a minimum.

Although much was written on the subject of overhead in the early days of sysplex, many technology enhancements have been realized and sysplex overhead is no longer a significant issue. In a typical multi-system sysplex installation today, the additional cost for both the base sysplex and data sharing usually remains contained well below 10%.

2.3 Step-wise deployment/versioning

Parallel Sysplex provides you with the possibility to run different versions of an application on different servers, depending on the infrastructure setup in your installation. You can deploy a new version of the application on one or more servers and let a subset of users run on this version before you release the version on the other servers and to all the other users. Different front-end mechanisms can help direct users to the intended servers.

To exploit this function, you must develop the application in a way that makes it possible for (at least) two different versions to coexist. In this section, we describe some techniques to handle this problem.

There are two aspects of coexistence:

- ▶ On the code level, you may need different versions of the code to be able to communicate via specific management APIs.
- ▶ On the data level, you probably need the different versions to access the same shared data.

When new subsystem levels are rolled out sequentially on multiple servers, you have to wait before you make new functionalities available to all instances of the application. With new application levels you may want to ensure that the new functionalities work correctly on specific servers before you upgrade the rest of the servers.

Backward compatibility

This is probably the most usual way of establishing coexistence. The rule is that the new version ($n+1$) has the responsibility to coexist with the older version.

You may not have to do anything with the old version. If the new version accesses new columns that have been added to a table, the old version may still be able to do its usual work on the data it knows. Naturally, you must not use *select from ** in your programs.

Forward compatibility

Another technique requires the old version to tolerate the new version. When a version is developed you may not know what to expect in the next version, so when you are close to deployment of the new version you may have to provide a “toleration” change to the old version and deploy this in all servers before the new version can be deployed. The toleration may relate to new or changed functions or data fields.

Services

If multiple applications or parts call common services from other applications or parts, it is usually necessary to define an agreed interface; see Figure 2-3 on page 28.

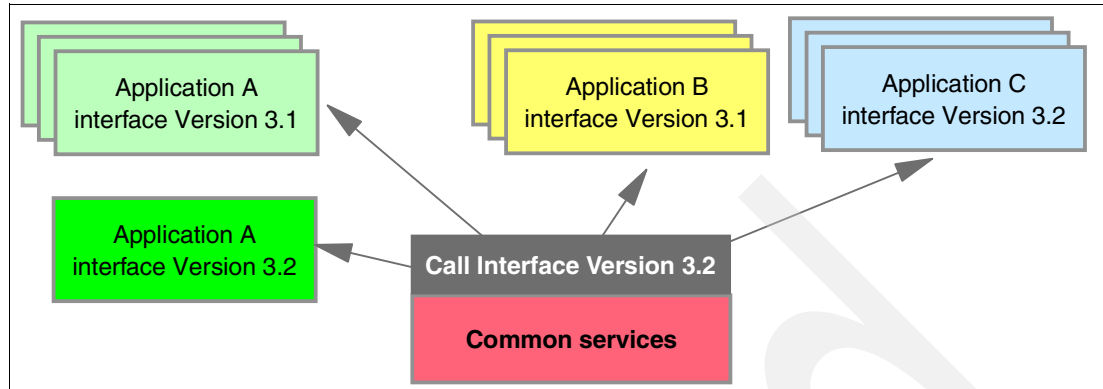


Figure 2-3 Common services accessed by multiple applications

The interface should include information about which level of services are to be provided. Typically, the call interface should contain information about the supported interface levels (basic service interface information, plus version and level information).

Version awareness

You may let the application add and check information on the current version of the data. When the application references the data, it may check whether it can understand and support the current level of the data, and what it must do if they do not match. And when it updates or adds a row, the application must update the current version number field.

You may add a version number to all interfaces between parts of an application. The called/invoked part must check whether the desired version is supported. If not, it should give a sensible return code to the caller. Generally speaking, it is a good idea to support at least two levels in your interface.

You may also consider checking version compatibility with the other applications or instances when the application starts.

Develop the application to react intelligently if it is confronted with a version that it does not know. Depending on the impact, you may decide to let it do the part of the functionality that it recognizes and return a sensible reason code; this may be better for continuous availability than terminating abnormally. But it is important that any discrepancies are reported (for example, through writing an error or information message) so that someone can look into the problem.

Release management and version control

Consider gathering changes to the application in large but less-frequent releases; that way you will not have to cope with change so often. On the other hand, you may have customer requirements for more frequent changes, not only in order to fix errors but also to provide new functionality.

With few, large releases, you may have a greater need for developing different versions in parallel. When one release of the application is close to being put in production, you may already have started work on the next release. This may imply increasing demands for version control, whether done manually or by using advanced tools. It is always a challenge to ensure that a change made to an older version is implemented or dealt with in subsequent versions as well.

It is a good idea to number or otherwise identify all changes and keep track of which changes should be implemented in which versions—possibly even add a change control number to the code.

Front-end distribution and workload balancing

If you have deployed a new version on one or more (but not all) running instances, you probably want only a subset of the users to use this new version. This is generally one fundamental reason for deploying step-wise, and the users must use the new version every time to ensure consistency.

2.4 Other considerations

These are more general z/OS application considerations to keep in mind when designing applications, and most of them are not exclusively intended for sysplex environments. However, we mention them here as a reminder because experience has shown that it is easier to start exploiting sysplex facilities when all previous z/OS application development recommendations are already in place.

2.4.1 Granularity, modularity, and generally reused modules

General development methodologies encourage a strict division in small entities (or objects) and a service-oriented approach, where use of functions and access to data is encapsulated. These principles are in harmony with your needs for exploiting Parallel Sysplex.

- ▶ Small, granular modules are easier to maintain. The units of recovery are more granular and help to minimize locking constraints. It is also easier to minimize errors and implement fault tolerance.
- ▶ Reuse of broadly used and well-tested modules provides more stability. However, a risk is that you implement a single point of failure, so general modules must be subject to very strict development, testing, and quality assurance procedures.
- ▶ Small modules provides more granular workload balancing.
- ▶ Prioritize transactions and keep the application running if one or more of the lowest priority transactions becomes unavailable. You may suspend low-priority transactions if your capacity is low as a result of a server outage.
- ▶ Small modules make it easier to provide step-wise deployment and versioning as long as the changes relate to internal (encapsulated) functionality. Changes to the interface may be more difficult, especially if you are going to change the interface for a widely reusable module. Modules used by multiple applications must always support different versions of the interface, so you do not have to change the users of the interface along with the module itself.
- ▶ Only the data owning application's modules must be allowed to access the data. Or even better, only one object must access specific data related to this object. These modules ensure that data is consistent at any time, and that problems arising from locking and hot spots are minimized.
- ▶ Data access modules must be available throughout the Parallel Sysplex in order to be consistent with the data sharing.
- ▶ Register applications that use each of the generally reused modules. Then, know who to contact for support and maintenance.

- ▶ The users of data, as opposed to provider modules, do not have to care about data sharing affinities and locking. By reusing modules you can reduce the number of places where you have to be concerned about the affinities and other inhibitors to parallelism.

Note the following considerations:

- ▶ You must provide version compatibility up to a certain degree. General and other heavily reused modules must be upgraded once in a while; however, this must not require all users of the module to change synchronously. Add a version number in the interface giving the users a time frame to change, if ever.
- ▶ Widely reused modules imply a risk of creating single points of failure; if there is an error in the code, all application instances using the module are affected, so make sure you test all code thoroughly before implementing it in a module.

2.4.2 Fault tolerance

Errors in applications cannot be avoided and when they occur, you must have some kind of recovery procedure in place. If the application is running on a single image outside a sysplex and an error occurs, the application will not be available until you have found a solution and recovered from the error.

If the application is running inside a sysplex, you maybe able to recover immediately depending on the type of error you are facing:

- ▶ If the error is in the application, the situation is not much different than without sysplex; you still have to locate the error, find a solution, and recover.
- ▶ If the error is in the infrastructure, it is very likely that the infrastructure itself will recover using sysplex backup/recovery capabilities. However, you may not be completely unaffected by the error; the application may have a return code from an API call that says that a particular service is not available.

Also keep in mind that the code itself can be a single point of failure, so you focus on avoiding errors and possibly enabling for fault tolerance.

2.4.3 Debugging and tracing

Debugging and tracing in a Parallel Sysplex is different from in a non-sysplex environment, because most of the time you do not know which image your transactions are executing on or where they will be executed next time. As a result, although debugging and tracing facilities are still available, it is more difficult to locate and interpret the information related to one specific transaction malfunction.

At the time of writing, we are not aware of any development tool that completely solves this problem. If the application is service-oriented or uses generally reused modules, you may consider developing tracing facilities into your code:

- ▶ Build tracing capability in your general modules.
- ▶ Depending on the input from a caller or from another source, choose a tracing level and write a message to a database or a queue (consider using WebSphere MQ). You must have different tracing levels and be able to turn tracing on or off.
- ▶ Consolidate the trace messages in one place. Again, WebSphere MQ is useful for gathering messages from different sources.
- ▶ Develop a simple tool to read and format tracing data and clean up the trace database.

2.4.4 Testing environment

The most important feature of the testing environment is to provide a similar architecture and setup as in production. This includes all the aspects of Parallel Sysplex including CICS/TS, IMS, DB2, TVS, WebSphere MQ, and any implied environment-specific affinities.

You may have modified special IMS or CICS exits, and you must be able to test them at the same time as the rest of applications. You must also provide cryptographic features, ATMs, or connections to external companies in the same manner, or at least simulate them.

There are many other issues that affect the ability to do proper testing; these are not specific to Parallel Sysplex but are general recommendations for a sound development and test environment, as follows:

- ▶ Is the application well maintained at the testing environment?
- ▶ Can you execute any production program or module in the testing environment without problems (for example, obtaining -805 sqlcode code)?
- ▶ Can you recreate the application from production if you failed with a new development of an old application?
- ▶ Do you have the proper data for testing all of the capabilities of the applications?
- ▶ Are all of the applications prepared to give the services that other applications want?
- ▶ Can you test all of the changes of different applications that are interconnected at the same time?

If you have a high level of reusability, you can develop much quicker, but the applications become interdependent and you may find it more difficult to test them at the same time.

You may create more than one test environment. But with this solution, you must be careful to keep all of the software that is installed in the production environment active in all the test environments as well. Developers must also ensure that all data in the different test environments is well maintained in order to allow testing to the rest of the applications.

Archived

DB2 application considerations

In this chapter we discuss the advantages of DB2 data sharing and the use of Coupling Facility structures by data sharing members. Considerations for exploiting DB2 data sharing are provided for both application developers and system administrators.

The chapter contains the following:

- ▶ “Introduction to DB2 data sharing” on page 34
- ▶ “Application considerations for developers” on page 40
- ▶ “Application considerations for DB2 system administrators” on page 56
- ▶ “Example application” on page 62

3.1 Introduction to DB2 data sharing

A *data sharing group* is a collection of one or more DB2 subsystems that access shared DB2 data. The data sharing function of the licensed program DB2 UDB for z/OS enables applications that run on more than one DB2 subsystem to read from and write to the same set of data concurrently.

DB2 subsystems that share data must belong to a DB2 data sharing group that runs on a Parallel Sysplex. A Parallel Sysplex is a collection of z/OS systems that communicate and exchange information with each other.

Each DB2 subsystem that belongs to a particular data sharing group is a member of that group. All members of a data sharing group use the same shared DB2 catalog and directory. Currently, the maximum number of members in a data sharing group is 32. For more information, refer to *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417.

3.1.1 Advantages of data sharing

DB2 data sharing improves the availability of DB2, extends the processing capacity of your system, provides more flexible ways to configure your environment, and increases transaction rates. You do not need to change SQL in the applications to use data sharing, although some tuning might be needed for optimal performance.

Improved availability of data

More DB2 users demand access to DB2 data 24 hours a day, 7 days a week. DB2 data sharing helps you meet this service objective by improving availability during both planned and unplanned outages.

As Figure 3-1 on page 35 illustrates, if one subsystem becomes unavailable, users can access their DB2 data from another subsystem. Transaction managers are informed that the DB2 subsystem is down and can switch new user work to another DB2 subsystem in the group.

For unplanned outages, z/OS Automatic Restart Manager (ARM) can automate restart and recovery of the subsystems and application instances.

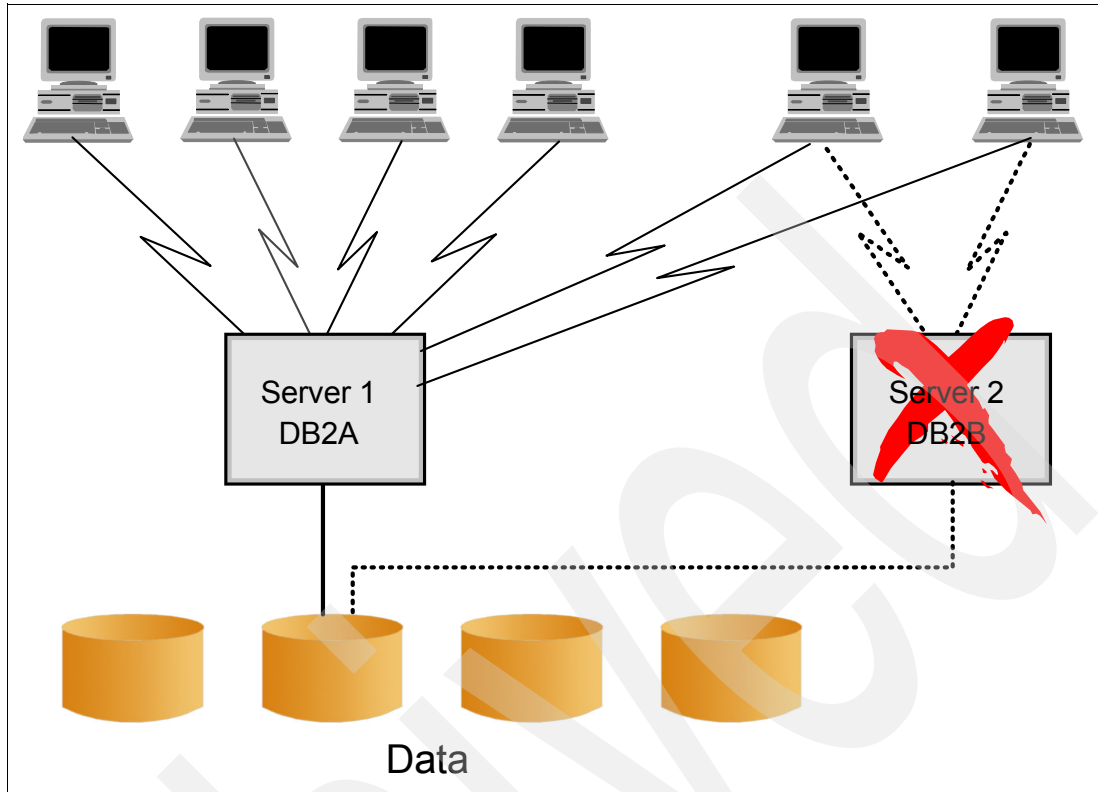


Figure 3-1 Data sharing improves availability during outages

While DB2's increased availability has some performance cost, the overhead for inter-processor communication and caching changed data is minimized. DB2 provides efficient locking and caching mechanisms implemented in a Coupling Facility (CF).

A CF is a logical partition that runs the Coupling Facility control program, and it is an integral part of a Parallel Sysplex. A Coupling Facility provides high-speed caching, list processing, and locking functions in a sysplex. Much of the communication between DB2 members in a data sharing group is accomplished by each member sending information to the CFs and querying the CFs for information from the other members.

Scalability

With DB2 data sharing, you get the following benefits:

- ▶ Incremental growth

The data sharing group can grow incrementally. You can add a new DB2 onto another LPAR which is part of the Parallel Sysplex and access the same data through the new DB2. You no longer have to manage copies or distribute data. All DB2 subsystems in the data sharing group have concurrent read-write access, and all DB2 subsystems share a single DB2 catalog and directory.

- ▶ Workload balancing

DB2 data sharing provides flexibility for growth and workload balancing. DB2 does *not* use the partitioned data approach to parallelism (sometimes called the shared-nothing architecture) where a one-to-one relationship exists between a particular database management system (DBMS) and a particular subset of data. As a result, data in a DB2 data sharing environment does not have to be redistributed when a new subsystem is

added or if the workload becomes unbalanced. The new DB2 member has the same direct access to the data as all other existing members of the data sharing group.

DB2 works closely with the z/OS component Workload Manager (WLM) to ensure that incoming work is optimally balanced across the systems in the group. WLM manages workloads that share system resources and have different priorities and resource usage characteristics.

- **Capacity when you need it**

A data sharing configuration can handle your peak loads. You can start data sharing members to handle peak loads (such as end-of quarter processing), and then quiesce them when the peak passes.

Since a data sharing group can have more DB2 subsystems and the extra data sharing cost has been minimized, you can now process more transactions concurrently and increase throughput.

Sysplex query parallelism enables DB2 to use all the processing power of the data sharing group to process a single query. For complex data analysis or decision support, sysplex query parallelism is a scalable solution. Because the data sharing group can grow, you can direct more power to those queries even as those queries become increasingly complex and process larger and larger sets of data.

Flexible configurations

Data sharing lets each set of users access the same data, which means that you no longer need to manage copies.

DB2 data sharing also lets you configure your system environment much more flexibly. It is possible to have more than one DB2 data sharing group on the same z/OS sysplex; you might, for example, want one group for testing and another for production data.

Flexible decision support systems

Figure 3-2 on page 37 shows two different decision support configurations. A typical configuration separates the operational data from the decision support data. Use this configuration when the operational system has environmental requirements that are different from those of the decision support system. The decision support system might be in a different geographical area, or security requirements might be different for the two systems.

DB2 offers another option—a combination configuration. This configuration combines your operational and decision support systems into a single data sharing group and has these advantages:

- You have the capability to join decision support data and operational data with SQL.
- You can reconfigure the system dynamically to handle fluctuating workloads. (You can dedicate servers to decision support processing or operational processing at different time periods.)
- You can reduce the cost of computing:
 - The infrastructure used for data management is already in place.
 - You can create a prototype of a decision support system in your existing system, and then add processing capacity as the system grows.

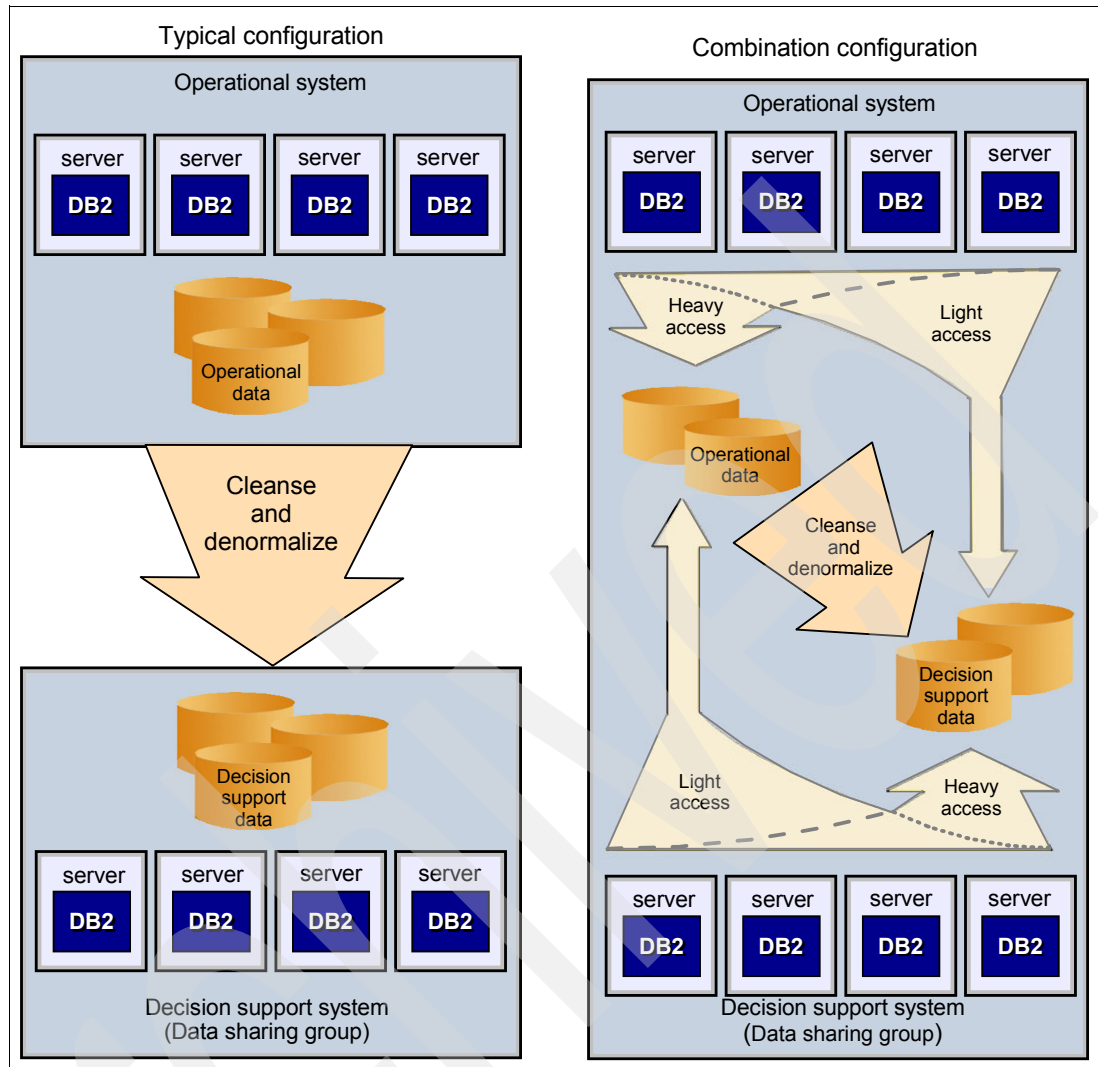


Figure 3-2 Flexible configurations for decision support

If you want to configure a combination system configuration, you must separate decision support data from operational data as much as possible. Buffer pools, disks, and control units that you use to decide on a support system should be separate from those that are used in your operational system. This separation greatly minimizes any negative performance impact on the operational system.

If you are unable to maintain that level of separation, or if you have separated your operational data for other reasons such as security, then using a separate decision support system is a better option.

Flexibility to manage shared data

Data sharing can simplify the management of applications that must share some set of data, such as a common customer table. Perhaps these applications were split in the past for capacity or availability reasons. But with the split architecture, the shared data must be kept in synchronization across the multiple systems (that is, by replicating data).

Data sharing gives you the flexibility to configure these applications into a single DB2 data sharing group. It also allows you to maintain a single copy of the shared data that can be read and updated by multiple systems with good performance. This is an especially powerful option when data centers are consolidated.

Leave application interface unchanged

Your investment in people and skills is protected because existing SQL interfaces and attachments remain intact when sharing data.

You can bind a package or plan on one DB2 subsystem and execute that package or plan on any DB2 subsystem in the data sharing group.

3.1.2 How data sharing works

A DB2 data sharing group requires three types of structures to be allocated in a Coupling Facility. There will be a lock structure, a shared communications area (SCA), and one or more group buffer pools. Also, although the log data sets and BSDS are not shared, each member must have access to the data sets of all the other members.

Lock structure

Each member of the data sharing group will have its own IRLM. However, since it is necessary to provide concurrency across all the members, certain lock information is propagated to the lock structure.

One part of this structure is the *lock table*, which contains lock status information and the members that own those locks. This provides global lock serialization. The other part is the *modified lock list*, which contains lock information that must be maintained in case of the failure of any of the members. The locks held by the failed member must be retained until that member is restarted and the updates to the DB2 objects can either be committed or backed out.

SCA

Control information from each member of the group is passed to the SCA. Database exception conditions and other information needed for recovery is contained here.

Group buffer pools

Applications can access data from any DB2 subsystem in a data sharing group. Many subsystems can potentially read and write the same data. DB2 uses special data sharing locking and caching mechanisms to ensure data consistency.

When multiple members of a data sharing group have opened the same table space, index space, or partition, and at least one of them has opened it for writing, then the data is said to be of inter-DB2 read/write interest to the members. To control access to data that is of inter-DB2 interest, whenever the data is changed, DB2 caches it in a storage area that is called a group buffer pool.

When there is inter-DB2 read/write interest in a particular table space, index, or partition, it is dependent on the group buffer pool, or GBP-dependent (group buffer pool-dependent).

As shown in Figure 3-3 on page 39, a mapping exists between a group buffer pool and the buffer pools of the group members. For example, each DB2 has a buffer pool named BP0. For data sharing, you must define a group buffer pool (GBP0) in the Coupling Facility that maps to buffer pool BP0. GBP0 is used for caching the DB2 catalog and directory table spaces and indexes, and any other table spaces, indexes, or partitions that use buffer pool 0.

Although a single group buffer pool cannot reside in more than one Coupling Facility (unless it is duplexed), you can allocate any group buffer pool in any of the Coupling Facilities that are part of that Parallel Sysplex.

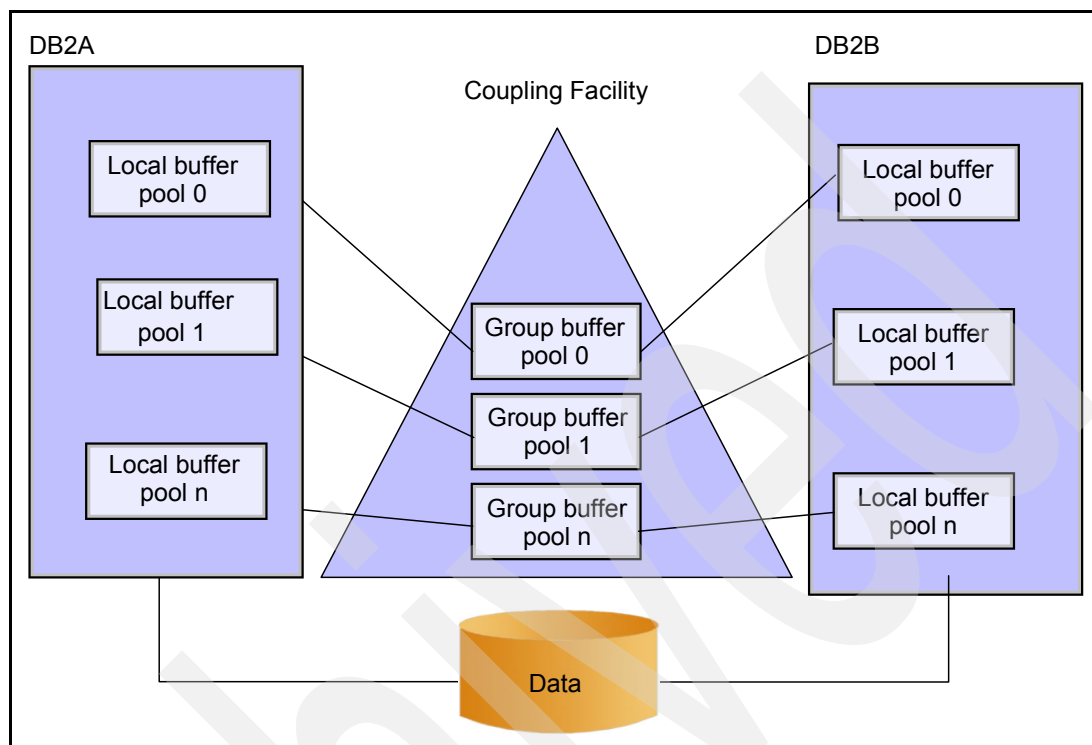


Figure 3-3 Group buffer pool correspondence to local buffer pools

When a DB2 subsystem requests an index or data page, the local buffer pools are searched first. If the page is not found, the group buffer pool is then searched. If the page still has not been found, the I/O process is invoked to retrieve the page from DASD.

When an application changes a particular page of data, DB2 caches that page in the group buffer pool. The Coupling Facility invalidates any image of the page in the buffer pools associated with each member. Then, when a request for that same data is subsequently made by another DB2, that DB2 looks for the data in the group buffer pool.

Logs and BSDS

Each member of the DB2 data sharing group will have its own set of active log data sets, archive log data sets, and bootstrap data sets. These data sets must be available for read access from all of the members of the group, but can be updated only by the member who owns them.

When it is necessary to recover a DB2 object, the Recover utility can be run on any member of the group. The shared catalog contains the information about the image copy and the log ranges that must be applied. However, each member's BSDS must be read to determine whether any logs from that particular member must be included in the recovery process.

A single DB2 uses the log relative byte address (RBA) to determine the order of updates. However, a data sharing group needs a different mechanism for determining the order of updates among the multiple subsystems. Each DB2 will generate a value based on the time-of-day clock of its processor. This value is called the log record sequence number (LRSN). It will be consistent among all the members of the group because of the sysplex timer that is required for the parallel sysplex. The LRSN is used in the recovery process instead of the RBA.

3.2 Application considerations for developers

DB2 data sharing introduces no changes to the application interfaces to DB2. A program that executes successfully in a non-data sharing environment will also execute successfully in a data sharing environment. This does not mean that any or all applications should be executed in a data sharing environment unchanged. A number of application changes must be considered to optimize the benefits of a Parallel Sysplex environment.

Lock avoidance and frequent commits are required to achieve good application and system performance. Designing or modifying applications to minimize commit scopes and to release locks quickly can also improve application availability. Locking is the most significant consideration for the application designer and developer in a Parallel Sysplex environment.

Application design and database design should complement each other to realize the benefits of workload balancing, increased capacity and reduced elapsed time. Routing work to the server that has the most available capacity can increase transaction throughput while reducing user response time. Critical batch job streams can complete more quickly if multiple servers in the Parallel Sysplex execute the jobs in parallel. Effective design will enable these benefits while minimizing the overhead related to CF traffic.

Most of the considerations in this section apply to any workload. Some are specific to the particular requirements of batch processing. All should be part of the design process for new applications, and they should be included where appropriate in existing applications, especially during application maintenance activities.

3.2.1 Locking

In any DB2 environment, it is desirable to reduce the number and duration of the data base locks held by an application. Long duration locks can effect transaction response times and application elapsed times and may lead to timeouts or deadlocks. Locks are managed by the IRLM and each lock requires processing and storage. The more locks the IRLM must manage, the more storage and CPU are required.

When an application holds a lock on an updated resource, other applications that want the resource must wait for the lock to be released. This can lead to elongated response times or elapsed times for the waiting applications. This can also lead to timeouts for the waiting DB2 applications. When two applications each hold locks that the other application is waiting for, a deadlock occurs, and one of the applications will be abended. Timeouts and deadlocks cause the roll-back and loss of all previous uncommitted work of the failing applications. Handling timeouts and deadlocks consumes CPU cycles and time, and the application which failed must process the work again.

In a data sharing environment, some locks must be propagated to the lock structure in the Coupling Facility. Each access to the Coupling Facility involves CPU overhead and service time. This overhead and service time is in addition to the CPU and time that is required for local locking. Deadlock processing is also more expensive in a data sharing environment.

Application developers and designers can minimize these costs of locking in a data sharing environment by producing applications that optimize lock avoidance and by managing locking overhead. The more important of these is optimizing lock avoidance.

3.2.2 Lock avoidance

Lock avoidance is a technique which DB2 uses to reduce the numbers of locks the IRLM must manage. DB2 will avoid taking a read lock on a page if it can establish that the page contains no uncommitted data. By avoiding read locks DB2 reduces CPU overhead and elapsed time. In a data sharing environment, lock avoidance is important in reducing requests to the lock structure and thereby reducing CF-related traffic and overhead. Application developers should understand the bind parameter settings that achieve the most lock avoidance, the DB2 mechanisms that determine if a lock can be avoided, and the programming techniques that will optimize lock avoidance. Effective lock avoidance should be part of the application design process.

BINDPARM settings

Refer to 3.3.1, “BIND options” on page 57 for more details on this subject.

The application package or plan settings determine how much lock avoidance is possible. The following settings are listed in decreasing order of lock avoidance:

- ▶ **ISO(UR)**
A program bound with isolation level Uncommitted Read takes almost no locks and avoids all L-locks. ISO(UR) be chosen if the application can tolerate reading uncommitted changes to rows.
- ▶ **ISO(CS) CURRENTDATA(NO)**
A program bound with isolation level Cursor Stability and CURRENTDATA (NO) can avoid locks on pages containing qualifying or non-qualifying rows. Most application processes that cannot tolerate reading uncommitted data should be bound with ISO (CS) and CURRENTDATA (NO).
- ▶ **ISO(CS) CURRENTDATA(YES)**
A program bound with isolation level Cursor Stability and CURRENTDATA (YES) can only avoid locks on pages containing non-qualifying rows. This is the default setting for ISO(CS). The fact that CURRENTDATA (YES) is the default means that in many customer sites DB2 is not avoiding the optimal number of locks.
- ▶ **ISO(RS)**
For programs bound with isolation level Read Stability, some lock avoidance is possible for pages containing unqualifying rows if EVALUATE UNCOMMITTED is set to YES on installation panel DSNTIP4.
- ▶ **ISO(RR)**
Repeatable Read is the default isolation level and avoids no locks. A plan or package bound with repeatable read holds locks on all pages touched until commit. This is the least desirable setting for lock avoidance and for concurrent access to data by multiple processes.

Lock avoidance mechanisms

The following is a brief introduction to how DB2 determines if a lock can be avoided. For a detailed discussion with illustrations, refer to the IBM Redbook *Locking in DB2 for MVS/ESA Environment*, SG24-4725. There is also extensive information on lock avoidance presented in *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413.

Non-data sharing

In a non-data sharing environment, DB2 maintains a Commit Log Sequence Number (CLSN) for each table space or partition. The CLSN for a table space is the beginning log RBA of the oldest active unit of recovery for that table space. The CLSN represents the earliest point where uncommitted data for the table space could appear on the DB2 log.

When an application process reads a page, DB2 attempts to avoid taking a read lock on the page. DB2 makes this attempt for read only cursors and ambiguous cursors. DB2 compares the CLSN for the table space or partition in question with the page RBA. The page RBA indicates the log RBA when the page was last updated. If the page RBA is less than, or older, than the CLSN for the table space or partition, then the page cannot have been updated during any active unit of recovery. This means the page contains only committed data, and a read lock can be avoided.

If the page RBA is greater than, or newer, than the CLSN, there may be uncommitted rows on the page. In that case, DB2 checks the row header for a bit that indicates whether the row may contain uncommitted data. If the *possibly uncommitted data* bit, or PUNC bit, is not set, the row does not contain uncommitted data. DB2 then checks to see if the row qualifies the SQL. If the row qualifies, DB2 checks the BIND parameter *currentdata* setting. If YES, then DB2 takes a lock on the page and returns the qualified row to the application. If NO, then DB2 avoids the lock and returns the qualifying row to the application.

If the PUNC bit is set, DB2 takes a read lock. The important issue is that PUNC bits do not get reset on a regular basis. This means lock avoidance is much more effective if the page passes the CLSN test. The more frequently application update processes issue commits, the more current the CLSN value is and the better the lock avoidance. In a non-data sharing environment, the lock avoidance test for each table space or partition is based on the CLSN value for that table space or partition. This is also true in a data sharing environment for table spaces or partitions that are not being shared between members.

Data sharing

In a data sharing environment, there is a significant difference between lock avoidance for shared table spaces or partitions and for non-shared table spaces or partitions. Lock avoidance for all shared objects (that is, table spaces or partitions subject to inter-DB2 read-write interest) is based on a single CLSN for the DB2 data sharing group, or the Group CLSN (GCLSN). Each DB2 data sharing member identifies the oldest, or lowest, CLSN in the DB2 subsystem, regardless of whether the corresponding table space or partition is being shared with other DB2 members. From these oldest CLSN values in each member, the overall oldest one is chosen to be the GCLSN.

The GCLSN is based on the log record sequence number (LRSN) rather than the log RBA. The GCLSN is then compared with the page LRSN (instead of the page RBA) for the first lock avoidance test. Thus, for pages in any shared object, lock avoidance compares the GCLSN to the page LRSN.

A long-running process on one member that changes a table (insert, update, or delete) and commits infrequently can cause *all* lock avoidance checks based on the GCLSN to fail. All lock avoidance checks for shared objects would then be based on the PUNC bit test. Since many of these tests would also fail, DB2 would take many page locks. This can have a dramatic effect on locking volumes and system and application performance. In one case, a customer observed that the number of IRLM locks per minute increased by an order of magnitude (that is, 10 times increase) and transaction response time increased over 100 times. This case was due to a single distributed application that issued an update but did not commit.

Programming techniques for lock avoidance

There are two aspects of lock avoidance that apply to an application program. One is how frequently the program issues commits, which contributes to keeping the GCLSN current and improves lock avoidance for all processes in DB2. The other is how the program can be coded to avoid the most locks while maintaining data integrity.

In a data sharing environment, it is critical that *all* application processes that access DB2 tables commit frequently, even if they do not participate directly in sharing data with other DB2 members. The only exception is if they always execute in a period when there is no active sharing of data between members.

For new applications, make sure all processes are designed to commit frequently if they execute when objects are being shared. In the case of existing applications, long-running processes that commit infrequently must be identified as early as possible before implementing data sharing. Then they must be analyzed and, if they execute when objects are being shared, they must be modified to commit frequently.

Refer to 3.2.3, “Commit frequency” on page 45 for a general discussion of commit frequency and to 3.3.3, “DB2 installation options” on page 59 for information on how DSNZPARM values URCHKTH and URLGWTH can identify processes that may need to commit more frequently.

Writing applications to avoid locks

Most application programs should be written to execute in packages bound with ISO(CS) and CURRENTDATA (NO). For read-only cursors and ambiguous cursors DB2 will attempt to avoid read locks. Some programs, especially ad-hoc queries or reports, can be written to take advantage of ISO(UR); such programs will take almost no locks and no child L-locks will be propagated to the lock structure. In either case, ISO (CS) with CURRENTDATA (NO) or ISO(UR), the application programs will read data from pages or rows that are not protected by locks. For most read-only applications, this will not present any difficulties. In the case of ISO(UR), the application may read rows that were updated but not committed at the time the ISO (UR) program read them, but were subsequently roll back.

For update-type programs, which issue UPDATE or DELETE statements, there are additional considerations. If the application uses an UPDATE CURSOR and issues UPDATE WHERE CURRENT OF CURSOR to make its updates, the underlying data pages and rows are protected by update locks during the application process. But if the application uses a read-only cursor and issues its updates as independent statements, there is a potential exposure. Because the pages or rows the application reads may not be protected by locks, the data in the table can be changed by another process while the application continues to execute. The application must take steps to ensure that it does not overwrite changes that other processes may have made.

Consider the following example:

PgmA uses a read-only cursor and fetches 100 rows. No locks are taken on the table because PgmA is bound with ISO(CS) and CURRENTDATA(NO). As it fetches each row, it performs other SQL or executes other logic to determine whether the row should be updated. PgmA identifies 10 rows to update, rows 10, 20, 30, and so on. PgmA creates the update statements for each row but saves the update until the end of the unit of work. At the end of the unit of work, PgmA issues 10 update statements and commits.

Meanwhile, after PgmA has fetched row 20 and before it has updated row 20, PgmB issues an update to row 20. The change PgmB makes is to a column or columns that PgmA will also update. Pgm B commits. Row 20 no longer contains the columns that PgmA previously read into program storage. PgmB had no difficulty obtaining the page or row lock for row 20, because PgmA held no read locks.

When PgmA commits, it will overwrite row 20. The update PgmB made is lost.

There would have been no problem if PgmA had declared an update cursor, as an update lock would have been held on the row, or page, for row 20, preventing PgmB from issuing its update. Nor does this problem occur if PgmA is bound with ISO(CS) and CURRENTDATA (YES). The exposure only exists if PgmA is bound with CURRENTDATA (NO), avoids locks and issues an unqualified update of row 20. Instead PgmA should issue a qualified update, based on the column values of row 20 that it fetched originally.

There are several ways that PgmA can qualify its update. If the table has an audit column, based on a timestamp or a control number, where that column value is increased each time the row is updated, then PgmA could issue the update statement with a WHERE clause specifying that the timestamp or control number was equal to the value originally fetched. Otherwise, PgmA should qualify the update statement with a WHERE clause that specifies the 'before' value for each column it intends to update. This last option has the advantage that it does not rely on whether PgmB updates the timestamp or control number. In any of these cases, an SQL return code +100 on the update statement means the row has changed, and PgmA should fetch the row and execute the program logic for that row again. PgmA would then 'see' the change PgmB made, so the update PgmB made would not be lost.

Here is the same example but with the qualified update:

PgmA uses a read-only cursor and fetches one hundred rows. No locks are taken on the table, since PgmA is bound with ISO(CS) and CURRENTDATA (NO). As it fetches each row, it performs other SQL or executes other logic to determine if the row needs to be updated. PgmA identifies 10 rows to update, rows 10, 20, 30, and so on. PgmA creates the update statements for each row but saves the update until the end of the unit of work.

Meanwhile, after PgmA has fetched row 20 and before it has updated row 20, PgmB issues an update to row 20. The change PgmB makes is to a column or columns that PgmA will also update, say Col1 and Col2. PgmB also updates the audit column to show that row 20 was updated. PgmB commits. Row 20 no longer contains the columns that PgmA previously read into program storage. PgmB had no difficulty obtaining the page or row lock for row 20, because PgmA held no read locks.

PgmA prepares to issue the update statement. For row 20, which has an audit column based on control number, the update statement is similar to the following SQL, where the key for row 20 is contained in the host variable :HVKEY, and the original value of the audit column is contained in :HVAUDIT.

```
UPDATE TABLEZ SET COL1=:HV1, COL2=:HV2...  
WHERE KEYCOL=:HVKEY AND AUDITCOL=:HVAUDIT
```

At the end of the unit of work, PgmA issues 10 update statements. The update statement for row 20 will fail with SQL+100, and PgmA will retry its application logic for row 20. Then, during the retry, the values PgmB updated will be subject to the application code in PgmA. Thus, PgmB's updates will not be lost.

It is important to note that PgmA must issue the qualified update even if it does not save the updates to the end of the unit of work. Depending on many factors, including the path length of PgmA's code, relative processor speed where the programs execute, and other waits, PgmB could still update row 20 and commit before PgmA determines if row 20 should be updated.

One of these programming approaches to qualifying updates should be part of the standard for developing DB2 application programs. The same concept applies to delete statements. In conjunction with ISO(CS) and CURRENTDATA (NO) these approaches will optimize lock avoidance and ensure the integrity of the data.

3.2.3 Commit frequency

Frequent commits are important to free update locks and maximize concurrency of operations. This is true whenever multiple processes are updating the same tables, whether in a data sharing environment or not. To avoid lock delays between batch jobs and other work, for example CICS or IMS transactions or between the online user-driven transactions themselves, issuing frequent commits is important.

The risk of lock contention is generally low in online user-driven transaction workloads. Each unit of work is generally small and discrete, and accesses specific database rows. For example, a customer account is not usually being updated by more than one user at any one time. DB2 commit points naturally present themselves after each unit of work in an online workload.

Commit points in batch jobs are not as simple. There is trade-off between commit frequency and optimum batch job performance.

The costs and benefits of frequent commits are:

► **Costs**

- Increases CPU overhead and elapsed time
- Closes all open cursors not marked WITH HOLD

► **Benefits**

- Improves the effectiveness of lock avoidance, especially in a data sharing environment
- Reduces the duration of rollback after application failures or explicit rollback requests
- Maximizes concurrency of application execution
- Facilitates execution of utilities, such as Online REORG, that require drains to execute
- Reduces the elapsed time for DB2 system restart following a system failure

Similarly there are costs and benefits associated with taking no commits at all.

► **Costs**

- Extends application rollback time and rerun (minutes, hours, days)
- Increases likelihood of lock collisions with online transaction workload
- Decreases effectiveness of lock avoidance for all application processes
- Prohibits utilities, such as Online REORG, from acquiring drains

► **Benefits**

- Reduces CPU overhead and elapsed time
- Keeps cursors open, which may reduce need to materialize result sets

The cost of commits can be seen in DB2 accounting reports

- For a single phase commit (for example, batch) the cost of commit processing will show up in DB2 Accounting Trace Class 3 counter called Update Commit Wait.
- For a two-phase commit (for example, IMS or CICS), the cost of the first phase of commit will show up in DB2 Accounting Trace Class 3 counter called Active Log Write Wait. The cost of the second phase of the commit will show up in the DB2 Accounting Class 3 counter called Update Commit Wait

The application designer must consider the trade off when designing or modifying batch programs. Generally only very short and predictable jobs should be coded without commits. Remember, any process that holds locks for a long time will reduce lock avoidance for all shared objects in a data sharing environment.

For considerations on managing commit frequency in batch processes, see 3.2.7, “Managing commit frequency in batch” on page 52.

3.2.4 Locking recommendations

The following list includes specific data sharing recommendations and general non-data sharing recommendations.

- ▶ Commit work as soon as is practical: to avoid unnecessary lock contention, issue a COMMIT statement as soon as possible after reaching a point of consistency, even in read-only applications.
- ▶ Avoid using SQL LOCK TABLE as much as possible. SQL LOCK TABLE prevents concurrency with other transactions or programs. It provokes delays and timeouts. Even so you may consider using it in the following cases:
 - Taking a ‘snapshot’: if you want to access an entire table throughout a unit of work as it was at a particular moment, you must lock out concurrent changes. If other processes can access the table, use LOCK TABLE IN SHARE MODE.
 - Avoiding overhead: if you want to update a large part of a table, it can be more efficient to prevent concurrent access than to lock each page as it is updated and unlock it when it is committed. Use LOCK TABLE IN EXCLUSIVE MODE.
 - Preventing timeouts: the application has a high priority and must not risk timeouts from contention with other application processes. Depending on whether the application updates or not, use either LOCK IN EXCLUSIVE MODE or LOCK TABLE IN SHARE MODE.

Use caution when considering a LOCK TABLE statement for accessing simple table spaces. The statement locks all tables in a simple table space, even though you name only one table. No other process can update the table space for the duration of the lock. If the lock is in exclusive mode, no other process can read the table space, unless that process is running with UR isolation.

- ▶ Avoid binding plans and packages with isolation levels Repeatable Read or Read Stability. These isolation levels reduce concurrency. Their purpose is to keep the column values returned to the program unchanged during the execution of the program. In each case DB2 may hold many locks while the program executes.
- ▶ Avoid lock escalation. Lock escalation is the act of releasing a large number of page, row or large object (LOB) locks, held by an application process on a single table or table space, to acquire a table or table space lock, or a set of partition locks. So a *gross* level lock is taken on the DB2 table object which causes applications to serialize access to that object. One effect of lock escalation is to reduce the number of locks the IRLM must manage. It also reduces the number of locks that must be propagated to the CF. The cost of this lock reduction is reduced application concurrency.

The best way to avoid lock escalation is to increase the number of commits. If that is not possible in a given situation, a DBA or system administrator can also change the LOCKMAX setting for the table space (LOCKMAX of zero disables escalation) or the value for the DSNZPARM parameter, NUMLKTS.

When escalation occurs for a partitioned table space, only partitions that are currently locked are escalated. Unlocked partitions remain unlocked. After lock escalation occurs, any unlocked partitions that are subsequently accessed are locked with a gross lock; that is, a partition level lock.

For an application process that is using sysplex query parallelism, the lock count is maintained on a member basis, not globally across the group for the process. Thus,

escalation on a table space or table by one member does not cause escalation on other members.

- ▶ **Close Cursors:** if you define a cursor using the WITH HOLD option, the locks it needs can be held past a commit point. Use the CLOSE CURSOR statement as soon as possible in your program, to release those locks and free the resources they hold. Whether the locks are held is determined by DSNZPARM RELCURHL. See *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418, for more details.
- ▶ **Avoid inter-DB2 Read/Write interest in an object if possible.** This happens when multiple DB2 subsystems intend to read or write pages in the same object at the same time. You can use partitioning techniques to avoid this conflict, so that different processes access different partitions from different DB2 members. This is not possible in the majority of cases for transactions, but you can design batch processes to behave this way. The two purposes of this technique are 1) to avoid the propagation of locks to the lock structure and 2) to reduce group buffer pool activity. We discuss this technique further in the database design topic later in this section.
- ▶ **Issue a ROLLBACK statement after a failure;** it prevents unsuccessful SQL statements (such as PREPARE) from holding locks. Statements issued through SPUFI can be committed immediately by the SPUFI autocommit feature.
- ▶ **Optimistic locking.** This technique minimizes the length of time that a lock is held for update. If the row contains an audit column containing a time stamp or version number value, then when it is read prior to being updated this time stamp or version number can be held in memory. When the update statement for the row is constructed, the row to be updated is qualified by the value in the audit column. If the time stamp or version number comparison is different, the update will return an SQL +100, and the program can retry. In this way the lock for the update is held for a very short time and other work can read the data in parallel without being the victim of a lock contention.
- ▶ **Use caution when SQL statements may affect more than one row.** Cursor statements such as OPEN CURSOR FOR UPDATE and DELETE/UPDATE FOR CURRENT OF CURSOR will affect one row, unless you use cursor set processing. When a table has few rows, a DELETE statement for all the rows can be acceptable. But if you code an SQL statement to handle multiple rows, and the table increases in size over time, you can impact the performance of the entire system, either through lock contention or because other processes, such as DB2 logging, are involved. Row set processing, with multi-row fetch and update statements, and dynamic scrollable cursors may take many more locks in a single SQL statement than other techniques. In each case DB2 may hold multiple page locks during the execution of an SQL statement, since the rows affected by the statement may reside on multiple pages.

3.2.5 DB2 database design

A good clustering and partitioning strategy will balance the requirements for efficient transaction access and short batch elapsed time. Parallel processing against different table partitions can contribute to increased availability and efficient use of CPU capacity. Spreading parallel workload across the members of a DB2 data sharing group can further increase availability, reduce elapsed time, and contribute to overall capacity and throughput improvements. Application and database design should complement each other to take advantage of the benefits of data sharing and Parallel Sysplex while managing the level of traffic to the CF structures.

Index access is an important consideration for database design and application access. A table may be partitioned on one key and clustered on another. For additional index access, non-partitioned secondary indexes (NPSIs) and data partitioned secondary indexes (DPSIs) each allow applications to achieve efficient data access with differing degrees of availability

benefits. NPSIs provide single-index access to a partitioned table space, but can cause contention during insert activity and can increase the time to complete some utility execution. DPSIs have one index space per partition, which reduces contention and reduces time for some utility execution, but some index access will require DB2 to search each partition of the DPSI. The costs and benefits of index access through DPSIs and NPSIs must be considered when choosing the clustering and partitioning strategy.

Clustering

Clustering refers to the sequence in which data rows are stored in DB2 tables. At SQL INSERT time and during REORG utility execution, DB2 attempts to store the new data rows in clustering sequence. Clustering sequence is determined by index definition. DB2 will use an index defined as clustering or, if no index is defined as the clustering index, DB2 will use the first index defined on the table. Clustering can be defined or changed using the CREATE or ALTER INDEX statements.

Very careful consideration must be given to the order in which table data rows are clustered. I/O can be minimized if rows that are commonly accessed together are stored together. Batch processes which perform sequential processing work more efficiently if they can process the data rows in clustering sequence, rather than randomly. Advantages are gained by clustering related tables in the same way, so that processes can move through the related set of tables in a common sequence. By allocating distinct key ranges to different job streams, cloned applications can then process the same tables in parallel without causing contention problems.

Using this approach, the concurrently running jobs will read and update rows associated with different key ranges, taking locks on entirely different DB2 page ranges. For optimum performance of parallel processing, all tables to be updated need to be keyed based on the same sequence. Special attention must be paid to minimizing the need for secondary (non-partitioning) indexes, as heavy insert activity can result in contention, and increased CPU resource consumption and elapsed time.

Clustering a set of table(s) in the sequence required by heavy batch processes achieves many DB2 performance benefits. Data is read sequentially, which:

- ▶ Allows DB2 to read data just before it is required. DB2 achieves this result using:
 - Sequential prefetch, which is determined at bind or prepare time.
 - Dynamic prefetch, which is triggered when skip sequential processing is discovered by the DB2 run time mechanism called sequential detection.

Use of prefetch aims to reduce elapsed time by overlapping CPU and I/O operations. Ideally programs are not delayed by waiting for I/O to complete before they can continue CPU processing.

- ▶ Reduces the need to reread the same page, so processes make only one pass through the data. There are both read and update benefits, as they are performed in sequence through the set of tables.

All tables are clustered in some sequence by default. CLUSTER is a parameter which can be used to explicitly define an index on a table as the clustering index. This is strongly recommended. The data rows are clustered in data pages with respect to the specific index key. The ability to maintain clustering is dependent on distributed free space (PCTFREE and FREEPAGE) defined for the table object and the distribution of new keys to be inserted. Tables should be monitored for data disorganization and poor clustering ratio, and REORGs scheduled to re-instate distributed free space and return the data rows to clustering key sequence. In many situations, but not necessarily all, data will be clustered in primary key sequence. But this may not be best choice for heavy batch sequential processing.

To conclude, the physical database design strategy should be to identify the high-volume tables that are read and updated sequentially by the major batch processes. The key sequence used for heavy sequential processing is often the best choice of clustering key. Keys must be selected for these tables and associated tables that lend themselves to this clustering approach. Subsequently large batch processes can be designed to progress through the tables sequentially. In contrast online transactions tend to make fewer database accesses and their access is generally more random in nature.

If you have more than one type of batch process with different processing order, you should select the clustering sequence to favor the process that has the greatest impact on the critical path of the batch workload. Other criteria include frequency of execution, number of rows accessed, or I/O wait time.

Partitioning

DB2 partitioned table spaces are generally used for tables that contain large volumes of data. They allow the data within a given table to be split into multiple physical data sets. Although the data is physically split in this way, the split is logically transparent to the user or application process that accesses the data; a partitioned table space is accessed via SQL statements in exactly the same way as any other DB2 table.

The optional partitioning index on a partitioned table is also split into different physical data sets, based on the ENDING AT parameter specification defined when the index is created. In this case, there is a one-to-one relationship between the data and index partitions.

Each partition can be clustered, either by the partitioning index or, for table-controlled partitioning, by a non-partitioning index using the CLUSTER keyword.

Advantages of using partitioning include:

- ▶ Processing by key range
- ▶ Reducing or avoiding physical contention
- ▶ Reducing or avoiding logical contention
- ▶ Processing programs in parallel to use available capacity more effectively

The construction of the partitioning database design has a number of major inputs. These include:

- ▶ Identification of the very large tables
- ▶ Batch processes identified as being the time critical
- ▶ Degree of parallelism at the application level to meet target service level agreement (SLA)
- ▶ User transaction usage profiles
- ▶ Expected table insert rates
- ▶ Parallel processing requirements
- ▶ Key design - sequential versus random, and how a key value is assigned

It is often considered that once a table reaches 2 GB in size, it is a candidate for partitioning. With improvements in DB2 performance, demands to store ever-increasing amounts of data and increasing partition independence, this rule of thumb is now perhaps too simplistic.

Partitioning of very large tables is usually a requirement to facilitate more granular utility operations, encourage query parallelism, and facilitate parallelism at the application level for long running batch processing.

Effective clustering and partitioning can provide the following advantages for programs that are designed to execute in parallel:

- ▶ Gives the database a structure that supports cloning of applications across members of a Parallel Sysplex.

- ▶ Allows the workload to be spread evenly to maximize the use of all available CPU capacity on the server, or on multiple servers within a Parallel Sysplex data sharing environment.
- ▶ Reduces I/O contention, which reduces elapsed time and can reduce timeouts. Effective clustering and application design will reduce cases where tables are accessed in different orders by different processes, which could otherwise cause deadlocks.
- ▶ Partitions may be allocated to different devices. The placement of data sets can be critical when running parallel jobs. If each job is running on one or more partitions placed on separate DASD volumes, and accessed by separate paths, I/O contention is reduced. The latest DASD technology includes large caches for data and parallel access paths to volumes, however partitioning and separation of datasets on different volumes continue to be useful in reducing I/O contention.
- ▶ Utilities can run against a single partition (as opposed to the whole table space). Running the utility as several jobs, each for one partition, in parallel with each other may reduce the overall elapsed time for the total task. Some DB2 utilities now provide for parallelism within a single job step based on data partitioning, for example COPY and RECOVER. This should reduce elapsed time for utility processing. LOAD can operate on multiple partitions in one job step and avoid non-partitioned index (NPI or NPSI) contention.
- ▶ Scheduling flexibility can be achieved by a utility processing one partition at a time. The other partitions can then be processed by the utility later. For example, reorganizing a table one partition at a time each night may meet the requirement instead of reorganizing the whole table at a weekend. Recovery time for a single partition will be a lot smaller than for recovering a complete table.

A clustering and partitioning scheme that addresses the major batch requirements will probably be acceptable for most online transaction processes. Single row access via an index should perform well. However, some transaction and query processes may access multiple rows in an order not supported by the clustering scheme or located in different partitions. In these cases, some design trade-offs may have to be made. For example, certain online “browse list”-style processes that use non-partitioning indexes (NPIs or NPSIs) may experience contention.

Sequential key

A common requirement for a new table is to have a sequential key in it. If you maintain one record with the last key used you are creating a hot-spot on that record. There are better ways of achieving this goal:

- ▶ Use an identity column. This provides a way to have DB2 automatically generate unique, sequential and recoverable values for specific columns in a table. You must define the column with the AS IDENTITY attribute. Each table can have one identity column.
Identity columns are ideally suited for the task of generating unique primary key values. Applications can use identity columns to avoid the concurrency and performance problems that can result when an application generates its own unique counter outside the database.
Using identity columns you can avoid:
 - Inter-system physical contention for the page containing the counter (page physical lock (page P-lock) negotiation plus buffer invalidate and refresh).
 - Loss of access to the shared counter in case one DB2 member fails. The surviving member would not be able to use the counter as a retained lock would be held on the counter.
- ▶ Use a sequence object. Sequence objects are similar to identity columns except that they are not related to a specific table. The same sequence object can be used for rows in more than one table.

See “Application considerations for DB2 system administrators” on page 56 for more information about these subjects. For a detailed discussion of sequence objects and identity columns, refer to the IBM Redbook *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know,... and More*, SG24-6079.

Note that both identity columns and sequence objects give best performance in a data sharing environment if the CACHE option is used to cache a number of values for the identity or sequence. However, caching in a data sharing environment will not guarantee sequential insertion of values. A program executing on one DB2 data sharing member, having cached values 1 through 20, may insert row number 4 right after another program, executing on another member and having cached values 21 through 40, has inserted row number 22. If your application must produce numbers in strict sequence regardless of how many application copies may be running in the Parallel Sysplex, then you should specify CACHE NO.

- If you do not use either identity column or sequence object, which we recommend, you can still maintain a unique sequential key. You could define a descending index on the key column, for example, if it is a character column. Then you can use the MAX function to retrieve the last key used, and calculate the new key value.

It is not necessary to create a descending index to support a MAX function where an ascending index already exists. It may be possible to drop some indexes that were created explicitly for the purpose of supporting the MAX (or MIN) function.

Consider rebinding packages containing SELECT MAX(col) statements if there is an ascending index on column or SELECT MIN(col) statements if there is a descending index on column. They may benefit from the improved access path. Review whether any indexes created to support MAX or MIN processing can be dropped.

You can find more information on this item in IBM Redbook *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129.

3.2.6 Batch considerations

Parallel application processing can exploit the full power of Parallel Sysplex and data sharing, and reduce the elapsed time of batch processing. Effective parallel application processing requires multiple job streams with little or no logical contention. In the ideal situation, a batch job stream can be designed to execute on one server and only process certain partitions of a related set of tables. Other job streams would execute on other servers against other partitions. This would minimize contention and would also minimize traffic to CF structures for global locking and Group Buffer Pool (GBP) access.

In the real world, there are generally non-partitioned indexes or the job stream must access tables that are either not partitioned or are partitioned on a different key. If batch jobs executing on different servers must all access the same index space, table or partition, the jobs will incur the additional cost of sending lock and GBP traffic to the CF. Application design and database design coupled with job scheduling can minimize CF traffic and its associated costs. However, minimizing CF traffic means actively managing the workload. To a certain extent, this is contrary to the goal of autonomic computing, in which the system directs where work is executed to achieve the best overall results consistent with available capacity and service goals.

Parallel Sysplex and DB2 data sharing performance on current hardware technology is very good, so there should not be a concern if existing batch job streams cause lock and GBP traffic. It is particularly difficult to avoid CF traffic if batch jobs are executing on multiple servers and online transactions are accessing the same tables.

When partitioning, be sure you have considered the locking advantage if you process one partition only from one single DB2. That technique improves performance as the locks are not propagated to the CF but are resolved locally. This applies more in a batch window situation. Often online transactions are designed to run in any server in the Parallel Sysplex, and it may be counter-productive to attempt to limit transactions that access a specific partition to a particular server.

Whether in a non-data sharing or data sharing environment, batch jobs that process rows of large tables sequentially can be split into several clones. For example, one job clone handles A-F, the second handles G-K, and so on. All of these clones can then be run in parallel to reduce the overall elapsed time of the complete process. This can help you to shorten the batch windows for some applications.

You can shorten and even eliminate your batch window if you consider processing your batch with new transactions instead of a job process. This can be more costly in terms of processing and may require different programming techniques to handle exception conditions.

Consider using WebSphere MQ as a means to send work from online transactions to batch transactions. Each message you write in an MQ queue can start a transaction. There are other examples in Chapter 7, “WebSphere MQ application considerations” on page 131.

You can also consider an intermediate solution, where you create transactions to process a big part of what is currently in the batch stream, and retain a much shorter batch job to handle a specific situation. For example, the logical end-of-day may be handled best by a batch process, while transactions would be able to keep up with mid-day activities.

Consider using a *soft fail* exit. When a severe programming error is detected in an environment with concurrent batch and online processing, it becomes very difficult to retain a *restore and rerun* policy for the jobs in the batch stream. It may be preferable to “soft fail” the process in error and let all other processes carry on. This would maximize the availability of the program components that are processing correctly. Then fix the program error, fix the data in error, and finally restart the corrected function.

As discussed, frequent commits should allow online REORG to break in and take the necessary drains it needs. Use of cursors WITH HOLD in a program will cause a conflict between the program and online REORG, because the WITH HOLD cursor will retain a read claim. Avoiding cursors WITH HOLD will enable online REORG to break in when the next intermediate checkpoint occurs. However, this may affect the performance of the batch process as the SQL cursor has to be re-positioned after each commit. From a business point of view it may be more appropriate to avoid scheduling online REORGs when key business processes execute.

3.2.7 Managing commit frequency in batch

Our recommendation is to design batch programs, or any long-running application process, with externally controlled commit points. This requires checkpoint/restart logic to be included early in the application design process. External control can be managed by a DB2 table or tables, in which commit frequency and restart information are included for each job. The batch job then consults the values in the table to determine how often to issue commits. Before taking a commit, the batch job writes progress and restart information to the table. This allows operations staff to modify batch behavior by reducing commit frequency when only batch jobs are running and increasing commit frequency when online or internet transactions are prevalent.

In the example billing application illustrated in 3.4, “Example application” on page 62, commit frequency is another data element in the batch job control table. It was specified in seconds. A value of zero means commit after every complete unit of work. A value of 2 means commit every two seconds once the unit of work in progress has completed. This potentially enables many more units of work to be completed before a commit is taken and reduces the elapsed time of the whole batch process.

There are benefits if the commit frequency can be changed dynamically. During the overnight schedule, when very few users are using the system, a value of two seconds may be good as there will be very little contention with user transactions. However, if the batch schedule is not complete when the online day begins, or if batch jobs can run at any time, then a value of zero may be preferred to reduce the impact on user transactions. If the commit frequency can be changed dynamically while the job is running then more scheduling flexibility is possible. This flexibility may be of further importance if the application is available to the World Wide Web at all hours of the day.

The nature of the work being done between commit points also needs consideration. For example if redundant rows are being deleted from a table, a commit after each delete would cause the delete job to have a longer elapsed time than if a commit was taken after, say, every 500 deletes or the number that could be deleted within a second. However the more deletes that are performed between commit points, the greater the risk that DB2 lock escalation will occur. This may have a considerable impact on the whole system. If long elapsed time for a delete job is not a concern from a business point of view, then frequent commits should be coded.

Another point to be considered is that your DB2 processes may access IMS databases. In that case you must be aware of the recommendations you can find in the section, Solutions to Locking Problems, in the IMS chapter.

3.2.8 Distributed access to a data sharing group

Programs that access DB2 data from distributed clients should continue to execute successfully if the DB2 subsystem that contains the data is part of a data sharing group. Existing distributed programs should be reviewed and new distributed programs designed to optimize lock avoidance and take frequent commits. In addition to the locking and commit considerations, changes to programs or configuration settings may be required to maximize the benefits of workload balancing, high availability, and expanded capacity that Parallel Sysplex and DB2 data sharing offer.

The following paragraphs offer an overview of these ideas, in which DB2A, DB2B and so on are members of a data sharing group. Refer to IBM Redbook *Distributed Functions of DB2 for z/OS and OS/390*, SG24-6952, for details on how to implement distributed access to DB2 data. Also, *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417 devotes Chapter 4 to discussing connectivity to the members of a data sharing group.

Workload balancing

Distributed programs specify a LOCATION name to indicate the DB2 application server that contains the tables they wish to access. Configuration settings in another DB2 for z/OS or in DB2 Connect™, for example, indicate the TCP/IP host address or the SNA LU that corresponds to that LOCATION name. When the DB2 application server is part of a data sharing group, the ability of the distributed program to access more than one member of the group depends on how the data sharing group is defined during data sharing implementation. Generally, the LOCATION of the originating DB2 member remains the LOCATION of the data

sharing group. If this is true in your environment, the distributed program will not have to change.

The original TCP/IP host name and SNA LU may or may not apply to the entire data sharing group. If not, configuration settings must be updated to allow the program access to more than one member. For example, if the program happens to connect to DB2A on its first execution, but DB2A is running close to capacity, it is desirable for the next execution of the program to connect to DB2B. Both TCP/IP and SNA networks provide support for workload balancing.

LOCATION name and LOCATION ALIAS

Some customers want the advantages of workload balancing without allowing distributed work to flood all the members of a data sharing group. For example, in a six-member data sharing group, a customer may have a set of distributed programs that have unpredictable requests that can consume a lot of resources during peak transaction times. It may be desirable to limit distributed access from such sources to four members of the data sharing group, ensuring two members were always able to handle high-priority work. This can be done using a LOCATION ALIAS. The LOCATION specified in the distributed programs would be an alias to a LOCATION that only applied to four members of the data sharing group. LOCATION ALIAS techniques can be applied to SNA or TCP/IP traffic.

Availability

A distributed application process that normally connects to DB2A will need to connect to another member of the data sharing group if DB2A is brought down for maintenance. If the program is executing queries and not updating, then it can remain available as long as it can access another member of the group as described in the workload balancing comments above.

If the distributed program is changing DB2 for z/OS data and experiences an outage in the midst of a unit of work that involves two phase commit processing, the program must reconnect to the same DB2 member to complete the unit of work, either through commit or rollback.

Capacity

Generally, most distributed application processes should be able to execute on most or all members of the data sharing group. Sometimes, the relative capacity available on each member of the data sharing group will vary widely. In such cases, it is desirable to direct work based on available capacity.

DB2 Connect servers or DB2 for z/OS application requesters can direct work to the members of the application server data sharing group based on WLM information about capacity. The application requester keeps a list of the members of the data sharing group. When the applicator requester connects to a data sharing group, WLM will return capacity information about the members of the group. The application requester can match the entries in the list with the capacity information WLM returns to decide where to send the next request. This capability should be transparent to the application.

3.2.9 Programming guidelines

Here are some general programming considerations that apply to both data sharing and non-data sharing environments:

- Access data in a consistent order. When different applications access the same data, try to make them do so in the same sequence. For example, make both applications access rows 1,2,3, and 5 in that order. In that case, the first application to access the data delays

the second, but the two applications cannot deadlock. For the same reason, try to make different applications access the same tables in the same order.

- ▶ Retry an application after deadlock or time-out. Include logic in a batch program so that it retries an operation after a deadlock or time-out. That could help you recover from the situation without assistance from operations personnel. Field SQLERRD(3) in the SQLCA returns a reason code that indicates whether a deadlock or time-out occurred.
- ▶ Include error handling in batch programs. For example, do not stop the entire batch process because one transaction within it has failed. Design should allow for the reporting of anomalous processing and the ability of the batch job to continue to complete as many transactions as possible.
- ▶ Include provisions to report any negative SQL return code. Make sure your program does not only handle and report expected return codes, but is able to report unexpected return codes, too. For new applications, use GET DIAGNOSTICS, and consider changing existing applications to use GET DIAGNOSTICS instead of the SQLCA.
- ▶ Use GET DIAGNOSTICS in conjunction with ATOMIC and NOT ATOMIC for multi-row cursors (multi-row FETCH, INSERT and UPDATE). Applications should have retry logic based on whether the individuals rows can each succeed or fail (NOT ATOMIC) or the rows all succeed or fail together (ATOMIC).
- ▶ Avoid creating cursors with complex DB2 queries that intend to solve any case that could happen in a transaction. Consider creating a separate cursor for each situation. That way, you execute the query that applies to a specific situation. You may improve performance, your program increases in clarity, and further modifications can be made more easily, with fewer errors.

3.2.10 Additional DB2 considerations

Although these considerations are not specific to Parallel Sysplex, they can help you improve how your application uses DB2. Since each customer has unique requirements, some of the considerations may not apply to your installation.

- ▶ Use standard utilities for initial loads. Standard utilities have good performance with initial massive loads. Avoid creating nonstandard programs for that purpose. Compared with utilities, they can increase cost because of extended use of DB2 resources such as log updates and locks.
- ▶ Run RUNSTATS and REBIND after massive initial loads. If the application programs have been deployed into production before you have executed the initial load, it is strongly recommended you execute a RUNSTATS to actualize your table statistics in the catalog, and REBIND to be sure you are using the proper access paths. Otherwise your process might keep accessing a table through table space scan instead of using an index, because when the statement was bound the table was empty or had few rows.
- ▶ Plan to run Runstats and Rebind on a regular basis for tables that do not have initial data loads or that have high data increases. Consider defining a table as VOLATILE if the number of rows changes dramatically on a frequent or periodic basis. CREATE or ALTER TABLE with VOLATILE keyword tells DB2 to favor index access. This is useful for cases where the number of rows at any given point in time is unpredictable and where RUNSTATS is difficult to schedule or may not capture a representative number of rows.
- ▶ Plan to reorganize tables regularly if they have a high degree of inserts throughout the table. Tables that are very disorganized tend to decrease application performance.
- ▶ Try to use an index to access your data, except for very small tables with just a few rows or for sequential processes.

- ▶ Define indexes that can be used to avoid sorts for SQL statements that use ORDER BY. This will help avoid internal sorts.
- ▶ Create multiple indexes for different types of access with care. Creation of additional indexes is optimal whenever the columns of these indexes are not updated frequently and where rows are not inserted often. When you define a new index you increase the cost of subsequent inserts, updates and deletes.
- ▶ Consider using a unique index for queries that create lists. This may facilitate the ability of the program to reposition as the user pages forward and backward. Scrollable cursors may be useful in this case, too.
- ▶ Consider index only access when designing non-unique indexes on tables. For some transactions and queries you can avoid the access to the data pages. If you have to access the data you are at least doubling the cost.
- ▶ Denormalize the data design in specific instances to avoid joins and enhance performance. Denormalization should be the exception, not the rule, since it makes application design and coding more complex.
- ▶ Include SCOPE(GROUP) as appropriate when issuing START, STOP and DISPLAY commands for DB2 functions and procedures. The functions and procedures may be active on multiple members of a DB2 data sharing group. Possible errors could occur.
 - Assume a data sharing group containing two members: DB2A and DB2B.
 - Assume SP is a stored procedure with STAY RESIDENT YES executing in both members.
 - Assume a program modification is performed on SP.

If, in DB2A, the operator issues -STOP (SCHEMA.SP) and then -START (SCHEMA.SP) to refresh the stored procedure, only the DB2A resident copy will be refreshed since SCOPE(LOCAL) is the default. DB2B's resident copy will still be the old version. This may not be desirable. Instead the -STOP (SCHEMA.SP) SCOPE(GROUP) and -START (SCHEMA.SP) SCOPE(GROUP) format should be used.

You can find more information in the following publications:

- ▶ *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417
- ▶ *DB2 UDB for z/OS Version 8 Application Programming and SQL Guide*, SC18-7415
- ▶ *DB2 UDB for z/OS Version 8 SQL Reference*, SC18-7426
- ▶ *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413
- ▶ *DB2 for z/OS Application Programming Topics*, SG24-6300

If your company has designed its own tools to generate the SQL statements, you should include support for this functionality.

3.3 Application considerations for DB2 system administrators

This section is intended for the person who performs the SYSADM functions. These functions include those related to creating and operating DB2 subsystems such as:

- ▶ Selecting the DSNZPARM values
- ▶ Selecting the DSNHDECP values
- ▶ Managing the logs and BSDS
- ▶ Managing the buffer pools, edm pool, rid pool, and sort pool
- ▶ Managing the work file database
- ▶ Backing up and recovery of the DB2 Catalog and Directory and critical user data sets
- ▶ Defining default standards for BIND and locking modes

3.3.1 BIND options

Minimizing the number of global lock requests propagated to the Coupling Facility will reduce the host CPU cost of active data sharing and reduce CF resource consumption. There are multiple ways of minimizing lock requests.

Use of RELEASE(DEALLOCATE) to reduce XES messaging for page set L-locks is not as important in V8 as in previous releases. However, it is still recommended for long-lived threads like batch applications and protected CICS threads.

Bind with ISOLATION(CS) and CURRENTDATA(NO) to minimize the amount of global lock propagation to the Coupling Facility and to reduce the possibility of global lock contention. This combination allows DB2 to attempt to avoid taking the lock when it can determine that the data on that page is committed. A latch will be taken instead of a lock and a possible lock propagation to the lock structure is avoided. ISOLATION(CS) also lets DB2 release acquired locks as soon as possible.

ISOLATION(UR) is also recommended in situations where programs can tolerate the very latest version of the page which have updated rows that may not have yet been committed to the database. There is the risk that the application unit of work could rollback. Use of UR avoids the acquisition of locks for read requests only.

Other combinations include:

- ▶ ISOLATION(CS) with CURRENTDATA(YES), when data you have accessed must not be changed before your next FETCH operation
- ▶ ISOLATION(RS), when rows you have accessed must not be changed before the application commits or rolls back. However, you do not care if other application processes insert additional rows.
- ▶ ISOLATION(RR), when rows you have accessed must not be changed before the application commits or rolls back. New rows cannot be inserted into the answer set.

Avoid widespread the use of the bind parameter IMMEDIATEWRITE(YES). This option ensures that updated pages are externalized to the CF and are visible ahead of when application commit point is reached. The DB2 recovery log must be forced to DASD ahead of externalizing an updated page to CF. This is very expensive. This support was introduced by DB2 to support a very specific case where bad application design became exposed when the applications migrated to Sysplex and active data sharing. For example, a primary transaction running on System A spawns a secondary transaction ahead of commit of the primary transaction. The secondary transaction is scheduled on a different member and does not see the updates which have not yet been committed. In a non-sharing environment, the spawned transaction would be serialized behind the primary transaction and would have to wait for the secondary transaction to commit the updates. With use of BIND parameter IMMEDIATEWRITE(YES), similar behavior will occur.

IMMEDIATEWRITE(NO) causes updated pages to be externalized to the CF during phase 1 of the two-phase commit. IMMEDIATEWRITE(PH1) is available as a bind option in V8 for compatibility reasons only, while IMMEDIATEWRI(PH1) was totally removed from DSNZPARM. There is no performance penalty even in application rollback; that is, there are no additional forced writes to the DB2 recovery log. This option ensures that transactions correctly spawned at application commit time will always see the very latest committed version of data.

Bind Plans with ACQUIRE(USE) which is best for concurrency. Packages are always bound with ACQUIRE(USE), by default. ACQUIRE(ALLOCATE) gives better protection against deadlocks for a high-priority job.

Reduce the scope of BIND operations by using packages. This reduces DB2 catalog and directory contention. Advantages also include:

- ▶ Ease of maintenance
- ▶ Incremental development of programs
- ▶ Versioning
- ▶ Flexibility of using BIND options
- ▶ Flexibility in using name qualifiers

3.3.2 DDL options

LOCKSIZE

Use LOCKSIZE ANY or if needed use LOCKSIZE PAGE instead of LOCKSIZE ROW. LOCKSIZE ANY allows DB2 to choose the lock size, and DB2 usually chooses LOCKSIZE PAGE for non LOB objects and LOCKSIZE LOB for LOBs.

Physical locks (P-locks) must be obtained on a page to preserve physical consistency of the data between members. Page P-locks are used, for example, when two subsystems attempt to update the same page of data and row locking is in effect. Physical locks are used only in data sharing environments.

Because of the possible increase in P-lock activity with row locking, evaluate carefully before using row locking in a data sharing environment. If you have an update-intensive application process, the amount of page p-lock activity might increase the overhead of data sharing.

MAXROWS

To decrease the possible contention on those page P-locks for smaller table spaces, consider using page locking and a MAXROWS value of 1 on the table space to simulate row locking. You can get the benefits of row locking without the data page P-lock contention that comes with it. A new MAXROWS value does not take effect until you run REORG on the table space.

LOCKMAX

LOCKMAX specifies the maximum number of page, row, or large object (LOB) locks an application process can hold simultaneously in the table space. If a program requests more than that number, locks are escalated.

DB2 statistics and performance traces can tell you how often lock escalation has occurred and whether it has caused timeouts or deadlocks. If deemed necessary, you might ALTER the table to increase LOCKMAX and thus decrease the number of escalations.

Alternatively, if lock escalation is a problem, use LOCKMAX 0 to disable lock escalation. However, acquiring too many locks can cause DB2 to fail if IRLM runs out of storage for the locks. If you use LOCKSIZE ANY LOCKMAX 0 to disable lock escalation, DB2 might acquire an X lock on the table space instead of ANY, PAGE or ROW locks. To avoid the table space lock in these cases, ALTER the table space to increase LOCKMAX to a large value.

Example: Assume that a table space is used by transactions that require high concurrency and that a batch job updates almost every page in the table space.

For high concurrency, you should create the table space with LOCKSIZE PAGE and make the batch job commit every few seconds.

LOCKSIZE ANY is a possible choice, if you take other steps to avoid lock escalation. If you use LOCKSIZE ANY, specify a LOCKMAX value large enough so that locks held by

transactions are not normally escalated. Also, DSNZPARM (NUMLKUS) LOCKS PER USER must be large enough so that transactions do not reach that limit.

- ▶ If a batch job is concurrent with transactions, then it must use page or row locks and commit frequently, perhaps every 100 updates. Review DSNZPARM NUMLKUS, LOCKS PER USER to avoid exceeding the limit. The page or row locking uses significant processing time. Binding with ISOLATION(CS) may discourage lock escalation to an X table space lock for those applications that read a lot and update occasionally. However, this may not prevent lock escalation for those applications that are update-intensive.
- ▶ If a batch job is non-concurrent with transactions, then it does not need to use page or row locks. The application could explicitly lock the table in exclusive mode.

GRPCACHE

Review the GBPCACHE option when you create or alter table spaces or indexes to define what data, if any, should be cached in the group buffer pool. The options are NONE, SYSTEM, CHANGED, and ALL. The default is CHANGED and should be used except for unusual circumstances.

TRACKMOD

Consider TRACKMOD NO when there are frequent updates from multiple members.

This option can degrade incremental image copy performance. If you rarely or never use incremental copies, or if you use DFSMS™ concurrent copies and LOGONLY recovery, then choosing TRACKMOD NO can help transaction performance.

MEMBER CLUSTER

Consider the MEMBER CLUSTER option when there are many inserts to the same table from multiple members.

CREATE SEQUENCE

Sequences provide unique, sequential, numeric values for DB2 applications. Applications can use sequence numbers for a variety of purposes, including the avoidance of concurrency and performance problems that can result when applications generate their own sequence numbers.

The CACHE option of the CREATE SEQUENCE statement directs DB2 to preallocate a specified number of sequential values in memory. This performance and tuning option provides faster access to the sequence by eliminating synchronous I/O to the SYSIBM.SYSSEQUENCES table each time an application requests a new sequence number.

Strict sequential order cannot be guaranteed in a data sharing environment with sequence numbers being cached on multiple members. In data sharing, each member has its own cache. For example, if you have 2 data sharing members, DB2A and DB2B, and sequence SEQ1 was created starting with 1 and incremented by 1, DB2A may have a cached set of 1-20 and DB2B 21-40. If a request first came to DB2A for a sequence, a value of 1 is provided to the application, then a request to DB2B, a value of 21 is provided, back to DB2A, a value of 2 is provided, back to DB2B, a value of 22 is provided. The end result would be values of 1,21,2,22 and so on. For data sharing environments, if sequence numbers must be provided in strict numeric order, use the NO CACHE option of the CREATE SEQUENCE statement.

3.3.3 DB2 installation options

In this section, we briefly review the installation options.

NUMLKTS and NUMLKUS

Avoid lock escalation. Review the DSNZPARM NUMLKTS, LOCKS PER TABLE parameter if a general lock escalation problem appears in your installation. NUMLKTS specifies the number of page, row, or LOB locks that a single application can hold simultaneously in a single table or table space before lock escalation occurs. Also review the DSNZPARM NUMLKUS, LOCKS PER USER parameter which specifies the number of page, row, or LOB locks that a single application can hold simultaneously in a single table or table space before lock escalation occurs.

IRLM 2.2 can manage up to 100 million locks, so that DSNZPARM parameters NUMLKTS and NUMLKUS now allow up to 100 million locks each. Make sure there is enough real storage to support back it up.

URCHKTH and URLGWTH

Use the DSNZPARM URCHKTH, UR CHECK FREQ field of the installation panel DSNTIPN to help you identify those applications that are not committing frequently. The setting of UR CHECK FREQ should conform to your installation standards for applications taking commit points. Along with URCHKTH, use DSNZPARM (URLGWTH) UR LOG WRITE CHECK which will produce a message when a UR produces more log records than what is specified for URLGWTH without committing. Using URLGWTH is more granular than URCHKTH.

LRDRTHLD

The DSNZPARM LRDRTHLD, LONG RUNNING READER THRESHOLD records the time that a task holds a read claim. For parallel operations, DB2 records this information only for the originating task. When a task holds a claim longer than the number of minutes specified in LRDRTHLD, and for each multiple of minutes, DB2 writes an IFCID 313 record which contains the total number of minutes that a claim was held. However, no console message is displayed.

Sysplex parallelism

DB2 sub-tasking can be employed in a sysplex environment with active data sharing to achieve multi-CEC query parallelism.

Queries become eligible for a possible parallel query plan depending on the DEGREE(ANY) parameter on application BIND for static SQL and on the DEGREE(ANY) keyword on SQL cursor for dynamic SQL. The full set of rules is documented in *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413.

When sysplex query parallelism is invoked for a long-running CPU and/or I/O-bound query, multiple sub-tasks are spawned and distributed around the members of the data sharing group. The DB2 member in which the query originated acts as a *coordinator* and the other DB2 members are involved act as *participants*. Individual members can be included or excluded based on ZPARM settings or buffer pool settings. Make sure there are sufficient resources including virtual storage, I/O controller bandwidth, and CPU processing power to be able to support and absorb the concurrent sequential I/O streams on every member that is participating.

PARAMDEG in DSNZPARM can be used to “cap” the maximum of parallelism for an individual query. It is prudent to set this to a reasonable value as a conservative or defensive measure, for example 2-4x number of CPs. It is a compromise decision between very best elapsed time performance for an individual query versus over-commitment of resources.

EDMPOOL

For Data Sharing, you may want to increase the size of the EDM pool - DSNZPARM EDMPOOL, EDMPOOL Storage size by 10% because of the way DB2 cross-invalidates DBDs (Database Descriptors). This percentage is just an estimate; the actual amount of the increase depends on how often you CREATE, DROP, and ALTER objects in the data sharing group. For CREATE, DROP, or ALTER statements, the DBD is not modified until a COMMIT is issued. You can significantly reduce the number of EDM versions by issuing CREATE, DROP or ALTER within a single COMMIT scope. However, the exclusive lock on the DBD is held until the COMMIT. You can find more information on this topic in *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417.

Pseudo-close

When in Data Sharing, DSNZPARM parameters PCLOSEN, RO SWITCH CHKPTS and PCLOSET, RO SWITCH TIME add a level of complexity. When an open data set that is not updated (INSERT, UPDATE, or DELETE) for either PCLOSEN number of system checkpoints or PCLOSET number of minutes, the data set is pseudo-closed. This causes the data set to be converted from R/W to R/O which can result in the data set losing inter-DB2 read-write interest and no longer being GBP-dependent.

There is considerable overhead when a data set either becomes GBP-dependent or drops out of GBP-dependency. The concern is that a data set could be pseudo-closed and, very soon afterward, be updated and become GBP-dependent again.

PCLOSEN defaults to 5 checkpoints and PCLOSET to 10 minutes. You may choose to raise either or both of these values to cause data sets to avoid being pseudo-closed to minimize the possibility of switching back and forth between being GBP-dependent and non GBP-dependent.

Many shops have changed DSNZPARM CHKFREQ, CHECKPOINT FREQ from a large number of log records to minutes, typically between 5 and 10 minutes. If this is the case, both PCLOSEN and PCLOSET are actually using minutes to cause pseudo-closing. The difference is that PCLOSEN has a direct relationship to CHKFREQ, while PCLOSET is an absolute value.

Let's assume that PCLOSEN is set to 5, PCLOSET is set to 20, and CHKFREQ is set to 5. A data set is converted from R/W to R/O if it has not been updated in the past 20 minutes (PCLOSET), or 25 minutes (PCLOSEN=CHKFREQ*5). In this example, you can see that PCLOSEN will never be used unless, of course, CHKFREQ is changed to a lower value.

You may want to consider setting PCLOSEN to a high value to eliminate it as a determining factor and set PCLOSET to a value between 20 to 30 minutes.

Both DB2PE and DB2PM tools have the ability to generate the DB2 Statistics Long report. This report has a section called the OPEN / CLOSE Activity. One of the values in this section labeled DSETS CONVERTED R/W -> is an indicator of pseudo-close activity. A general rule is to keep the number of data sets indicated by this value to be less than 10 per minute.

3.3.4 Deadlock and time-out detection

Quick detection of deadlocks and timeouts is necessary in a data sharing environment to prevent a large number of waiters on each system. These large numbers of waiters can cause much longer wait times for timeouts and deadlocks. Based on this assumption, here are two recommendations:

- ▶ If your DB2 non-data-sharing subsystem has a problem with deadlocks, consider reducing the deadlock time to prevent these long lists of waiters from developing. (If you don't have

a problem with deadlocks, it is likely that you won't have to change any parameters for data sharing.)

- ▶ If you have stringent time-out limits that must be honored by DB2, consider decreasing the deadlock time before moving to data sharing.

You can find more information on this topic in *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417.

3.3.5 Backup and recovery

High availability is achieved in a Parallel Sysplex by configuring with no single points of failure. However, there is only one copy of all the shared data including the DB2 catalog and directory. A loss of a critical object, especially one that is part of the catalog or directory, can cause an outage of the entire data sharing group. Therefore, it is very important to be able to recover any of these objects as quickly as possible.

Note: For the shortest recovery duration, it is necessary to have *all* required logs from *all* members in the data sharing group to be available on DASD.

At a minimum, take full daily backups of the DB2 Catalog and Directory and all critical data sets. Consider taking backups on a more frequent basis to facilitate a quick recovery in the event of a failure or corruption.

The following actions are available for you to take to make sure recovery runs quickly:

- ▶ Take more frequent image copies.
- ▶ Allocate larger active log data sets.
- ▶ Allocate more active log data sets.
- ▶ Write archive logs to DASD.
- ▶ Evaluate taking image copies of large indexes.

Make sure that the DSNZPARM LOGAPSTG, LOG APPLY STORAGE is set to 100 MB so that DB2 fast log apply is enabled.

3.4 Example application

This example application illustrates how clustering and partitioning can be designed to ensure maximum parallelism is utilized in batch processing. This example is loosely based on a real billing system used in a UK utility company.

In this example application, there are two main tables whose keys determine the way in which many other tables are clustered. These tables are Customer Account and Service Account, where customer accounts can have multiple service accounts. The processing strategy is designed to cater for situations where processing of Service Accounts (or tables sequenced by the service account key) proceeds in Customer Account order. We will examine the main batch billing process that lends itself to processing in Service Account order.

Customer account key

To ensure an even spread of Customer Accounts across the partitions, the keys are randomly generated. There is a one-to-many relationship between Customer Account and Service Account.

Service account key

When creating a new Service Account row, the left-hand portion or most significant part of the Service Account key will mirror the key of the owning Customer Account. The least significant part of the key or the right-hand portion will be allocated sequentially within a given Customer Account. A new external identifier will be allocated for a Service Account (the reference used by the customer) and will be distinct from the internal key. The external identifier will be held on the Service Account table as an indexed column to facilitate application lookup via CICS transactions and so on.

The diagram in Figure 3-4 illustrates the table design.

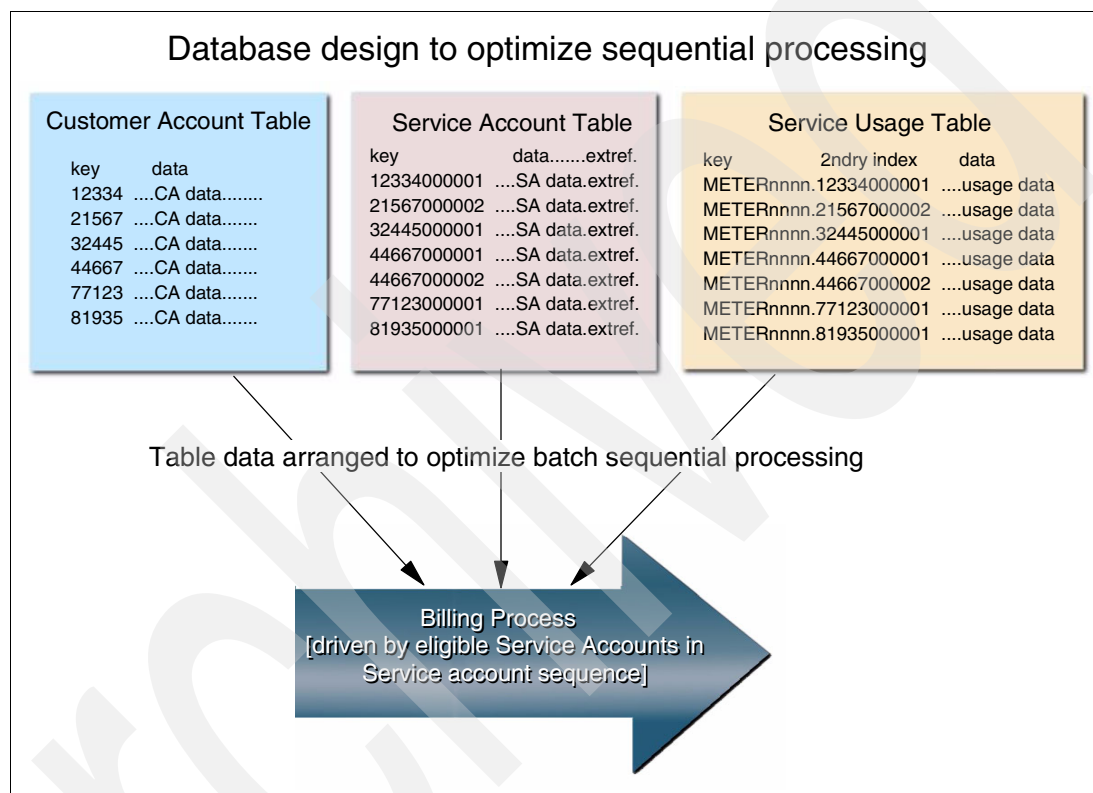


Figure 3-4 Database design for sequential processing

3.4.1 Batch processing

The partitioning and clustering strategy has a major effect on the design of the batch process functions and vice versa. The table design has been structured to allow batch processes to run in parallel to minimize elapsed time and to take advantage of all available CPU resources.

The predominant clustering sequence of the billing process is based on the customer account and service account keys. The sequence in which they will be processed is determined by driving related SQL cursors in parallel. The driving cursors are designed to order the data in Customer Account and Service Account sequence. Other related tables are also processed in this sequence. The service usage table has a different primary key but can be processed in service account order by use of a secondary index also based on service account.

3.4.2 Parallelizing sequential processes

In order for the billing process to be made clonable another table, the batch job control table, has been introduced in the diagram shown in Figure 3-5. This batch control table specifies the driving key range that each clone will process, for example, Clone 3 processes service accounts within the key range of 40000 to 59999. The database design isolates the processing within this clone from any other clone. The batch job control table will be accessed and updated by each clone, but each row of this table will be on a separate DB2 page to avoid contention.

With the large sequential billing process split between five clones, provided each service account processes in approximately the same time and provided the service account keys are generated randomly we can expect each clone to have approximately a fifth of the elapsed time of the total billing process.

Using the batch job control table in this way means that the number of clones in use can be changed very simply by adding or removing rows. This enables scalability.

The driving cursor is designed with min/max key predicates where the values for the min/max keys are read in from the batch job control table. This mechanism controls which range of keys each clone will process.

The application this example is based on, also used this table for holding restart information, the last service account processed and committed is held in another column in the batch job control table. In addition this table was used to hold commit frequency information (see discussion below on DB2 locking strategies).

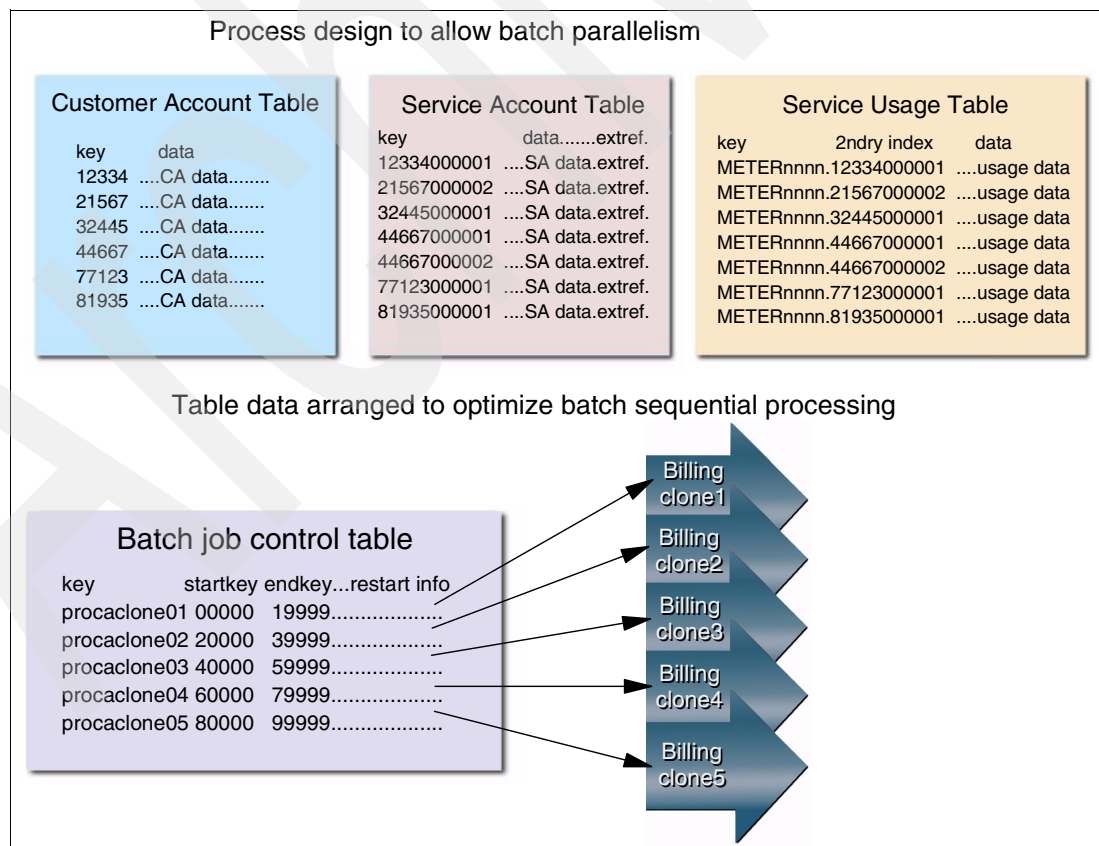


Figure 3-5 Batch parallelism

Note: For most applications in the real world, since it will not be possible to process all tables in clustering sequence, the design should make compromises to ensure that the largest volumes processing or most critical process is based on clustering sequence.

3.4.3 Extraordinary processes

The example application referred to is simplistic. In the real world there are bound to be “out of the ordinary” business processes that have to be accommodated. A compromise made in the design of the application in the example is based on having information built into the Service Account key where the significant portion of it reflects the owning Customer Account. An application requirement was to be able to move service accounts from one customer account to another. The solution to this requirement achieved this by creating new service account rows and deleting the old ones. This involved several locks being taken over multiple partitions.

In practice these service account moves, a very low volume compared to the number of Service Accounts being billed daily, were scheduled to run at a time when billing was not running and other activity was minimal. This avoided the risk of time-outs and deadlocks. As billing was cloned and thus had a shorter elapsed time, a larger scheduling window was available for the moving of the required service accounts.

To fully exploit Parallel Sysplex and data sharing, the architecture of an application should adhere to the following design principles:

- Design the individual batch processes so that they can be replicated or cloned; that is, copies of the same program can run independently against different key ranges of the databases simultaneously (-void logical and physical contention across parallel streams), and balancing workload across streams.

Regardless of whether the batch process is data driven or transaction-driven, try to balance activity across the clones/partitions.

Design the database to enable optimum performance for its large volume processing using clustering and key range partitioning

- Design all functions, both batch and online, so that they can all run at any time in parallel with any other function. This may not be totally achievable, as some business requirements may have to run in isolation to other function, but the design must minimize such serialization requirements to maximize application availability (for example, can the batch run into the online day by committing more often?).
- Minimize locking protocol used by DB2 applications consistent with their requirements.
- Long-running DB2 applications (for example, batch) should be designed so that they can take frequent intermediate commit points driven by metadata stored outside of the program and can be modified easily.

If these design principles are adhered to, the ability to run the applications functions anywhere. On any logical partition, participating in a Parallel Sysplex with active data sharing will be achievable and the application will be able to fully exploit the processing power available in the run time environment. It is good design practice to follow these principles for an application designed to run in a single logical partition in order to allow for future scalability requirements.

Archived

DFSMStvs application considerations

In this chapter we provide guidance for application programmers who want to create applications and batch jobs that run in a DFSMStvs environment and exploit the data sharing capabilities. We discuss the application considerations in a DFSMStvs environment, with focus on the migration tasks and efforts from a traditional world.

The chapter contains:

- ▶ “Introduction to DFSMStvs” on page 68
- ▶ “Batch application considerations” on page 69
- ▶ “Transactional recovery” on page 71
- ▶ “Programming changes” on page 77

4.1 Introduction to DFSMStvs

Prior to the availability of DFSMStvs, the following mechanisms were available for sharing VSAM files:

1. Using VSAM share options, which are specified when defining a VSAM file.
2. Using z/OS serialization services and organizing all sharing through the application. A variation of this is the DISP parameter in JCL, which implicitly issues z/OS serialization services which apply to the entire VSAM file.
3. Using VSAM record-level sharing (RLS), which allows multiple batch readers to share a file with CICS systems, which may be writers to files.

Even given these capabilities, the VSAM environment still presented restrictions to sharing files between different users. For example, one significant impact for installations was that VSAM provided sharing support that is not sufficient to allow CICS still to be up and running when batch jobs update the very same file at the same time.

VSAM has continued to evolve over recent years and now provides an additional step beyond RLS to provide functions similar to those of a database manager.

As illustrated in Figure 4-1, DFSMStvs provides the ability for files to be updated by the CICS transaction server and batch jobs concurrently. So, the transaction server can provide continuous service to the attached network and no longer needs to be stopped in order for batch update jobs to be run.

DFSMStvs builds on the functions provided by coupling facilities. It uses z/OS services for logging into and through the Coupling Facility and for providing management of sync points, as well as building on the locking and data caching functions provided by VSAM RLS, which also uses the Coupling Facility for these functions.

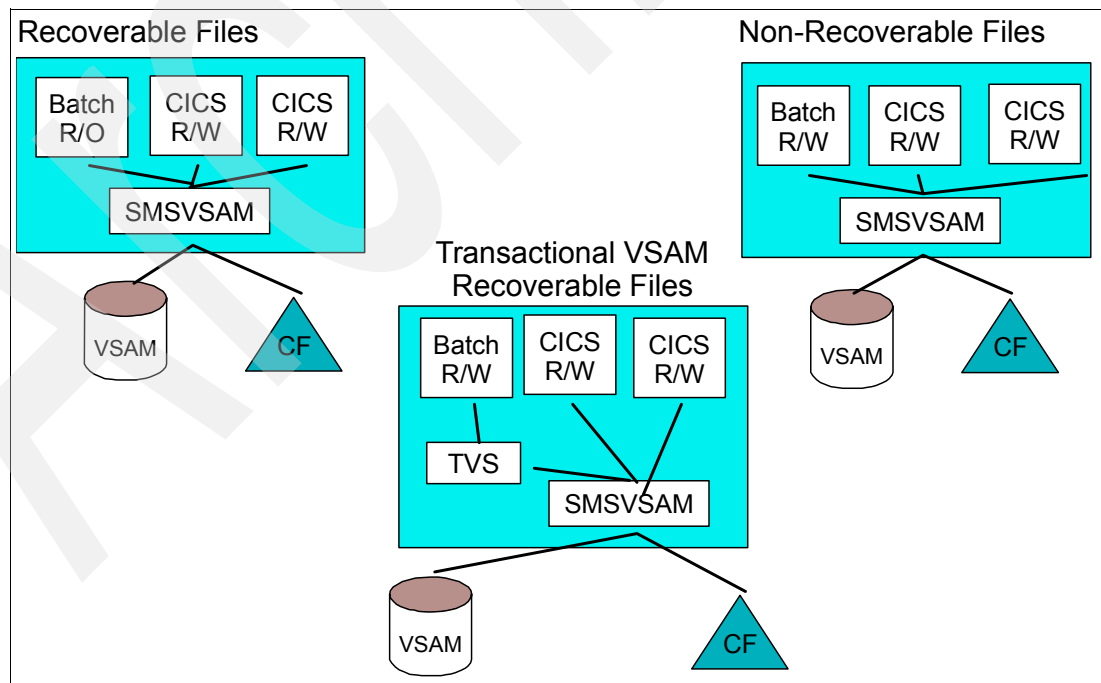


Figure 4-1 DFSMStvs

When migrating your environment to DFSMStvs, two major environments are going to be impacted. You must analyze them to determine whether they should be changed to perform correctly in the new infrastructure:

- ▶ Batch application programs
Verify how to access the VSAM file, whether and how they handle the commit.
- ▶ The batch job flow
You need to understand the transactional recoverability and the restart capability structure.

The following sections explain these concepts and provide samples and potential solutions you can apply to exploit the new DFSMStvs infrastructure.

4.2 Batch application considerations

Usually, a batch job step modifies some interim or permanent master files without any consideration of logging, two-phase commit or atomic updates. DFSMStvs introduces the concept of *transactions* to batch jobs. This concept requires moderate changes to existing programs to allow recoverability and minimal impact on other users in a shared environment.

4.2.1 Unit of recovery

A *unit of recovery* is defined as the set of all changes between two sync points. When using DFSMStvs, you must include all interrelated files within the update step in the same units of recovery. This applies for VSAM files only, as there is no support for non-VSAM files.

You can no longer use file backup and restore as part of your restart design. Why? Because these files might now be used by other users or applications due to the DFSMStvs capability of sharing these files—and recovering a file to permit restart of a failed step will wipe out the changes committed by the other users.

The atomic update approach is based on the idea of all-or-nothing, which has been used for many years in transaction processing programs. Three basic functions together provide this backout capability as implemented and used by DFSMStvs:

- ▶ Resource locking
This function is provided in VSAM RLS and is totally transparent to DFSMStvs. To facilitate a high level of concurrent execution of transactions, a fine granularity for locking is used. DFSMStvs uses VSAM RLS to serialize and lock on a logical record's granularity within VSAM files. In contrast, a conventional z/OS file is serialized at the entire file level, independent on the number of logical records within this file.
- ▶ Resource recovery logging
DFSMStvs uses system logger services to record all changes. Recording of the changes usually takes place in the coupling facility.
- ▶ Two-phase commit or backout
DFSMStvs uses z/OS recoverable resource services (RRS) to provide this function. A commit makes the changes permanent. If a problem occurs before the commit, RRS instructs DFSMStvs to back out the changes.

All three functions are required to provide a transactional approach to manage multiple resources as a logical entity: First, serializing and locking the resources which are going to be changed, and second, the logging of changed resources or records. To be able to back out, images of the records are written to a backout or undo log prior being changed. When forward

recovery ability is requested, records are also written to a forward recovery or redo log after they were changed. Third, this two-phase commit and backout protocol ensures that all changes occur as a single entity, or atomically, which is that either all of the changes within this transaction occur, or none of the changes occur.

Commit frequency

As mentioned, a unit of recovery is the set of all changes between two sync points, so in practical terms this leads to the question of how many changes there should be between two commit points. This is important because the duration of locks being held can depend on the interval between sync points.

The number of changes might be determined by the application implicitly because of the duration of a business transaction. It might also be determined by the duration of a program or, within a program, when a logical entity can be defined. At the end of such a window or cycle, a commit will harden the changes in all related files.

This is a new approach for traditional batch programs. One way to achieve it is to do commit processing after a certain number of updates. This requires the programmer to break processing within the batch program into groups of update I/Os comprising a unit of recovery. Please note that, during a unit of recovery, locks are held for all the records concerned until a commit or backout takes place to end the unit of recovery. On one side, commit will release the locks, thus minimizing locking contention; on the other hand, each commit or backout uses system resources.

4.2.2 Sharing files in the batch job structure

With the DFSMSHVS capability to share files between multiple batch jobs, current job sequences have to be carefully reviewed. Often, application dependencies dictate the processing sequence and there might be little flexibility to run these processes in parallel rather than serially. Running more than one batch update job against the same file has the potential for logical data inconsistencies. Again, careful planning and analysis of which process might change a file in parallel with other processes leads to a redesign of job schedules.

You should also understand that, without any effort to run jobs in parallel, DFSMSHVS will lead to greater batch elapsed times. Only a higher degree of running batch jobs in parallel might lead to an overall shorter batch elapsed time. For additional detail, refer to *DFSMSHVS Application Migration Guide*, SG24-6972.

Application restart

You can no longer use file backup and restore as part of your restart design. When implementing transactional concepts within batch jobs and utilizing DFSMSHVS, a job or step restart design leads to a quicker restart process, especially within long-running update steps.

Refer to *DFSMSHVS Application Migration Guide*, SG24-6972, for programming examples that show how to use DFSMSHVS for a quick restart after the last successful commit by using a *positioning file* (a recoverable VSAM file updated in the same unit of recovery as the file to which it refers).

Batch performance

VSAM performs best when running in non-shared resources (NSR) mode. Because no resources are shared, no serialization effort is required and, therefore, there is no cost for serializing except for some basic support through VSAM's share options. But, as the name implies, the NSR resources are not shared with other users or applications.

With local shared resources (LSR) and global shared resources (GSR), VSAM provides extra functions to allow sharing to some degree and to manage buffer pools. Record-level sharing (RLS) goes another step further and provides sharing support across a Parallel Sysplex on a logical record level.

With DFSMStvs, the process is taken one step further by providing serialization support through record locking, logging support for commit or backout, as well as forward recovery functionality through forward recovery logging. All of this does not come for free, and requires some extra effort in various system areas.

Compared with NSR or LSR, DFSMStvs requires some extra processing cost in the following areas:

- ▶ Cross-address space communication
- ▶ Record locking
- ▶ Data logging for commit or backout
- ▶ Commit or backout processing
- ▶ Forward recovery logging
- ▶ Coupling Facility cache access
- ▶ Loss of chained sequential I/O
- ▶ Loss of deferred write

Taking these into consideration, DFSMStvs performance cannot be compared directly with the performance of NSR access to VSAM files because so much more work is done on behalf of the application. Batch applications that use DFSMStvs will take longer than traditional batch applications that were designed to run in a batch window without a concurrent online system. The motivation to use DFSMStvs comes from the need to access VSAM files concurrently, especially with a transaction server which is up and providing online service even during batch update processes.

4.3 Transactional recovery

The concept of transactional recovery is new to VSAM batch jobs, but familiar from other types of work. The concept of transactions and transaction processing is common in products such as CICS, IMS, and DB2, and is now also important for DFSMStvs batch jobs.

A transactional view implies that a number of changes within the scope of a transaction are internal to the transaction itself until this transaction reaches a point of logical completion, when all changes are committed: they become permanent and visible to all other users. If an application does not wish to commit all changes at logical completion of the transaction, all outstanding changes are withdrawn by a backout and no changes are made within the scope of that transaction.

This particular capability of backing out, or transactional recovery, is important to transaction processing. Furthermore, because a data set can be shared, you cannot assume exclusive use during recovery; neither can you do anything to remove changes committed by other users of that data.

Batch application restart

Because of the sharing capabilities of DFSMStvs, using it within batch jobs not only requires changes to application programs themselves, but also changes the restart design of the batch job. DFSMStvs makes it possible to define a much shorter period to restart a batch process after a system or program failure when using commit logic.

Note that the following example does not include point-in-time backups and the availability of intermediate files within a more complex sequence of steps within batch jobs. The main purpose here is to simply show why DFSMStvs requires changes in existing steps.

When a conventional batch job uses simple files without using logging services or database manager support, usually a restart after a failure means that you start from the very beginning of the job or job step.

The flowchart illustrates the Material Requirements Planning (MRP) process. It begins with 'Files' and 'QSAM' (Quasi-Sequential Access Method) data sources feeding into the 'Collect' step. The process then flows through 'Sort' and 'Update' steps, which also interact with 'QSAM' databases. A 'Transaction Server' is shown as a central component, with a red 'X' indicating a break or a specific connection point. The process continues through 'Reports' and 'Create' steps. The 'Create' step outputs a 'New Master File' (labeled 'Legal version') and a 'Master File' (labeled 'Master File'). The 'Master File' is then used in the 'Update' step, which also interacts with a 'QSAM' database. The 'Create' step also interacts with a 'VSAM' (Variable-Size Access Method) database.

Figure 4-2 Conventional batch job flow

1. In the first step, COLLECT, some input files are collected and consolidated into a single file. These files are QSAM files. Input files might come from files that arrived through the File Transfer Program (FTP), in the system, or from files that are output files from other processes. If this step fails, it must be restarted from the beginning of the job step.
2. The next step, SORT, is a common step in batch job streams. Because sequential or skip sequential updates to a master file are faster than random updates, the QSAM input file is usually sorted in key sequence in order to allow sequential update process. The sort output file is also QSAM. If the SORT step fails, restart at the beginning of the step is

sufficient, because the consolidated input file is not affected by a failure during execution of the SORT step.

3. The most important step in this small sequence is the UPDATE step. The sorted input records are used to update a master file. This master file is often an unloaded database file and is processed with the QSAM I/O access method for performance reasons. The number of input records and the complexity of each single update process determine the elapsed time of this step. When a failure causes the update program to abnormally end (ABEND), the master file is already changed to some extent. When there is no specific code in the update program to provide information on how far the update process got, a restart requires that this step must be completely repeated after restoring the master file from a previous copy taken at the start of the job step.

When this update process is a long-running step, the update program usually provides some provision to allow a restart within the job step. There is no simple way to provide an easy restart within the UPDATE step. One possible approach might be to erase the input record as soon as the change in the associated master file is complete. Then, a restart just re-opens the input file for sequential update and continues the update process, which is a simple process.

However, this approach requires that the program does not simply read the input file but also that, within the same unit of recovery, it deletes each logical record from the input file after the update process to the master file has successfully completed. This may well require a different file organization to allow deletion of records from the input file. Also, the input file is read in an inefficient way by reading each single logical record. A blocked I/O gives the potential to lose data when restarting after a system failure.

Another approach, taken in more advanced update programs, is to write an additional file which has logging characteristics, but this requires more programming effort and introduces more complexity into the application program. Usually, the UPDATE step processes more than just one single file. It is also common for more than one update program to change the same master file. Because no locking support is provided by simple sequential access methods, these UPDATE jobs and UPDATE steps cannot run in parallel.

A solution to these issues is to use a database manager, which coordinates all I/Os to concerned files, or use enhanced functions in z/OS. VSAM RLS and DFSMSStvs within the z/OS DFSMS component provide support for locking records and logging changes during concurrent update processes to the very same VSAM master file.

4. The steps following the update read the updated master file and create reports or statements. As the last step, the updated master file is input to a reload process that creates a new master file, which then becomes the new master file for the next online cycle. This also provides a reorganization of the master file as a side effect. A backup copy process provides data availability by creating one or more copies to tape or automated tape libraries, and perhaps on to other disks as well so that it is possible to restore the data quickly in the near future.

The following sections outline two alternative ways to provide restart in a DFSMSStvs environment, and show what changes are required to the original batch job.

4.3.1 Using DFSMSStvs: unique input

Figure 4-3 on page 74 outlines how the batch job in the previous example changes when using DFSMSStvs. This alternative implies that the input files are consolidated into a VSAM KSDS and are only needed for this particular batch job. That is, the input files are unique to this job and do not serve as input to any other job or step. We will consider each step again to see how we can restart after a failure during batch job execution.

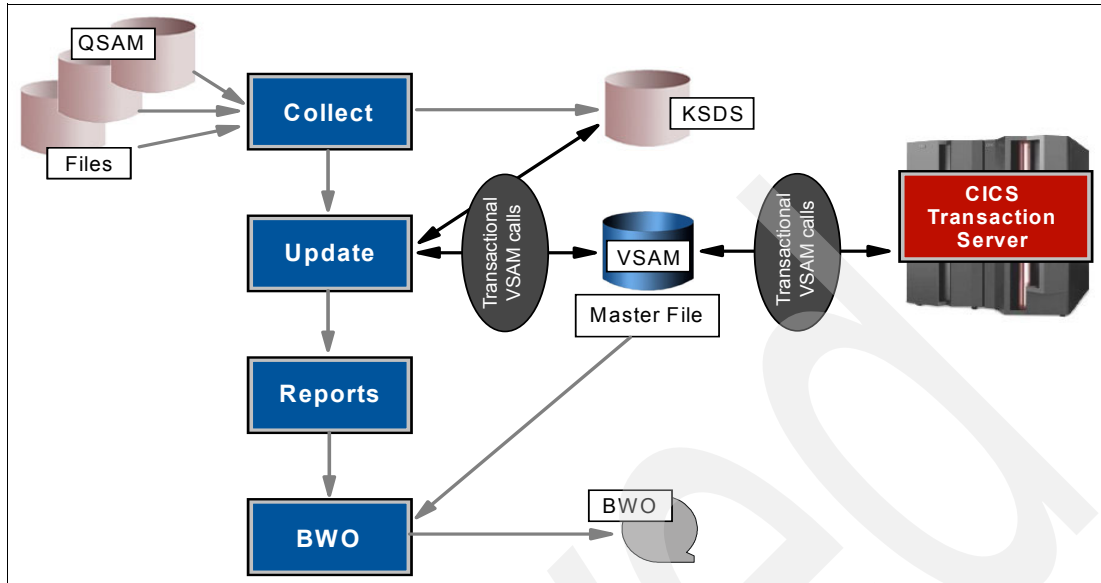


Figure 4-3 Changed job flow with DFSMStvs

1. The COLLECT step changes so that it now consolidates all the input files into a recoverable VSAM KSDS file. This KSDS contains all the collected input records and becomes part of the DFSMStvs environment in the next step. This is so that we can delete records for restart purposes and be sure that we always have the update and corresponding delete in the same unit of recovery. The COLLECT step might just be an IDCAMS REPRO step to load the KSDS, which becomes the input file for the next step.

If this step fails, it must be restarted from the beginning because the input files are non-VSAM and hence non-recoverable files. The COLLECT consolidates all input files into a KSDS. Assuming that the COLLECT program uses random writes to consolidate all input files into a KSDS, it has to be carefully planned due to the fact that random updates are less efficient than sequential or skip sequential updates.

As a basic rule of thumb, random update is feasible when less than five percent of the records in the master file are changed. This is not the case in this example, because an entirely new KSDS is created. Optionally, a SORT step before the COLLECT step allows you to load the KSDS in sequential mode, which is quicker than loading the KSDS in direct update mode.

2. The UPDATE program now runs in a DFSMStvs environment. DFSMStvs provides record level locking, as well as logging of all updates, with before and after images of the master file data and the input file to make forward or backward recovery possible at any point in time after a failure during the UPDATE step. The input KSDS and master file are both part of the DFSMStvs environment. This implies that each input record is deleted after successfully applying the changes to the master file.

After a certain number of updates, commit processing hardens the changes to both files. After a failure during this step, a restart does not require any particular code to handle the restart situation. An implicit backout at the time of failure guarantees that the input records and update activities are backed out to the point in time of the last commit before the failure. Then, the update program just continues after restart, and processes the remaining input records from the KSDS input file.

Note that CICS Transaction Server (CICS TS) transactions can access the master file at any stage concurrently with the batch job. Other batch programs updating the same master file can now run concurrently as well and so might shorten the overall batch

window. The most significant advantage here is that CICS TS can continue to run and service transactions without any service interruption due to batch processing.

3. REPORTS creates reports or statements.
4. Backup of the master file takes place after all batch updates are completed. Creating a backup copy is possible without interrupting the service by utilizing backup-while-open (BWO) techniques. Note that this approach performs two I/Os for every record in the input file under all conditions; you always read a record and delete it, whether during a normal run or a rerun.

4.3.2 Using DFSMStvs: shared input

The previous alternative assumed that, after making the changes to the master file, all the related input records could be deleted. When the consolidated input file is also required by other batch jobs and programs, the previous approach cannot be used. This second alternative represents a smaller modification to the original batch job, and keeps all records in the consolidated input file untouched; see Figure 4-4.

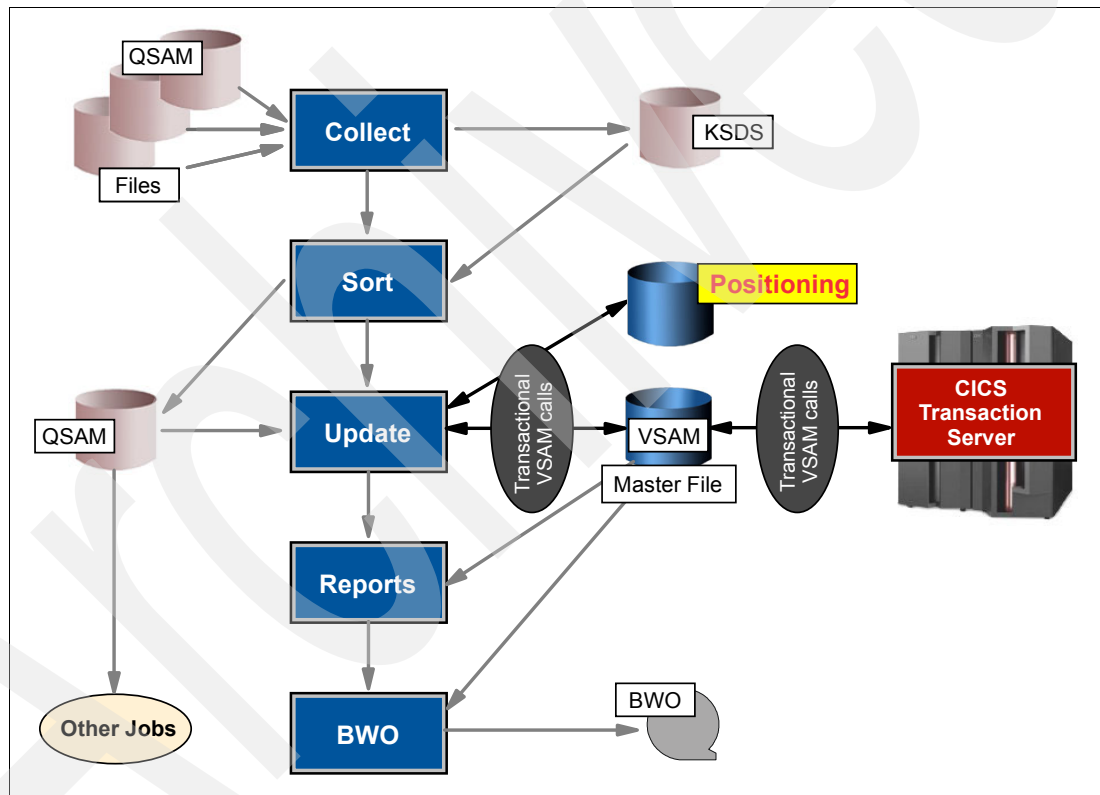


Figure 4-4 DFSMStvs in update step only

Here, DFSMStvs is implemented mainly to provide uninterrupted service availability of CICS TS and also allow us to run one or more batch jobs concurrently. There is no change to the first two steps concerning restart considerations compared to the original batch job.

The application program in the UPDATE step needs modifications to make use of DFSMStvs and share the master file. Restart considerations require you to keep the actual restart process as short as possible, and restart without affecting the availability of the master file for CICS/TS.

In addition to the master file, we suggest using a positioning file (see Figure 4-5) to record how far the application program in the UPDATE step processed the input file, to be used in the event of a failure within this step.

Because all the changes were already applied to the master file up to the last commit point before a failure occurred, a restart must first determine where to restart in the input file, because the input file is not part of the DFSMSHvs environment. So, backout processing applies only to the master file and the positioning file.

When reaching a commit point, a record is written to the positioning file indicating the key of the last record processed in the master file within this unit of recovery, just before the actual commit process; this is shown in Figure 4-5.

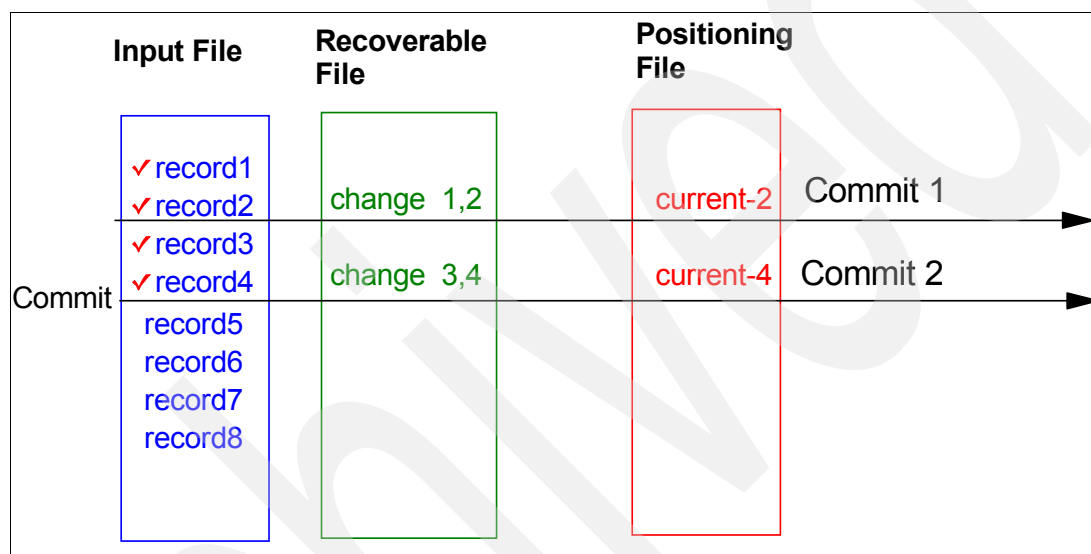


Figure 4-5 Positioning technique

We show three updates in each unit of recovery to simplify the diagram. Of course, in practice, you would perform many more updates in each unit of recovery.

The commit process includes both the positioning file and the master file, and therefore guarantees that both files are synchronized. A failure of the commit leads to a backout for both files to ensure a proper restart point within the input file. It is sufficient to keep only one record in the positioning file. The record always contains the last input record or the key of the last input record within the previous commit to the master file.

The application program for the UPDATE step now contains additional restart logic:

- ▶ Define the new positioning file as a recoverable VSAM RRDS. The logical record structure of this file just consists of the key of the input file (containing transactions to update the master file), plus a count indicating the number of successful commits.
- ▶ Open the positioning file for update.
- ▶ Read the positioning file. A number of zero for successful commits indicates that there have not yet been any permanent changes to the master file in this batch cycle. The program continues with the normal program logic and starts with the very first record in the input file.
- ▶ If reading of the positioning file returns a record that contains a number greater than one for successful commits, the record also provides the key of the last input record which made it within the most recent successful commit process to the master file. The program

reads through the input file, skipping over records until reaching this key, and then continues with normal program logic from the next record onwards.

When reaching a commit point, a positioning record is written to the positioning file indicating the last successfully processed input record within this unit of recovery and the incremented number of successful commits. Just updating this positioning record in place is sufficient. Both files are subject to the commit process so they are kept synchronized.

At this point, you can see that the advantage of the shared input approach is that the basic job structure requires little change. The UPDATE step has to be changed in the following ways:

1. Include JCL for the positioning file.
2. Define the master file and indicate DFSMSStvs mode.
3. Provide restart and commit/backout logic.

The application program requires restart logic like that outlined above. In the event of a failure, the positioning file indicates the offset in the input file from where to resume update processing. This requires the program to read the input file again, from the very beginning to the last successfully processed record.

This approach means that, in a successful run, you only read each input record once. You also do some I/O to the positioning file, but this is a small amount if the commit frequency is well-chosen. In the event of a rerun, you reread input records to skip over them. Even in the worst case of a failure just before the last input record, you read every record twice, which is no worse than the unique input case. Therefore, this approach is optimized for the normal processing path.

In the event of a restart, the alternative with unique input does not require the program to execute any specific restart logic nor to read the input file again from the very beginning. This is possible because you have deleted each input record after the record is successfully processed. So to restart, you simply start from the beginning of the input file, which is the normal program logic in the first place.

However, either approach requires that you change the structure of the batch job. This might shorten the flow, but it implies a significant change in the UPDATE step. In addition, this step requires that you create the KSDS input file in update mode. This is not as efficient as creating a file in sequential mode.

Another drawback of the unique input approach is that all input records are deleted by the end of the job step and therefore cannot be re-read by other jobs or programs. We believe that shared input will probably be the choice in most cases.

4.4 Programming changes

Application programs that use DFSMSStvs use the same I/O statements as any other program which uses VSAM to read and write data sets. Common languages like COBOL, object-oriented COBOL, C, C++, PL/I or assembler would not require any language-specific syntax changes for DFSMSStvs.

The only change with DFSMSStvs is the ability to request CRE using MACRF=RLS in the ACB parameter in Assembler. In COBOL and PL/I, there are no specific changes that need to be done to a program for it to access data in DFSMSStvs mode; however, in most cases, we would recommend strongly that changes be made to ensure that a program does not adversely affect other programs by holding locks longer than necessary.

In the flow of the program, there are several specific error conditions which should be handled in the program. A list of the error codes and conditions is given in *Overview and Planning Guide*, SG24-6971.

The logic of the program should be changed in such a way that there is scope for restarting the program. The restart logic is done on the basis of the number of records that have been successfully processed. The count of the number of records processed successfully is put in a positioning file which can be referred to later by the program. While restarting, this acts as a positioning file to get to the position from which the records need to be processed further.

As a part of the programming logic, care should be taken to commit the updates frequently. The best number of updates between commits must be decided by the application programmer, based on a trade-off between contention on locking and the cost of commit, as indicated in “Commit frequency” on page 70. When there is no explicit commit in the program, an implicit commit is done at the end of the program. However, we recommend that you use explicit commits.

There are a few changes that should be made in the JCL when you need to run the DFSMStvs program. Table 2 in *Overview and Planning Guide*, SG24-6971, tabulates the changes to the JCL.

How to commit

It is difficult to know how often an application program should commit. You can estimate how often to commit, and also retain the flexibility to change this easily, by passing the number of updates between commits to the program in a parameter file or as a parameter on the program's EXEC JCL card. As mentioned earlier, it is a good idea to commit the changes frequently.

The following figures show the code fragments that need to be included in order to call RRS to commit changes and code fragments to get and check the return codes and reason codes.

Example 4-1 shows a code fragment to do this for assembler programs.

Example 4-1 Assembler commit call

```
COMMIT EQU *
      CALL SRRCMIT,RETCODE,LINKINST=BASSM
      L 8,RETCODE
      LTR 8,8
      BZ CONTINUE
      {process non-zero return code from commit}
CONTINUE EQU *
```

For PL/I programs, you need an additional call to another error procedure, CEE3CIB. This Language Environment procedure returns the VSAM error information that should be handled in the PL/I program. Example 4-2 shows the syntax for commit and backout calls for PL/I programs and also shows the external subroutine declaration that is needed.

The complete sample program, including source code and call to CEE3CIB, is discussed in detail in *DFSMStvs Application Migration Guide*, SG24-6972.

Example 4-2 PL/I commit call

```
DECLARE (SRRCMIT) EXTERNAL ENTRY;

CALL SRRCMIT(RETCODE);
```

For COBOL programs, see Example 4-3 on page 79.

How to perform a backout

A backout is done in a similar way to a commit. The application program needs to decide when it would want to back out the changes.

As a specific case, an application program might want to back out changes instead of retrying if return codes indicate a deadlock or retained locks. A backout is done automatically if a commit fails.

A sample backout call is shown in Example 4-3 for PL/I.

Example 4-3 PL/I backout call

```
DECLARE (SRRBACK) EXTERNAL ENTRY;
```

```
CALL SRRBACK(RETCODE)
```

For COBOL, you might use code similar to this:

```
Call "SRRCMIT" using COMMIT-RESULT  
  
Display " return code " COMMIT-RESULT
```

Application program migration steps

The following is a brief list of what should be the major steps that need to be revised in the application program logic to verify that is ready to run in a DFSMStvs environment. Note that the program is not necessarily usable after each step. For example, you would not want to implement commit without adding restart logic, although these two steps are shown here separately for clarity.

1. Ensure that the opening on your VSAM data set is in DFSMStvs mode.
2. Add checks for deadlock detection or request time-out after each VSAM request for a recoverable data set. For example, you can chose to back out changes and stop the program if a deadlock was detected.
3. Add commit and backout logic. At this stage, the recovery procedures must also be changed as you must use CICSVR or something similar for forward recovery from the log if you recover a point-in-time backup. You can also choose to add a parameter file so that you could control the number of updates between commit attempts.
4. Add restart logic using a positioning data set.

For more detailed information on the examples refer to *DFSMStvs Application Migration Guide*, SG24-6972.

Archived



CICS application considerations

In this chapter we provide guidance for application programmers who want to create applications that run in CICS TS CICSplex® and z/OS Parallel Sysplex environments and exploit sysplex capabilities. We discuss the application considerations in a CICSplex environment, examine affinities, and describe the required actions.

This chapter contains:

- ▶ “Introduction to CICS TS” on page 82
- ▶ “How to exploit sysplex in CICS TS applications” on page 82
- ▶ “CICS TS application checklist” on page 92

5.1 Introduction to CICS TS

CICS TS is a transaction processing subsystem for high-volume, high-function online applications. Developed for the IBM mainframe platform more than 30 years ago, it has evolved steadily to take advantage of new hardware and software and to meet the growing requirements for online processing that have attended these enhancements. Hardware advances have included virtual storage, expanded storage, multiple-engine processors, storage protection, and an array of terminal devices. Major software interfaces include DB2, IMS, VSAM, VTAM, and TCP/IP. CICS TS has also been extended from its System/390® heritage to most other major platforms, so that it can support almost any client-server or other distributed-processing configuration.

As shown in Figure 5-1, the most recent version for the host platform, CICS Transaction Server for z/OS and OS/390, contains everything important to exploit Parallel Sysplex.

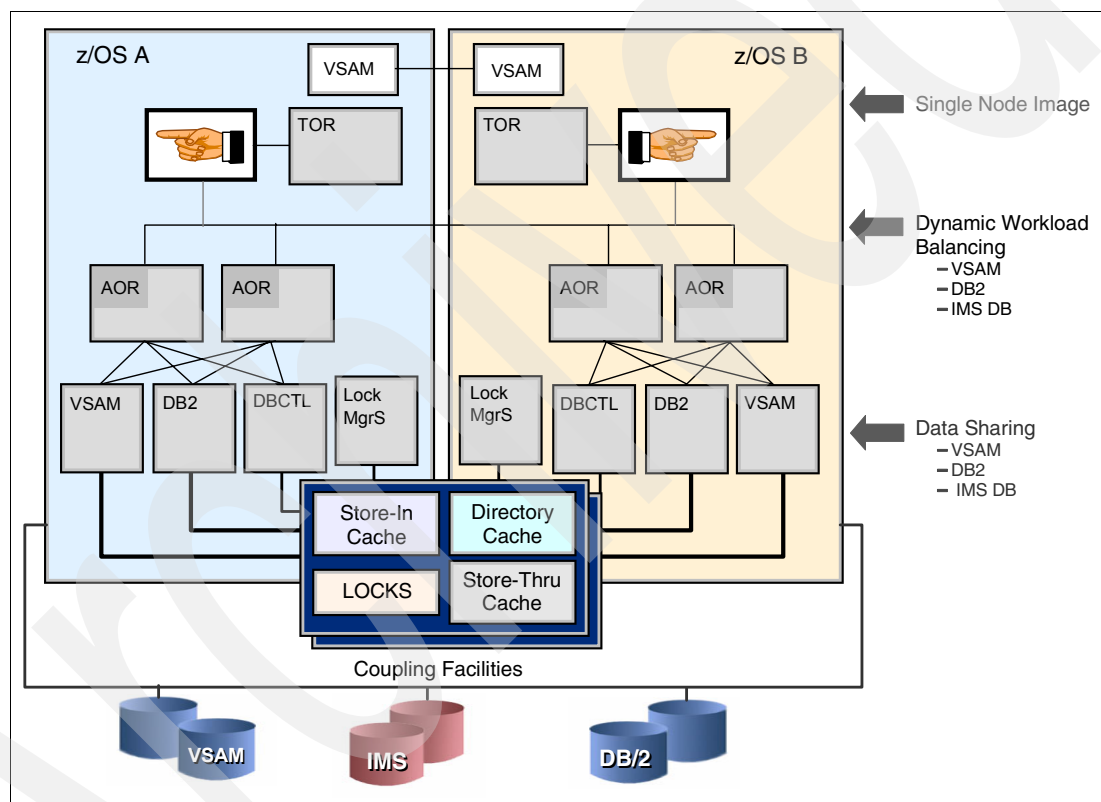


Figure 5-1 A CICSplex example

5.2 How to exploit sysplex in CICS TS applications

Existing CICS TS applications will run on a Parallel Sysplex without change. However, to exploit Parallel Sysplex so you can achieve high availability and increased capacity for an application, you need to be able to run multiple instances.

To effectively clone an application that resides in an application-owning region (AOR), transaction affinities to the region should not exist. For any new development, affinities to CICS TS AORs or logical partitions must be completely avoided. Regardless whether you use COBOL, PL/1, C, Assembler, Java, REXX, or even a combination to develop the applications, the avoidance of affinities, use of data sharing and the replication of CICS TS application

owning regions and applications are key to enabling your CICSplex applications to really profit from your sysplex environment. Although data sharing and the replication of the CICS TS regions is usually taken care of by your CICS TS system programmer, it is your task to ensure the applications are replicatable and without affinities. As in a non-sysplex environment, the code being reentrant will reduce program reload overheads, a real benefit when running multiple instances.

What is an affinity?

In order to avoid affinities, we must first understand what they are. An *affinity* is a requirement that a transaction or program runs in a particular region. The affinity may be a transaction requiring a specific CICS region (for example, requiring a table kept in the common work area of that CICS), or it may be an inter-transaction affinity demonstrated by one transaction in a pseudo-conversational sequence passing data to the next. Some common examples of affinity types are listed in Table 5-1.

Table 5-1 Affinity types

Description	Type of affinity
Concurrent CICS Tasks exchanging data via a common area of memory or local resources such as queues or data tables.	Inter-transaction
A CICS task leaves state data for a subsequent task in local resources such as files, queues, storage, memory, and so on.	Inter-transaction
CICS tasks synchronize their activities using CICS POST or WAIT ECB commands.	Inter-transaction
CICS environment is manipulated using INQUIRE and SET commands.	Inter-transaction (can be transaction-system depending on reason for manipulation)
Dependency on resources only available in certain regions such as data tables, queues, files, DB2, programs, user exits.	Transaction-system

There are also a few *image affinities*, which are requirements that a transaction execute within a particular z/OS image. They arise most often from the use of a database manager that cannot share data among images, or direct use of z/OS facilities such as the common system area.

Affinity lifetime

The *affinity lifetime* determines when the affinity is ended. An affinity lifetime can be classified as one of:

- System** The affinity lasts for as long as the target region exists, and ends whenever the target region terminates (at a normal, immediate, or abnormal termination). (The resource shared by transactions that take part in the affinity is not recoverable across CICS restarts.)
- Permanent** The affinity extends across all CICS restarts. (The resource shared by transactions that take part in the affinity is recoverable across CICS restarts.) This is the most restrictive of all the inter-transaction affinities.
- Process** The affinity exists until the CICS Business Transaction Services (BTS) process completes.
A Process is a collection of one or more BTS activities. It has a unique name by which it can be referenced and invoked. Typically, a process

is an instance of a business transaction. In a vacation scenario, an instance of the business transaction may be started to sell Jane Doe a vacation in Florida. To identify this particular transaction as relating to Jane Doe, the process could be given the name of Jane Doe's account number.

Activity	<p>The affinity exists until the CICS Business Transaction Services activity completes.</p> <p>An activity is the basic unit of BTS execution. Typically, it represents one of the actions of a business transaction--in a vacation scenario, renting a car, for instance. A program that implements an activity differs from a traditional CICS application program only in its being designed to respond to BTS events. Activities can be hierarchically organized, in a tree structure.</p>
Pseudo conversation	The (LUname or userid) affinity lasts for the whole pseudo conversation, and ends when the pseudo conversation ends at the terminal.
Logon	The (LUname) affinity lasts for as long as the terminal remains logged on to CICS, and ends when the terminal logs off.
Signon	The (userid) affinity lasts for as long as the user is signed on, and ends when the user signs off.

Affinity relation

The *affinity relation* determines how the dynamic routing program selects a target region for a transaction instance having the affinity. An affinity relation can be classified as one of:

Global	A group of transactions where all instances of all transactions in the group that are initiated from any terminal must execute in the same target region for the lifetime of the affinity. The affinity lifetime for global relations can be system or permanent.
BAPPL	All instances of all transactions in the group are associated with the same CICS Business Transaction Services (BTS) process. Many different user IDs and terminals may be associated with the transactions included in this affinity group. The affinity lifetime for BAPPL can be Activity, System, or Permanent. (If you are running CICSplex SM, this affinity lasts for as long as any CMAS involved in managing the CICSplex using the workload is active.) A typical example of transactions that have a BAPPL relation is where a local temporary storage queue is used to pass data between the transactions within a BTS activity or process.
LUname	A group of transactions where all instances of all transactions in the group that are initiated from the same terminal must execute in the same target region for the lifetime of the affinity. The affinity lifetime for LUname relations can be pseudo conversation, logon, system, or permanent.
Userid	A group of transactions where all instances of the transactions that are initiated from a terminal and executed on behalf of the same userid must execute in the same target region for the lifetime of the affinity. The affinity lifetime for userid relations can be pseudo conversation, signon, system, or permanent.

Notes:

- ▶ For userid affinities, the pseudo conversation and signon lifetimes are possible only in those situations where one user per userid is permitted. Such lifetimes are meaningless if multiple users are permitted to be signed on with the same userid at the same time (at different terminals).
- ▶ If an affinity is both userid and LUName (that is, all instances of all transactions in the group were initiated from the same terminal and by the same userid), LUName takes precedence.

Why affinities matter

Affinities impose restrictions on where a particular transaction can be routed, thereby reducing your ability to exploit the sysplex environment. If a transaction is not routed taking account of an existing affinity, it will be unable to access some or all of the data it requires. This results in either failure to execute at all or, more seriously, incorrect execution by using invalid or missing data.

In order for a transaction to execute correctly in a sysplex environment, you must either remove its affinities or accommodate them. With a transaction-system affinity, direct the transaction to a specific region. With an inter-transaction affinity, direct the transaction to the same region as related predecessor transactions.

The decision about whether to remove or tolerate an affinity depends on the difficulty in removing it while continuing to meet the needs of the business, as well as the extent to which it interferes with the goals you are trying to achieve (such as workload balancing, high availability, additional capacity, and so on).

Many affinities may not interfere with these goals and need not be removed. There is a problem only when affinities are such that they prevent the application from being parallelized to a level that is acceptable by the business requirement. For example, it would be a problem if an affinity ties together more than 25% of a application load and you intend to split the application in four equal clones, whether it is for workload balancing or business continuity reasons. But any affinity that affects only 10% of the total workload would probably be less of an issue.

A solid understanding of affinity sources will help you avoid them, while still meeting the needs of the business in new applications. Note that existing applications do not *necessarily* need to be changed to remove affinities, because affinities are not the result of poor design or implementation. Instead, they are result of a changing execution environment¹, and they appear more often in code that is designed to take maximum advantage of a current environment, rather than in very conventional applications.

5.2.1 Affinities through use of CICS commands

In order to avoid affinities in applications, you first need to understand what causes them. The following information describes how some types of CICS commands can cause affinities.

Storage Management commands

CICS commands GETMAIN, FREEMAIN, LOAD and RELEASE potentially create affinities because, although storage allocated by a CICS task belongs to that task and is subject to the storage protection features in use, the storage can be seen and modified by other tasks in the same CICS region. A CICS program allocating storage for its own use naturally does not create any affinities—but as soon as a storage address is passed to another program, regardless of the method used, then an affinity exists between the two programs.

¹ For example, read-only tables in memory before VSAM buffering, or Data Tables to save I/O overhead.

Similarly, programs are sometimes loaded by CICS tasks for use as in-storage tables. As with any other block of storage, the task that loads the program can use the address supplied, but any other task must determine its location in the current CICS region by using INQUIRE PROGRAM.

Task synchronization commands using storage address or request id (REQID)

Use of CICS commands CANCEL, DELAY, POST, WAIT EVENT/EXTERNAL and WAITCICS cause affinities because, as just described, as soon as an address is passed to another program, an affinity exists between them. Similarly, some of these commands have a REQID which is only valid within a single CICS region.

Note: Use of these commands within a single program will *not* cause an affinity.

Task synchronization commands utilizing names

For CICS TS Releases prior to 1.3 and previous versions of CICS, use of ENQ and DEQ commands meant an affinity between programs. But with the use of ENQMODELS introduced in CICS Transaction Server 1.3, these commands can be used within a sysplex without causing affinities; refer to *CICS Resource Definition Guide*, SC34-5990, for more information.

Temporary Storage commands

There are several types of Temporary Storage Queues manipulated by the commands WRITEQ TS, READQ TS and DELETEQ TS, with each type having its own implications for affinities and recoverability.

Queues that are held locally by a CICS create affinities. Queues held on a Shared Temporary Storage server are available to all CICS regions in the sysplex at very little overhead, but they are non-recoverable.

For shared queues that are used to hold read-only data, this is not a problem. However, many others need to be recoverable, so having the queues remote on a Queue-Owning Region (QOR) might be a more suitable option.

In both cases, some mechanism to ensure uniqueness of queue names across the sysplex where required is essential.

Manipulation of resource and CICS status

The commands ADDRESS, COLLECT STATISTICS, ENABLE, DISABLE, EXTRACT EXIT, INQUIRE, SET, PERFORM, RESYNC, CREATE and DISCARD either receive information about, or change, the status of local resources or the CICS region itself. Any program that either receives this information from, or replies on the actions of another, has an affinity, so although these commands can be used, the information should not be shared between tasks.

Starting of tasks

Starting tasks is an area with great potential for creating affinities. If the start is for a “fire and forget” task, then any data supplied does not have any affinities and the task can be routed to any cloned AOR without any problems.

However, if there is a need for some form of synchronization between two tasks using either the commands mentioned previously or the command RETRIEVE WAIT, care must be taken to avoid affinities; refer to “Task synchronization” on page 90 for more information.

Affinities through specification of SYSID in commands

There are many commands in CICS where the command can be routed to another CICS by use of the SYSID option. Although this does not preclude the application from being sysplex-enabled, it does force certain resources onto specific CICS regions. (This may be the case when business or application requirements dictate that certain resources such as files, programs, or temporary storage reside on a single CICS AOR.)

Note: The use of the SYSID option does mean that any future change of the resource location requires updates to potentially many programs, rather than the location being determined by CICS definitions.

Document commands

There are many document commands such as EXEC CICS document create, insert, retrieve, EXEC CICS Web Interface, and CICS Business Transactions Services interface which use a set of document commands. See *CICS Business Transaction Services*, SC34- 6237, for more detailed information on BTS Application programming commands

The documents these commands manipulate are private to the application; they cannot be accessed by other tasks. Their use will not create any affinities between tasks. However, keep in mind that the documents are *local resources* and cannot be accessed by programs forming part of the same task on other CICS regions.

Web commands

These commands, used to manipulate data passed between CICS and Web browsers, can only be used in the region that received the browser request. Similarly, transactions invoked over the Web 3270 bridge must perform their terminal activity on the region that received the Web browser request.

CICS Business Transaction Services (CBTS) browsing commands

These commands do not create an affinity as such, but are reported by the CICS Transaction Affinities Utility (see “CICS Transaction Affinities Utility” on page 87) because the information they supply is specific to the region on which they are issued.

5.2.2 CICS Transaction Affinities Utility

The CICS Transaction Affinities Utility is supplied as part of CICS TS to assist in detecting affinities within applications. It can be run in either within a CICS region detecting commands executed within the region, or against a program load library as shown in Figure 5-2 on page 88.

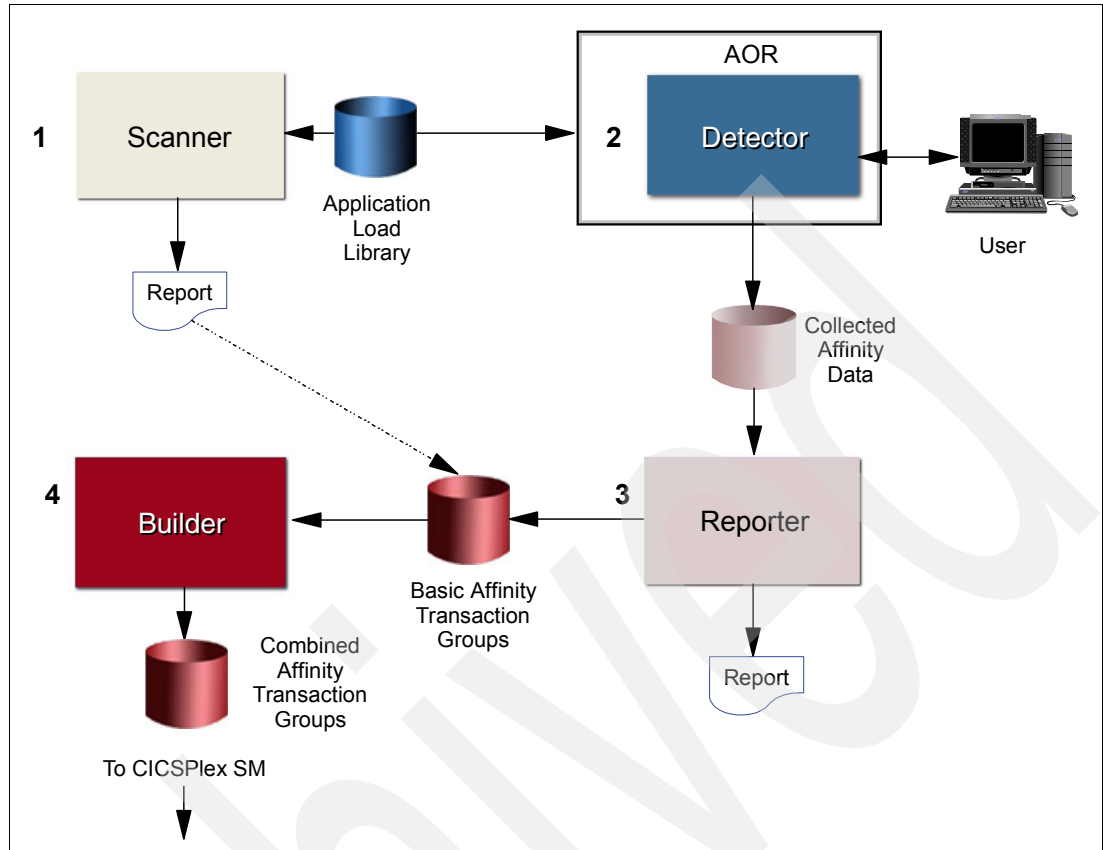


Figure 5-2 CICS TS Transaction Affinities Utility

However, keep in mind that the utility will only discover transactional affinities. There are other ways in which transactions can cause affinity with each other, but they are not readily detectable by the affinity utility program because they do not take place via the EXEC CICS API.

Table 5-2 lists affinities reported by the utility.

Table 5-2 Affinities

Inter-transaction affinity	Transaction-system affinity
ENQ DEQ READQ TS WRITEQ TS DELETEQ TS ADDRESS LOAD RELEASE GETMAIN FREEMAIN RETRIEVE WAIT DELAY POST START CANCEL COLLECT STATISTICS WAIT EVENT	ENABLE PROGRAM DISABLE PROGRAM EXTRACT EXIT INQUIRE SET PERFORM RESYNC DISCARD CREATE WAIT EXTERNAL WAIT EVENT WAITCICS CBTS STARTBROWSE CBTS GETNEXT CBTS ENDBROWSE

5.2.3 Programing facilities and techniques and their implications on affinities

Commarea and TCTUA

Using a commarea or the TCTUA to pass data between different programs within the same task will not create affinities provided they contain only data and not pointers, as CICS handles copying the data to a different CICS region and back (if required). Similarly, the TCTUA contents are available to any task next running on the terminal (provided it has not been detached from CICS), and data can be passed using a commarea by use of the RETURN TRANSID COMMAREA commarea.

Keep in mind, however, that the TCTUA is not copied across LU6.2 links and neither area is recoverable. Both areas should only be accessed through addresses supplied by CICS directly (such as by the ADDRESS command), rather than by using information determined by another task or program. For more information, including size limitations, refer to *CICS Application Programming Guide*, SC34-5993.

Shared Long-term Storage

Common Work Area (CWA)

The CWA is local to the CICS region and provided that its use is limited to being read-only, with the contents being consistent across all regions used for the same role and correct for that region, its use will not have any affinity implications. As with the commarea and TCTUA, its location must be determined using the CICS ADDRESS command.

GETMAIN SHARED

Storage allocated in CICS using a GETMAIN command with the SHARED option allocates storage that is not released when the task terminates. This address of this storage is frequently stored in a Temporary Storage Queue or the CWA. Holding the address in the CWA can be made affinity-safe if the data is read-only and duplicated in each appropriate region.

Note: Storing the address in a temporary storage queue is possible if the queue is a local queue, but this means that there is a dependency on the queue not being made shared in the future.

LOAD PROGRAM HOLD

This approach is often used to create a table in virtual storage, and it has affinity implications virtually identical to GETMAIN SHARED. A LOAD PROGRAM command causes CICS to load the specified program from disk (if a copy is not already in memory) and return the address through the SET option. The HOLD option means that the program is required after the task terminates.

The address of the loaded program can either be passed to other programs the same way as storage allocated through a GETMAIN SHARED, or by subsequent programs issuing a LOAD PROGRAM with the SET option. Again, if the storage is used for read-only access and duplicated in all appropriate regions, there is no risk of affinity.

For a table that is not frequently used and accessed read-only, simply use LOAD PROGRAM and allow CICS to delete the table if CICS becomes short on storage, then reload it later when it is next required. This does mean that all programs must use LOAD PROGRAM set to determine the location of the program, and there is a risk of delays while the program is reloaded, or of tasks not being authorized to load the table.

As already mentioned, shared storage can be used without affinity problems if it holds read-only data and the storage locations are determined from a *local* source. For data that is not read-only, then Shared or remote Temporary Storage Queues or Coupling Facility Data Tables should be used. Shared Data Tables, although effective in a single image environment, add a networking overhead in a sysplex environment.

Shared Task-lifetime Storage

Environments in which an owning task passes storage addresses to other tasks, and the two tasks coordinate the use of the storage through task synchronization commands (see “Task synchronization”) create an affinity between the two tasks, and this should be avoided if at all possible. Because of the nature of storage, the techniques discussed in “Shared Long-term Storage” on page 89 are not applicable to this type of environment.

Business needs may dictate that the two tasks share data, rather than one task performing the functions of both. In a sysplex environment, affinities can be avoided by Shared Temporary Storage Queues holding data, an initial queue name passed through a start command, and task synchronization using global ENQs and DEQs.

Task synchronization

WAIT EVENT, WAIT EXTERNAL and WAITCICS commands

These commands are used to synchronize a task with the completion of an event performed by another CICS TS using a z/OS task and signalled through the posting of an ECB. Any task that is waiting on this ECB naturally needs to have the ECB address passed to it, thereby creating an affinity. Because these are z/OS ECBs rather than CICS ones, the affinity is to the z/OS image rather than to the CICS region.

Use of these commands could be avoided by:

- ▶ Performing all work within the CICS regions using CICS tasks.
- ▶ For work initiated outside of CICS, using the External CICS Interfaces to either run programs inside CICS, or issuing global ENQ/DEQ commands through CICS to serialize access.

START and RETRIEVE WAIT commands

These commands always create an affinity, and they are often used to synchronize sharing of task-lifetime storage, which is an affinity in itself. A task running on one terminal issues a RETRIEVE WAIT and becomes suspended until another task issues a START command for that transaction, to that terminal, on that CICS.

This scenario offers the potential for unexpected affinities because the transaction will probably have been initiated on a TOR, but may have been routed to an AOR, which is where the follow-on start will need to be issued. If the task has finished running in the AOR but the remote terminal has not yet been deleted by CICS, the task will be restarted. But if the terminal has been deleted, the start will fail.

START and CANCEL with the REQID option

Use of the REQID option enables a delayed start to be cancelled before execution. The affinity here is that the cancel command needs to be issued on the same system as the start in order to be successful.

An alternative approach would be to modify the started task to expect to receive a temporary storage queue name data passed by the start command, perhaps using the QUEUE option. Entries in the queue (or the state of the queue itself) could be used to determine if the start is still required.

Use care with this approach to starting a task on a terminal, however; refer to the information about START command in *CICS Application Programming Reference*, SC34-6026, for further details. Another option is to use the PROTECT option on the start command, meaning that the start will only be issued if the task commits the work. If the task performs a rollback, the START is cancelled.

DELAY and CANCEL commands using the REQID

The delay command will cause a task to wait for a specific period of time or until cancelled by another task issuing the CANCEL command. This creates an affinity, as the CANCEL command has to be issued on the same region as the task.

Temporary Storage Queues

Queues holding read-only data

As mentioned, use of these queues does not cause a problem for affinities, as they can be held locally or in a Shared Temporary Storage Server. Holding them remotely introduces networking overhead—and a single point of failure, should the server fail. Locally held removes the need for a Shared Temporary Storage Server, but does add the requirement for each applicable region to ensure the queue exists when required.

Queues used exclusively by a single task

A queue used exclusively by a single task presents no affinity problems and can be held locally by the CICS region. From a sysplex perspective, in order to run multiple versions of the transaction, the queue name must be unique to ensure one task does not access data from another. You can achieve this, for example, by incorporating the CICS task number in the name or by using a named counter server.

Queues used to pass data between tasks

The attributes required for the queue being used to share will dictate where it should ideally be held to best sysplex-enable the tasks making use of it. If the queue does not need to be recoverable, Shared Temporary Storage can be used. If the queue needs to be recoverable, it has to be held by a specific CICS region.

Having the resource owned by a specific region introduces an affinity and a single point of failure, but the risk of the region failing can be virtually eliminated by using a region used only to hold the queues (known as a Queue-Owning Region or QOR) or, less commonly, a region that owns various resources (Resource-Owning Region).

The recovery time, should this region fail, can be reduced by using Automatic Restart Manager (ARM). It can be further minimized, if the resources owned or application tasks are business-critical, by using the CICS Extended Recovery Facility (XRF) to have a standby region on another image.

Programming logic

Affinities created by program logic are the most difficult to detect, and frequently can only be detected by examining the application code or design.

Passing of addresses to other tasks should be avoided at all costs, or if forced by business requirements, they should be fully documented, detailing what programs the address is passed to, directly and indirectly.

SYSID should ideally not be used on commands at all. The routing of commands to resources should either be handled by definitions or by dynamic routing and workload balancing facilities such as CPSM. If SYSID must be specified, a more flexible approach would be to determine the required value dynamically, either through CPSM commands or another shared resource.

Programs that have evolved over time to do different things on different systems, either because of changing business needs or to deal with perceived shortfalls of the environment, can create problems when moving to a sysplex environment. Logic of doing certain things on system A and certain other things on system B works fine until system C is added; it could be a duplicate of A, or B, but which? Any such program should be split into separate programs for each function.

5.2.4 Affinity summary

Following is a list of types of affinity programming techniques.

Affinity-safe programming techniques

- ▶ Passing data between transactions using COMMAREA or using TCTUA - provided addresses are not passed in these areas
- ▶ Use of global ENQ and DEQ through ENQMODELS

Affinity-unsafe programming techniques

- ▶ Using long-life shared storage by GETMAIN SHARED, LOAD PROGRAM HOLD, CWA (unless it is used to hold read-only data that is replicated across relevant regions)
- ▶ Using task-lifetime storage by synchronized tasks
- ▶ Synchronization or serialization of tasks that use WAIT EVENT, WAIT EXTERNAL, WAITCICS
- ▶ Synchronization of tasks using local ENQ, DEQ commands

Affinity-suspect techniques

These techniques could cause affinities, depending on how they are used:

- ▶ Using temporary storage queues or transient data queues and trigger levels
- ▶ Synchronization and serialization of tasks
- ▶ Using INQUIRE and SET commands
- ▶ Using global user exits

5.2.5 What happens if you have to create an affinity?

Sometimes affinities simply cannot be avoided, usually for business reasons or because of the configuration of your system. In these cases you must document the affinity relationship as well as its scope and duration. Utilization of intelligent transaction routing can then be used to keep transactions that share the affinity resource in same CICS region.

5.3 CICS TS application checklist

Creating applications involves many considerations, and this summary list of “dos and don’ts” is intended to be only a guide. Your own site may have procedures or requirements that preclude some of these items, and may even add some.

- ▶ Design granular transactions.

Favor granularity. Rather than having a few transactions that consist of a “Control Program” that then links to other programs for specific tasks, you should implement a large number of unique transactions identifiers. This enables easier distinction between

transactions requiring different levels of service (for example, a background stock report task requires far less priority than a point of sale terminal).

Another benefit is that transactions that do not require database resources can be routed to regions not connected to DB2, rather than having all regions connected to DB2 just in case a transaction needs it.

- Pay attention to affinity-generating commands.

In addition to the information provided in section 5.2.1, “Affinities through use of CICS commands” on page 85, refer to *CICS Application Programming Guide*, SC34-5993.

- Do not pass memory addresses between programs or tasks.

Any storage shared between tasks should be only used for read-only information (such as tables with copies in all applicable regions), and the location in any region should be dynamically determined, starting with CICS commands accessing resources local to that region.

- Do not create application dependencies on specific CICS regions.

Applications relying on specific regions, either through their internal logic or through using hardcoded SYSID options on CICS commands, force unnecessary affinities. An example is printing: the printer may be owned by a specific CICS region, but rather than hardcode the system in the application requesting the print, a more flexible approach would be to use CPSM calls to determine which CICS owns the printer, or to find an available region in the “printer group.”

If the resource has to be owned by a single region, CICS definitions can direct the work to the required region, thereby removing the affinity between the application and the system configuration.

- Do not create bottlenecks.

Any resource that needs to be serialized impacts throughput (for example, when all tasks access a specific data record in a file), so you should investigate alternative solutions.

- Use z/OS Logger for merged output logs.

Whether these output logs are used for audit trails, messages logs, or other purposes in a sysplex environment, a single log with entries in the correct order is required. Although an RLS dataset could be used for this, using z/OS logger presents a better option: the record order will be ensured and there will be virtually no concerns about space, because logger moves data to DASD for migration by HSM, if required.

- Avoid forced order of execution.

Within an application, there may be a need for certain transactions to be executed in sequence. Starting the process in the order received is often desirable—but completing in the same sequence cannot be guaranteed in a sysplex environment.

- New versions of programs can be added in a phased manner.

For more detailed information on this topic, refer to 2.3, “Step-wise deployment/versioning” on page 27.

In a large sysplex environment, installing a new version of a program in all regions at the same time may simply not be possible, so a method of phasing the installation of new version is required.

Suppose you have a transaction where program A calls program B, which needs updating to add new function. The updates have all been made and tested with updates to both programs. The new program B should be installed in the environment first (having been updated), so that calls from the original program A will still work correctly.

An approach to this is to have a field, in any data passed by A to B, indicating its version. After new program B is in use in all regions, the new A can then be introduced. However, it must be introduced on all regions at the same time—otherwise, users will see any changes to the interface appearing and disappearing as their transactions are routed to regions with and without the new version.

So either the new version needs to be loaded by all regions at the same time through a SET PROGRAM PHASEIN (which could be issued by a tool such as CPSM), or it should be enabled by logic within the program (for instance, by checking the date to determine which interface to use).

- Check your data and database design.

Transactions will not execute sequential inserts into a non-partitioned database or VSAM ESDS. All such sequential operations should be distributed over partitioned ranges so that multiple concurrent operations may proceed at the same time, without locking the same page or VSAM CI.

Also, a unique sequential number should never be used as the primary key, as this will have tremendous contention on writing new records, time stamps, and so on.

IMS application considerations

The vast majority of IMS databases and application programs require no changes to run with data sharing; this is *toleration mode*. But just as Parallel Sysplex improves the availability and capacity of the architecture, there are also considerations for improving application availability; this is *exploitation mode*.

In this redbook, we separated the IMS Database (DB) topics from IMS Transaction Manager (TM) topics in order to make it easier for installations that have IMS DB without IMS Transaction Manager to find the relevant information.

In this chapter, we discuss the following topics:

- ▶ “IMS Transaction Manager overview” on page 96
- ▶ “IMS Database Manager” on page 103
- ▶ “IMS Transaction Manager” on page 119

6.1 IMS Transaction Manager overview

In this section, we take a look at IMS Transaction Manager. First we provide a high-level explanation of how the architecture runs, covering the following topics:

- ▶ How multiple transactions, jobs, and batch jobs that are executed on more than one IMS can make calls against a single copy of an IMS (or DB2) database
- ▶ How locks are maintained in the Coupling Facility with the use of lock structures
- ▶ How Parallel Sysplex makes facilities available to share storage in the cache structures

Then, based on your understanding of the architecture, we make recommendations about the following areas:

- ▶ Avoiding dependencies (affinities) on your application code, so you can process the code on any of the IMS TMs, thereby improving application availability
- ▶ Reviewing lock contention, so you can increase your ability to execute transactions in parallel
- ▶ Avoiding hot spots, which becomes more difficult to manage with increasing parallelism

We have summarized information here that is derived from the IBM Redbook *IMS in the Parallel Sysplex Volume I: Reviewing the IMSplex Technology*, SG24-6908; for more detailed information on this topic, refer to that publication.

6.1.1 IMS basics

The IMS logic flow is as follows: one terminal produces a message requesting a service from an application. The Data Communication Manager of the IMS control Region accepts the message, transferring it to the Message Manager.

There are four kinds of messages, but here we look at only the messages driven to application programs. Each message provides the name of the transaction to be executed. The transaction is associated to an execution class in the IMS generation. You can start one or more message processing regions (MPR) for each execution class. Each MPR runs in a separate address space, and is responsible for executing the transaction program that is related to the transaction in the IMS generation. So since you can have more than one MPR for each class of transaction, using parallelism is a common way of processing the requests.

Note that there are other ways of processing work units in IMS. For example, you can submit a job executing in a Batch Message Program (BMP), or you can route the message to specific types of MPRs, such as the fast path message processing program (IFP).

Figure 6-1 on page 97 shows the IMS basics.

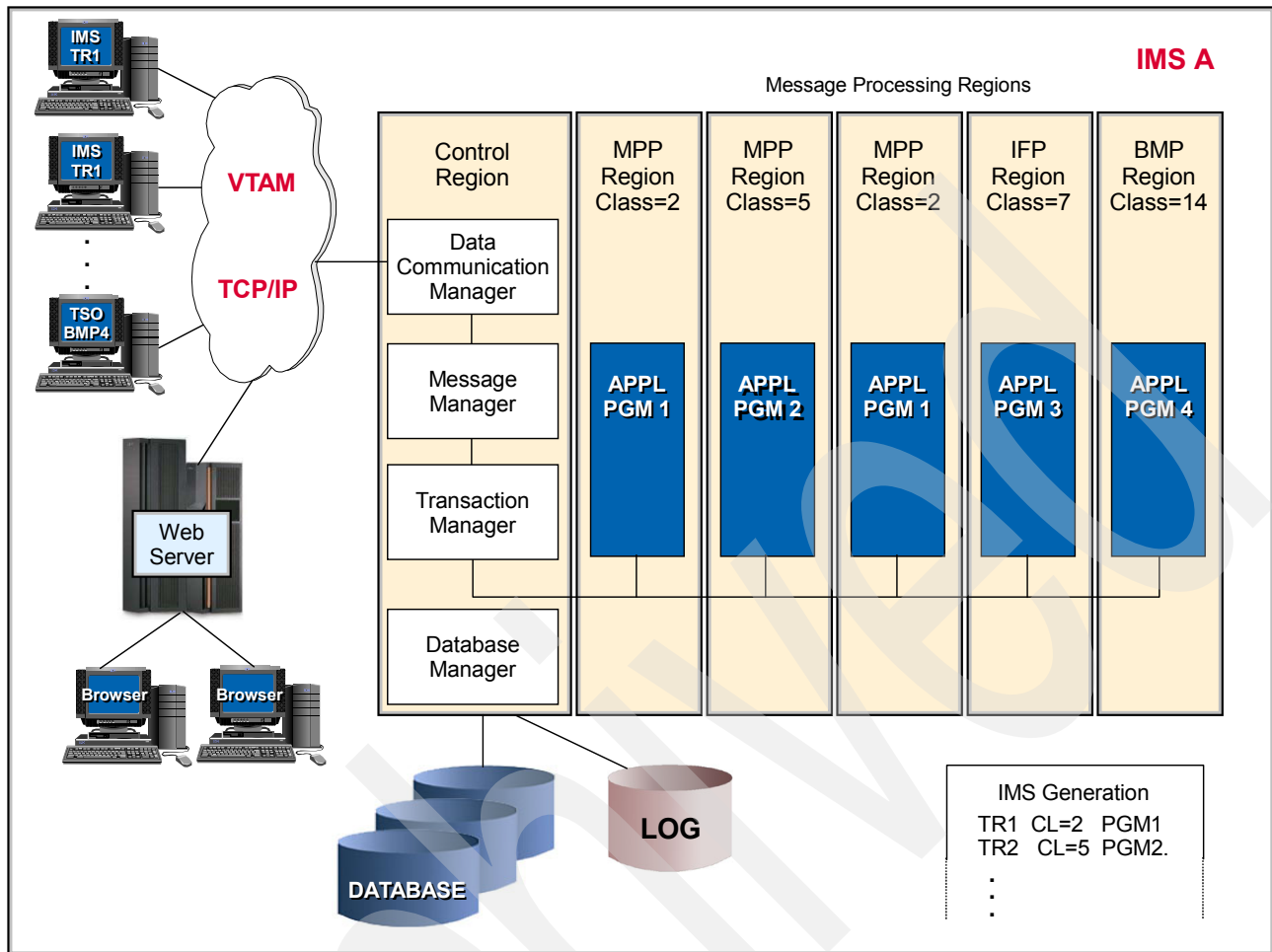


Figure 6-1 IMS basics

6.1.2 IMS in Parallel Sysplex architecture

A maximum of 255 IMS data sharing subsystems can simultaneously access and update a single set of IMS databases with full data integrity and good performance still guaranteed. These IMS subsystems, known as a *data sharing group*, can support any required combination of IMS TM, IMS DB control, and batch applications running in z/OS environments.

The data sharing group may consist of any combination of these IMS subsystems:

- ▶ IMS Transaction Manager control regions
- ▶ IMS Data Base Manager control regions
- ▶ IMS batch jobs

The components of IMS subsystems involved in Parallel Sysplex support are:

- ▶ IMS data sharing subsystems
- ▶ Coupling Facility structures contained in the Coupling Facility (CF)
- ▶ Sysplex services (XCF and XES) for data sharing control and communication
- ▶ The Database Recovery Control (DBRC) component
- ▶ The Database Recovery Control data set (RECON)
- ▶ Integrated Resource Lock Manager (IRLM)

Figure 6-2 illustrates the major IMS components involved in a Parallel Sysplex block level data sharing environments.

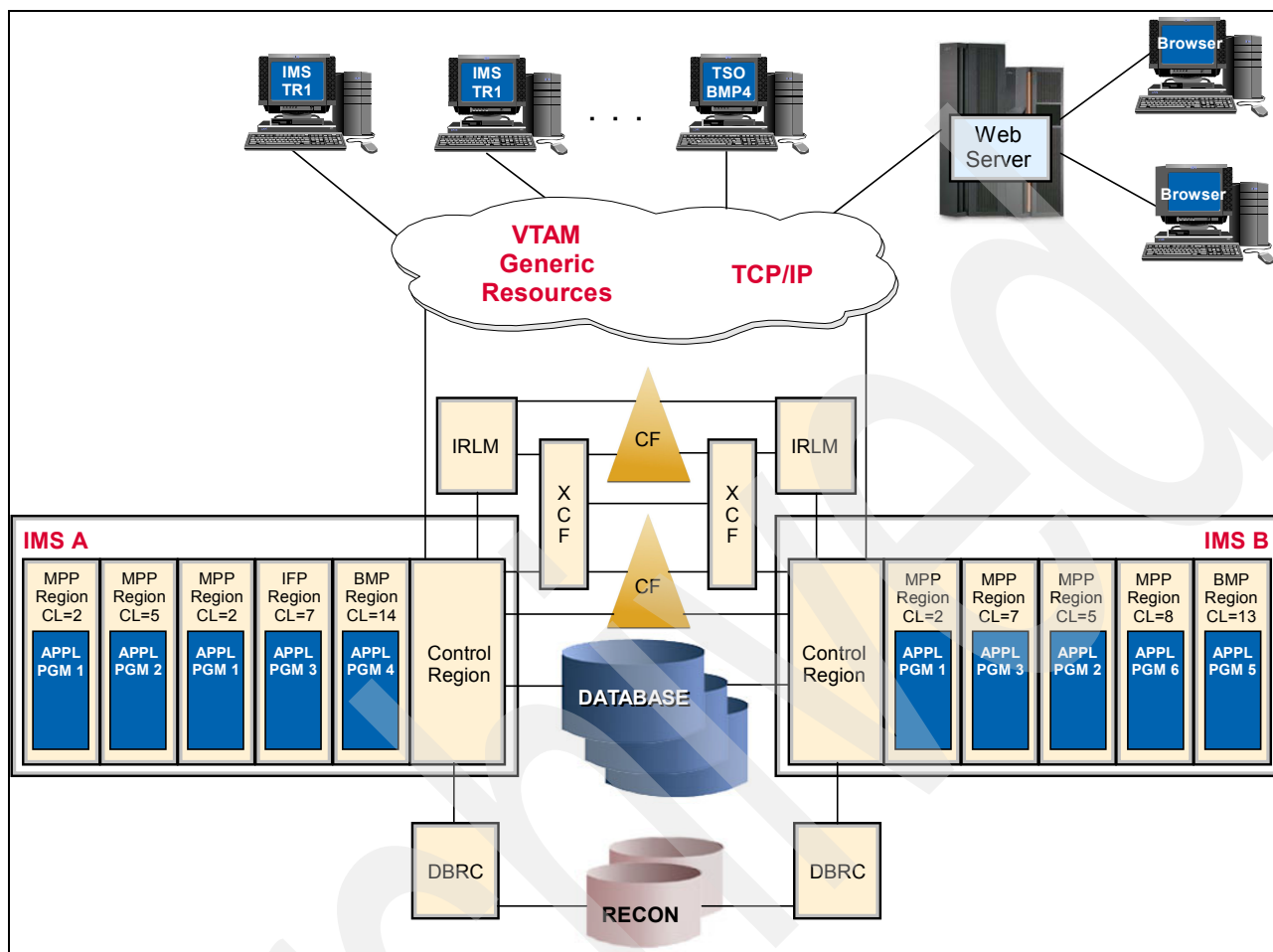


Figure 6-2 IMS in a Parallel Sysplex environment - an overview

Coupling Facility

The technology that assists in making high-performance sysplex data sharing possible is a combination of hardware and software products, collectively known as the Coupling Facility. The Coupling Facility is a microprocessor unit connected to z/OS processor complexes by high-bandwidth fiber optic links called *CF subchannels*. Coupling Facilities can also reside internally in the Central Processing Complex (CPC).

In a Coupling Facility, you allocate objects called *structures*. The storage is dynamically partitioned into structures of different types. Each type of structure provides a unique set of functions and offers different ways of using a Coupling Facility. The types of structures are:

- ▶ Lock structure
- ▶ Cache structure
- ▶ List structure

Sysplex services for data sharing

Cross-system Coupling Facility (XCF) is necessary in a block level data sharing environment to provide group, signaling, and status monitoring services to z/OS components and authorized applications.

Cross-system Extended Services® (XES) allows multiple instances of an authorized application or subsystem, running on different systems in a sysplex, to implement high-performance, high-availability data sharing by using a Coupling Facility. Features of the different structures, available through the use of XES, include the ability to:

- ▶ Implement efficient, customized locking protocols, with user-defined lock states and contention management (lock structure).
- ▶ Determine whether a local copy of cached data is invalid. Automatically notify other users when a data update invalidates their local copies of cached data (cache structure).
- ▶ Share data organized as a set of lists (list structure).

IMS block level data sharing use of Coupling Facility structures

IMS uses two different types of structures in the Coupling Facility for data sharing support:

- ▶ Lock structure

The IRLM lock structure is used by IRLM to serialize updates to the shared IMS database resources. The IRLMs manage data sharing global locks with the help of the Coupling Facility, which makes consistently high locking performance possible regardless of the number of sharing subsystems.

- ▶ Cache structures:

- OSAM directory only buffer invalidate (XI) structure

This cache structure is used by IMS data sharing partners to check whether an accessed buffer is currently valid. When an IMS subsystem makes a change to a buffer, each of the data sharing IMS partners that have accessed that buffered block has flags set to invalid. Whenever an IMS partner attempts to access that buffer it is found to be invalid, and that IMS must read the buffer from DASD. That particular flag is then marked as valid.

- VSAM directory only buffers invalidate (XI) structure

This cache structure performs the same task as the OSAM Buffer XI structure, but for VSAM buffered control intervals.

- SVSO data entry database (DEDB) AREA store-in structure

In order to perform block level data sharing or Virtual Storage Option (SVSO) DEDB AREAs so that multiple IMS subsystems can read and update the data concurrently, IMS uses a cache structure in the Coupling Facility.

6.1.3 Related improvements

Here are IMS facilities and improvements you should know about. Although they are not all related to Parallel Sysplex, they may be of interest when dealing with applications that have 24/7 requirements.

Open Database Access (ODBA)

Open Database Access provides a callable DL/I interface from application execution environments outside the scope of IMS control, such as DB2 Stored Procedures, WebSphere Enterprise Java Beans (EJBs), and CICS Java programs. Open Database Access is not needed for IMS-controlled regions such as MPRs, BMPs or IFPs, for calls to locally controlled databases. The application execution region, which may be any address space in a z/OS image, connects through a database resource adapter (DRA) interface to DBCTL or IMS TM/DB subsystems.

ODBA can be exploited in order to have a much more direct way of retrieving data from the shared databases in a Parallel Sysplex cluster.

Online Change for DBs

Adding, deleting, and changing of Full Function or DEDB databases and AREAs is fully supported. This allows changes to be made without bringing down IMS.

Beginning in IMS V8, IMS supports a Global Coordinated Online Change function. When IMS systems are running in a Parallel Sysplex, IMS can coordinate the online change across all of the IMS systems. This ensures that each phase of the online change is coordinated between all of the IMS systems and the systems are kept in sync.

When you want to bring changes online globally in all IMSs in an IMSplex using Global Online Change Command Sequence, you must make active the libraries that contain the changes to be made. Issue the following sequence of commands:

```
INITIATE OLC PHASE(PREPARE)
/DISPLAY MODIFY
INITIATE OLC PHASE(COMMIT)
```

Note that you might have to issue additional commands after INITIATE OLC to display and resolve work in progress that might cause the commit to fail.

With DB structural changes, extra work may be required in order to unload the data, change the structure of the data, change the DB, and reload the modified data. The process may be long and it can still have an impact on 24x7 applications. However, Online Change for DBs can improve the availability of a change, as it can be prepared at a time that is best for the application.

For additional information about modifications to IMS DB, refer to *IMS Version 8: Administration Guide: System*, SC27-1284.

/STOP REGION ABDUMP command enhancements

In prior releases, completion the /STOP REGION ABDUMP command was held if the dependent region to be aborted was waiting for an IRLM lock. If the wait for the command to execute is excessive, the operator wanting to terminate the dependent region can issue another form of the command, /STOP REGION CANCEL, to force the region to terminate. However, this form of the /STOP command causes the entire IMS online system to abnormally terminate if the dependent region was waiting on a lock.

With IMS V8, the /STOP has been enhanced to detect whether the region which is to be stopped is waiting on a lock request within the IRLM. If it is waiting on a lock when the command is issued, the waiting dependent region task is resumed and forced to return to its dependent region address space, where it is abended (U0474).

Shared sequential dependent (SDEP) segment

Use of the Fast Path DEDB sequential dependent (SDEP) segment function provides the user with a time-sequenced insert capability for a portion of a DEDB AREA that has SDEPs defined. The SDEP buffer is kept in main storage and is written by an output thread during Fast Path synchronization point processing after it is filled to capacity. At the same time, the next SDEP buffer is obtained on behalf of the subsequent CI.

Each IMS subsystem image in a sysplex data sharing environment is known as an *IMS partner*, and each IMS partner now manages its own SDEP CIs. The SDEP AREA is shared for read access, but the individual CIs are not shared for insert. A timestamp has been added to the end of each SDEP segment to allow SDEPs to be shared without changing time-of-insert processing. No Coupling Facility structures are used for shared SDEP support.

Image Copy 2 utility

Image Copy 2 utility (DFSUDMT0) is a new form of image copy that produces an image copy with high performance with no or minimal database data set outage. The fuzzy image copy option allows the database data set that is to be copied to remain fully accessible by the online system(s). A clean image copy does require a short outage (a few seconds). Image Copy 2 utility utilizes SMS concurrent copy and supports all types of database data set organizations (KSDS, ESDS, and OSAM) used by IMS.

For information about enhancements of this utility in IMS V7 and IMS V8, refer to *IMS in the Parallel Sysplex Volume I: Reviewing the IMSplex Technology*, SG24-6908.

Consider using this utility for applications with 24x7 objectives for increased availability.

Fast Database Recovery (FDBR)

FDBR is a facility that addresses a data sharing problem known as the retained locks situation.

Note: Following is a description of the differences between active locks and retained locks:

- Active locks

When a member is active, the locks that it holds are called *active locks*. For transaction locks (L-locks), the normal concurrency mechanisms apply, including suspensions and timeouts when incompatible locks are requested for a resource. For physical locks (P-locks), DB2 uses a negotiation process to control access.

- Retained locks

The particular concern for availability is what happens to locks when a member fails. For data sharing, active locks used to control updates to data become retained in the event of a failure and are then known as *retained locks*; information about those locks is stored in the Coupling Facility until the locks are released during restart. Retained locks are necessary in order to protect data that is in the process of being updated from being accessed by another active member of the group.

At the time a data sharing subsystem fails, it might hold non-sharable locks on database resources that cannot be released until the retained locks are released. Retained locks are usually released by dynamic backout during emergency restart of the failed IMS. The execution of a successful emergency restart typically takes a few minutes, but it can take much longer depending upon the type of failure and nature of the emergency restart.

Even though a data sharing IMS fails, the other data sharing partners can still be executing. If application programs executing on the remaining IMS subsystems try to access a database resource that is protected by a retained lock associated with this failed IMS partner, the program is abended with a U3303 abend code, and the input transaction is placed on the suspend queue.

If these access attempts to retained locks result in many U3303 abends, then the impact on the application programs executing on the surviving IMSs can be severe (that is, application programs are unable to get their work done, and terminal operators might be hung in response mode).

Application program failures related to the access of resources held by retained locks can be avoided or minimized in two ways:

- ▶ Application programs can be changed to issue an INIT STATUS GROUPA call at initialization and to check for BA and BB status codes associated with unavailable databases. If these status codes are returned, the application can decide to terminate or take another processing path. These application changes eliminate the U3303 abend when a database call attempts to access a resource protected by a retained lock.
- ▶ Fast Database Recovery (FDBR) regions provide a second way to address the retained lock problem by dynamically backing out in-flight units of work whenever an IMS subsystem in a data sharing group fails while holding retained locks. FDBR minimizes the impact of the problem by reducing the amount of time that retained locks are held. Work with your system programmer to set up an FDBR region.

BMP scheduling flexibility

Some IMS customers need to have a single BMP proclib member that is used for starting BMPs across the Parallel Sysplex so as to be able to run a BMP on any active IMS system in the sysplex. They can run IMS control regions anywhere and allow all BMP jobs to be floated anywhere with assurance of success.

No changes to the BMP JCL IMSID parameter are required when a BMP is executed under different control regions.

To use this option, an IMS startup parameter, IMSGROUP, is added to the control region procedure.

- ▶ IMSGROUP - IMS group name (no default value). This parameter is user-generated and is usually the IMSID value most often found in the BMP or other dependent region proclib members. The name can be one to four alphanumeric characters in length.
- ▶ The IMSGROUP= parameter must equal the same value as the IMSID of the BMP regions.
- ▶ The IMSID of the CTL region should be different as it must be unique in the sysplex. Using the IMSID of the BMP regions as the IMSGROUP= value in the CTL region eliminates the need to add any new parameters to the BMP (and other dependent region's) JCL.

Note the following example:

```
CTL Region 1 IMSID=IMS1,  IMSGROUP=IMSB
CTL Region 2 IMSID=IMS2,  IMSGROUP=IMSB
BMP Region 1 IMSID=IMSB
MPP Region 1 IMSID=IMSB
```

In this example, BMP region 1 will connect to either IMS 1 or 2. The IMSID of the BMP JCL is used to construct the z/OS name. This example also holds true for the MPP region and any IFP regions. There is no change to dependent region JCL required. The one EXEC parameter, IMSGROUP= is added to the control region procedure.

High availability large database

A major enhancement of IMS V7 is the high availability large database (HALDB). HALDB has been designed for full block level data sharing operation. HALDB allows DL/I databases to grow much larger than the previously supported 4 GB for VSAM and 8 GB for OSAM.

A HALDB is a partitioned DL/I database. Partitioning a database allows the use of smaller elements that are easier to manage and multiple partitions decrease the amount of unavailable data if a partition fails or is taken offline.

HALDB allows the grouping of DL/I database records into sets of partitions that are treated as a single database while permitting functions to be performed independently for each partition. Each HALDB partition has the same capacity limit as a DL/I non-HALDB database. Like a DL/I non-HALDB database, each partition may consist of up to 10 data sets; however, the number of data sets selected will then apply to all the partitions in that HALDB. This allows a large amount of data to be contained in a single partition.

HALDB provides the following benefits:

- ▶ Capacity increase - each partition can be equivalent in size to one DL/I non-HALDB database, and one HALDB can support up to 1001 partitions.
- ▶ Availability increase - partition independence leaves more data available because a particular partition can be taken offline while the rest of the partitions in the HALDB remain available.
- ▶ Better manageability - smaller sections of the database are easier to manage.
- ▶ Usability - reorganized databases can be immediately usable after image copies have been created because neither Prefix Resolution or Prefix Update utility processing is required.

6.2 IMS Database Manager

No changes in IMS application code are specifically required to coexist in an IMS data sharing environment, or to use IMS data sharing. Some changes in database design or application code may, however, be required to exploit data sharing in the most efficient manner.

6.2.1 Supported IMS databases

All IMS database types are supported for data sharing except the main storage databases (MSDB), GSAM, and SHISAM.

GSAM cannot be shared. SHISAM can be shared as long as it being used only by IMS applications. If non-IMS applications are accessing this VSAM KSDS, then sharing cannot be done.

From IMS Version 6, you have two database options available for data sharing:

- ▶ Data entry data bases with virtual storage option (DEDB SVSO). You must consider converting MSDBs to DEDB SVSO databases
Depending of your actual needs for the database, you can consider also converting them to a standard DEDB or to an Hierarchical Direct Access Method database (HDAM), which should probably be the closest type of full function database.
- ▶ Fast Path DEDB sequential dependent segments.

Fast Path DEDB SVSO

The Fast Path Virtual Storage Option allows you to map data into virtual storage (into a z/OS data space). You can map one or more DEDB Areas into virtual storage by defining them as SVSO Areas with DBRC commands.

SVSO does not use a dataspace. Data from the DEDB area is maintained in storage in a private buffer pool associated with the structure. These buffers are in ECSA. The SVSO data also resides in the CF structure. Only sharelevel 0/1 VSO DEDBs are mapped in a dataspace.

When a CI is read for the first time for a local VSO (shrlevel(0/1)) area, the CI is read into a common buffer and copied into a z/OS data space. Subsequent access is from the dataspace.

When a CI is read for the first time for a SVSO (shrlevel(2/3) area, the CI is read into a private buffer pool buffer and written to the CF structure. Subsequent access is either from the buffer pool (if the lookaside option is selected) or from the CF structure.

DEDB VSO have sharelevel 1 and are defined in a dataspace only in one system image. The recommendation is not to use DEDB VSO in a Parallel Sysplex, as you are creating an affinity since dataspace do not reside in the Coupling Facility.

DEDB SVSO have sharelevel 3 and are defined in the recon with CFSTR1 and CFSTR2 (optional) parameters that relate to structures in the Coupling Facility. Therefore, the data can be shared by any of the IMS system images. The use of CFSTR2 makes sense if you have a second Coupling Facility for availability, which is the recommended configuration.

You can find more information in the section “Fast Path DEDB SVSO sharing” in the IBM Redbook *Parallel Sysplex Volume I: Reviewing the IMSplex Technology*, SG24-6908.

Fast Path DEDB sequential dependent segments sharing

DEDBs have a special type of dependent child segment called a *sequential dependent* (SDEP). The SDEP segment type is defined within a DEDB by coding the TYPE=SEQ parameter on the SEGM statement in the DBD macro. SDEP segments are not stored near the root segment or other dependent segment types. Instead, they are stored in SDEP CIs at the end of each AREA data set (ADS).

When an application issues an ISRT call for a SDEP, the segment is saved in a dependent region's normal buffer allocation (NBA) buffer until it is committed, at which time it is moved into the current sequential dependent buffer (CSDB). As each application program reaches commit, it acquires a lock on this CSDB and moves its SDEPs to the next available position. When the CSDB is full, it is written to the ADS and another buffer is allocated and filled.

The net effect is that SDEPs are stored in the ADS in the time sequence in which they were inserted, without regard to which root segment they were stored under (hence the name *sequential dependent segment*).

The SDEP Scan utility (DBFUMSC0) is used to retrieve SDEPs in time sequence, and the SDEP Delete utility (DBFUMDL0) is used to delete SDEPs after they have been scanned. The CSDB buffers are not written until they are full, so there is no way to let two or more IMSs put committed SDEPs in the same buffer. Putting them in different buffers would destroy the sequentiality of the segments. (This is why SDEP data sharing was not supported until IMS V6.)

IMS V6 shared SDEP enhancement

IMS Version 6 solved the SDEP sharing problem through a combination of IMS internal processing changes, and SDEP CI and segment format changes. Although the SDEP segments are no longer physically in sequence, they now contain a commit timestamp (the 8-byte STCK value at commit time) which optionally enables them to be sorted as part of the scan process and presented to the end user in chronological sequence, the same as when they were not shared.

The changes are completely transparent to the application program. However, there are some changes to the SDEP utilities that you should be aware of:

- Processing changes

SDEP CIs are no longer allocated one at a time. Instead, each sharing IMS is preallocated a set of SDEP CIs which it proceeds to fill. The number allocated for each request is dependent on how fast that IMS is filling them up. The CI preallocate process ensures that changes to the CI are logged and that control CI is hardened before an IMS member begins to use any more preallocated CIs.

Note that this allocation method means that each IMS inserts SDEPs in sequence within its own set of allocated CIs, but that when combined with the CIs allocated to other IMSs, the overall effect is that they are not physically in chronological sequence.

- SDEP segment format change

An eight-byte time stamp is added to the end of each SDEP segment. This time stamp denotes the time that segment was committed. Since each IMS in the data sharing group is using a time synchronized by the external time reference (Sysplex Timer), these time stamps are consistent across all IMSs and can be used by the SDEP Scan utility to sort the SDEPs after scanning them and before writing them to the output file.

The net effect is that they appear to the end user to be in chronological sequence, just as they were when not shared.

- SDEP CI format changes

There have also been changes to the format of the SDEP CI itself. There are new flags and the IMSID of the IMS system which filled that CI. In all, there are an additional eight bytes added to the SDEP CI.

- Use of the IRLM

Prior to IMS V6, the only locking came in the form of an internal CSDB latch which serialized the inserting of committed SDEPs into the CSDB by different dependent regions.

Starting with IMS V6, and when SDEPs are shared, you may see an IRLM lock on the CI currently in the CSDB. The purpose of this lock is used by IMS to determine whether any particular CI is among those in its set of pre-allocated CIs. This in turn is used when a program running in one IMS needs access to a segment in a CI that is still in another IMS's buffer pool.

- SDEP scan and delete utilities

Prior to IMS V6, the scan and delete utilities worked solely on RBA boundaries.

With IMS V 6 and the support for data sharing, SDEP segments may not be physically in chronological sequence. The utilities now work on a time stamp basis. For example, the SDEP Scan utility may be directed to scan all segments inserted between 8:00 AM and 10:00 AM.

In a data sharing environment, it is possible that some segments inserted at 10:01 AM may physically be interspersed among those between the two specified times. The SDEP Scan utility will ignore these SDEPs and return only those between the desired times. There are a number of new control card statements made in IMS V6.

For detailed information about these changes, refer to *IMS Version 8: Utilities Reference: Database and Transaction Manager*, SC27-1308.

- Transparent application program support

Although the SDEP segment itself has been increased by eight bytes, these eight bytes are added to the segment by Fast Path when it is inserted, and removed by Fast Path when the segment is retrieved either by an application program or by the scan utility. No application changes are required.

- No changes to DEDB AREA root and direct dependent segment sharing
- No effect on XRF backup and takeover

Backup tracking and takeover restart open processing are not affected in their function by this enhancement.

Consider adding the QUITCI parameter in DEDB SDEP Scan process.

```
TYPE SCAN
ERRORACTION SCAN
AREA aaaaaaaa
QUITCI
GO
```

QUITCI specifies that all IMS partners will give up their current SDEP CI and any preallocated CI RBAs immediately. This option is useful if there is an IMS partner that processes a low volume of transactions and therefore takes an extended period of time to fill its preallocated CIs.

The QUITCI parameter does the following:

- It forces all in-use but only partially-filled CIs to be written to the ADS and then released, which in turn enables the delete utility to process the entire CI and advance the logical beginning (DMACXVAL) as far as possible, ensuring that sufficient space is available when the SDEPs cycle around.
- It guarantees that no new SDEPs will be inserted in the QUITCIs. The next utility run can therefore start from the CI following the last UHWM, assuming that the default STOP is used.

6.2.2 Data sharing integrity overview

Database integrity is maintenance of the correct internal database information (such as pointers between segments), and the maintenance of the user data stored in the database. It uses database locks to protect the database while making it available to multiple application programs concurrently. Integrity includes the isolation of effects of concurrent updates to a database by two or more application programs.

To maintain integrity, database updates must occur only on a transaction consistent basis, meaning that:

- None of the effects of a transaction can be visible to other programs until the transaction reaches a commit point. The data items that will be affected by updates are inaccessible (locked) during the updating process.
- All the effects of a transaction (multiple updates to multiple databases) are made available to other applications and programs after the commit point for that transaction if the transaction is successful, or
- If the transaction fails and is abended or rolled back, all updates are removed by the backout processes.

With block level data sharing, this definition has been extended to include protection of data in situations where updates occur on multiple processors in a Parallel Sysplex environment.

6.2.3 Locking recommendations

Most customers do not have problems with locking. Even so, some of the areas that may require application changes in a Parallel Sysplex implementation have to do with the performance impact of database locking. An application that is performing adequately in a single IMS system may not perform acceptably when the database is shared.

The following section outlines some examples of application designs that might need modification.

Small databases with few records

Frequent updating of the same segment or control interval by many IMS programs increases lock contention, and may effectively eliminate the possibility of running multiple copies of the application. (This is sometimes referred to as a database hot spot.) An example of this type of database could be a control record database where the same record is updated for every transaction.

This was never a good application design, as it inhibits parallel operation of the transactions even on a single IMS system, but the performance may have been acceptable, especially if the database was an MSDB. You have then two possibilities to follow: review the application design to eliminate the bottleneck, or use a DEDB SVSO.

Databases with ascending or descending keys

Databases with ascending or descending keys can cause problems in two ways:

1. The databases often require a control segment to keep track of the last key that was assigned. This can result in the contention problem described in the previous section, "Small databases with few records."
2. The new segment may be inserted next to the last one, in the same CI. This can result in contention for the same block or CI among all users.

Examples of applications that may use this type of database are:

- ▶ Invoice applications which assign the next invoice number by incrementing a counter.
- ▶ Audit applications that produce an audit trail using the current date and time for the key of the next segment. These keys will be in ascending order.

The application may have to be changed to remove the contention when the databases are shared.

Locking recommendations

Most locking problems can be addressed using the following IMS DB tuning techniques:

Avoid long locks: Avoid holding locks for long periods. For example, if a single call is made against PCB1 and then subsequent processing occurs on other PCBs, a "long lock" on the PCB1 resource will occur. It would be advisable to release the lock quickly by issuing a second call against PCB1 to a non-existent and/or high-key target.

Note: A program specification block (PSB) is a collection of program control blocks. A program control block (PCB) defines the view of the databases that an application program can access.

Do updates as late as possible: Execute updates closer to the end of the program, because you can then reduce the time of the contentions. For example, suppose you need to update a total field in a group of parents reading data of their corresponding children. If you know more less the number of parents you are dealing with, you can define a table in your program to store the calculations and do the updates after you have read all the children.

Reduce the number of DL/I calls: Locking rates and contention are directly related to the volume of concurrent, conflicting DL/I requests for shared resources, so one way to provide a solution to locking problems is to reduce the number of DL/I calls in the following ways:

- ▶ The use of path calls can retrieve more than one database segment per call.
- ▶ Programs should be designed to avoid issuing multiple calls for the same segment during a single execution.
- ▶ If possible, an attempt should be made to eliminate redundant or nonproductive calls that result in GE, GB, or II status codes, except where these status codes represent exceptional conditions.

Note: These status codes have the following meanings:

- ▶ A GB status code can be returned for an unqualified GN call or a qualified GN call without a maximum key (if no data is returned to the I/O area).
- ▶ A GE status code means that DL/I is unable to find a segment that satisfies the segment described in a Get command, or for an ISRT command, it means that DL/I cannot find one of the parents of the segment you are inserting. Another possibility is that the program issued a STAT3 command for OSAM or VSAM buffer pool statistics, but the buffer pool does not exist.
- ▶ A II status code means that the program issued an ISRT call that tried to insert a segment that already exists in the database.

- ▶ In order to reduce the large number of unneeded segments retrieved via unqualified GN calls, use fully qualified GU calls.

Examine and resolve deadlocks: Deadlocks sometimes arise because of poor application or database design. More than a small number per day should be investigated. They indicate that the application may not perform well in a shared data Parallel Sysplex environment.

One of the deadlocked programs is pseudo-abended with code ABENDU0777 and this event is logged as type 67FF records.

Deadlocks are reported in IMSPARS reports in the Management Exception Report in the section headed "ERROR CONDITIONS." If the IMS Monitor is running at the time, more detail can be obtained from it. The IMS DFSERA10 File Select and Formatting Print utility with DFSERA30 as the specified exit provides a comprehensive Deadlock Report for each event.

A short-term solution for transactions that are suffering these deadlocks is to funnel these conflicting transactions through a single dependent region via class-controlled scheduling. The ultimate solution might have to be the redesign of the applications to issue more streamlined DL/I call flow to conflicting databases.

After reading a record with integrity, issue a GU call for a non-existing root: Then you will release the lock. You can use the X'FF' key. You may still have some contention in the X'FF' position, but it will be reduced.

Use of INIT STATUS GROUPB: When the INIT call is issued with the character string STATUS GROUPB in the I/O area, the application program informs IMS that it is prepared to accept status codes regarding data unavailability and deadlock occurrences. The status codes for data unavailability are BA and BB. The use of this INIT call provides for a more effective interface to the application when deadlock conditions arise, but the original conditions should still be examined.

Use two PCBs if there is little updating: If an application does relatively little updating of a shared database, you could replace a single update PCB with a read or read-only PCB and another update PCB. The majority of application processing could reside with the read PCB and when an update is required, the data could be re-read and then updated via the update PROCOPT PCB.

Do not merge different types of accesses: When you have two PCBs for the same database, you should not merge read accesses with update accesses. That can help you to take fewer locks, or at least shorten them.

Checkpoint frequency: The interval between BMP checkpoints should be a small number of seconds at most, and the IMS facilities to suppress the MTO message for checkpoints should be exercised. Refer to “Batch considerations” on page 113 for more detailed information on BMP checkpoints.

Use call sequences that make good use of the physical placement of the data: Access segments in hierarchical sequence whenever possible, and avoid moving backward in the hierarchy. This is how IMS attempts to keep the segments, so you can obtain better performance. In one physical read, you can obtain several of the segments you need. Also make sure the DB is well organized; if necessary, plan for frequent DB reorganizations.

Process database records in order of the key field of the root segments: This will enable you to obtain better performance if you have sequential batch processes, or even in transactions that read some portions of the data.

For HDAM and PHDAM and Fast Path databases, this order depends on the randomizing routine that is used. Your DBA can define an special randomizing module that stores the segments in sequence order, which can be used for databases where you know the input data, and have many processes that must read it in sequence.

However, this approach is not useful for databases with concurrent updates on similar keys; the standard randomize is usually good enough for the majority of cases. Use this option only in special cases.

Review your program and calls structure: Avoid constructing the logic of the program and the structure of the commands or calls in a way that depends heavily on the database structure. Depending on the current structure of the hierarchy reduces the program's flexibility.

Examine transactions with MODE=MULT: As mentioned in “General IMS TM application performance considerations” on page 123, if MODE=MULT (which is the default) is specified on the TRANSACT macro, the database buffers are written at program termination and therefore tend to tie up database blocks and buffers longer than potentially necessary. Review all transactions with that designation and change to MODE=SNGL if possible.

Use PROCOPT effectively: The best way to reduce the overhead of lock contention is to take fewer locks. A share lock does not cause contention between sharing users; only update locks cause contention, and this is where the correct choice of the PROCOPT value is important.

IMS programs obtain exclusive access to an IMS database with the IMS PCB option PROCOPT=E. IMS acquires locks with PROCOPT=E in an IMS data sharing environment. Waits and deadlocks may occur in other members of the IMS data sharing group if they attempt to access a database controlled with this option in another IMS subsystem.

Without data sharing, no IMS locks are obtained, since no other application had access to the database at the same time. An example of such an application is an update BMP that runs overnight when there are no online or other IMS users around to access the database.

Database tuning - block/CI: The fewer segments that fit together into a block or CI, the less contention there will be between different programs for different segments in the same block. Clearly this is most likely to be a practical consideration for a “table database” where there might be many roots segments in each block or CI. Techniques to reduce the number of segments that fit into a block or CI include:

- ▶ Reduce the size of the block/CI

The smaller the CI, the fewer segments that will fit into it. However, there are limits to the size of a CI (4 KB for DEDBs), and buffer pools may have to be adjusted to reflect the new size without wasting space in the buffer pool.

Also, you need to strike a balance between more I/O activity with smaller block/CI sizes, and more lock contention.

- ▶ Increase the amount of free space within the CI

With less space for actual data, fewer segments will fit into the CI.

- ▶ Application change: change the key

In some cases, the application may have to be rewritten to produce an additional, more random, key. This could apply to situations where the database has a small number of large roots. In a single system, several programs can update different segments of a database record “almost concurrently” but this is not possible across multiple block level database sharing IMS subsystems.

Frequently reorganize KSDS structures: In a data sharing environment, IMS does not reclaim index space for a delete of an indexed segment because of integrity during backouts. The result of this is that after mass deletes of records with keys, you have many I/Os unless consecutive calls to HIDAM root segments with PTR=TB specified are executed. The many I/Os occur because the index component sequence set entries still point to empty data component CIs, and these empty CIs will be read until the target record is found.

Two recommendations for mass deletes in KSDS are:

- ▶ Frequent reorganizations (REPRO) to eliminate the empty (unreclaimed) CIs.
- ▶ Use enough database buffers to keep the CIs in the empty range within the buffer pool. This technique will not avoid the scan, but it will reduce the number of I/Os.

Avoid using empty HIDAM databases: If a program is scheduled to insert into an empty HIDAM database, the inserts will use the last block, thus causing block contentions and buffer invalidates for sharing systems. Therefore, it is wise to place a selection of roots scattered around the database before doing mass inserts.

Avoid the contention of HIDAM PTB root pointers: Whenever a segment is pointed to by a pointer in a different block/CI, there is the potential for two (or more) blocks to be locked as the result of an ISRT or DLET. If HIDAM roots using PTB pointers (twin backward and forward pointing) are inserted, it is possible for three blocks to be updated and locked:

- ▶ The data block for the newly inserted root
- ▶ The block containing the previous root via its PTF pointer

- The block containing the next root via its PTB pointer

This HIDAM option is usually chosen to enable key sequential processing to occur without reference to the index. If this process is observed as adding to locking problems, consider using PTF only, or no twin pointer on the root.

Minimize access to HDAM/HIDAM space bitmaps: When a bitmap in HDAM or HIDAM databases is updated, a global lock is taken on the bitmap block/CI. Therefore, all access to a bitmap is serialized across multiple systems. With the HD space search algorithm, when the bitmap is referenced, the block/CI is also locked.

Therefore, for volatile databases, minimizing the chance of needing the use of a bitmap should be considered, by reducing the chance of filling any block or executing the HD space search algorithm. Consider doing HIDAM reorganizations more frequently, with more generous amounts of free space. For HDAM structures, reorganize with larger Root Addressable Area (RAA®) and/or a smaller BYTES parameter to force more mass-inserted segments into overflow.

Read without integrity

Each IMS batch or online instance has OSAM and VSAM buffer pools defined for it. Without locking to serialize concurrent updates that are occurring in another IMS instance, a read without integrity from a database data set fetches a copy of a block or CI into the buffer pool in storage.

Blocks or CIs in the buffer pool can remain there a long time. Subsequent read without integrity of other blocks or CIs can then fetch more recent data. Data hierarchies and other data relationships between these different blocks or CIs can be inconsistent (for more information on this topic, refer to *IMS Application Programming Design Guide*, SC26-9423).

Note the following:

- The deletion of a dependent segment and the insertion of the same segment type under a different root, placed in the same physical location as the deleted segment, can cause simple Get Next processing to give the appearance of only one root in the database. The next and subsequent Get Next calls retrieve segments from the other root.
- Data Set Extensions: IMS instances with database-level sharing can open a database for read without integrity. After the database is opened, another program that is updating that database can make changes to the data. These changes might result in logical and physical extensions to the database data set. Because the read without integrity program is not aware of these extensions, problems with the RBA (beyond end-of-data) can occur.
- If, in the moment of reading, another program that had modified the data falls down and rolls back, then the data you have read is not correct.

PROCOPT=G causes share locks to be acquired while PROCOPT=R,I,D or A causes exclusive locks to be taken. If the processing is truly read-only, specify the PCB processing option PROCOPT=G, or even better, PROCOPT=GON (“dirty read”), rather than the default of PROCOPT=A (all read-write processing). This reduces the amount of locking that IMS performs. Let’s take a closer look at these options:

- Procopt GO

Programs using PROCOPT=GO must be prepared to handle additional IMS abends or status (return) codes that may result from an unsuccessful read request. Without PROCOPT=GO, IMS will obtain a lock on the requested segment. IMS will return to the program only when the segment is available.

However, with PROCOPT=GO, read-only programs can reference data being updated by other programs. When this happens, invalid pointers to the data might be used to perform the read.

Normally, if an invalid pointer is detected during IMS processing, the application is abended with IMS user abend U08xx for full function databases and U1026 for Fast Path DEDBs. This happens also in case of segment compression when the returned length is erroneous.

- **Procopt GON**

If you specify PROCOPT=GON, IMS returns status code GG instead of an abend. The read-only program can then recover from the error of invalid pointers, but you still have the problem of possible erroneous data read.

PROCOPT=GON will use no locks.

- **Procopt GOT**

With GOT, if the segment you are retrieving contains an invalid pointer, the response from an application program depends on whether the program is accessing a full-function or Fast Path database:

- For calls to full-function databases, the T option causes DL/I to automatically retry the operation. You can retrieve the updated segment, but only if the updating program has reached a commit point or has had its updates backed out since you last tried to retrieve the segment. If the retry fails, a GG status code is returned to your program. When a bad pointer is encountered with GOT, IMS will invalidate all shared data in the buffer subpool.
- For calls to Fast Path DEDBs, the T option does not cause DL/I to retry the operation. A GG status code is returned, so it works as GON.

In both cases, you have still the problem of possible erroneous data read.

With this option you can still have some performance degradation. The T option forces a re-read from DASD, negating the advantage of very large buffer pools and VSAM hiperspace for all currently applications and shared data. For more information on the GOx processing option for DEDBs, refer to *IMS Version 8: Utilities Reference: System*, SC23-1309.

6.2.4 Single points of failure

Sometimes, a high level of reusability in the code providing common services to many applications can increase the danger of having a single point of failure (SPOF). Although we acknowledge the interest of reusability, the design of such reused modules should consider avoiding as much as possible creating single points of failure.

Possible solutions are:

- **Replicate the data.**

Parallel Sysplex architecture provides hardware and software functions that protect your databases. If using DEDBs, you have also the ability to replicate the database (up to seven times). IMS updates all copies when a change is made to only one copy, and if an I/O error is detected for a particular record in a copy, then another copy is used.

- Spread the data as much as possible.

To spread data, make more partitions, decrease the control interval size, or increase the free space level. Locks are taken in parents, CI, or UOW, depending on the type of the database you are accessing. For example, if the lock corresponds to a CI and you only have one segment in it because you have spread segments around, then you are not locking other transactions that intend to access different segments. This a useful technique for avoiding hot spots.

An example of an undesired effect of a SPOF is produced because of the retained locks situation on a critical database. When a data sharing subsystem fails, it might hold non-sharable locks on database resources until the retained locks are released. Refer to “Fast Database Recovery (FDBR)” on page 101 for an explanation of how critical applications can successfully handle this situation.

When an IMS member fails, the resources that were accessed at the time of failure remain engaged. This is known as a retained lock, and retained locks are not released until IMS is restarted with /ERE, or you have FDBR operational. To limit the impact to only part of the transactions, avoid hot spots.

The abended transactions return an U3303. When there are more transactions that abend with U3303 than transactions that end successfully, the transaction stops and the application becomes unavailable.

6.2.5 Batch considerations

The use of data sharing for online systems does not require the use of data sharing for batch jobs. Some customers want to use Parallel Sysplex to provide availability and capacity benefits to the online system, but do not need to eliminate the batch window. Nevertheless, in this section we offer recommendations on how to convert batch jobs to BMPs or data sharing DLI Batch, how to shorten the batch window, or how to eliminate it.

The programming effort for converting a batch job to a data sharing batch job is the same as that for converting it to a BMP. In both cases it is important not to hold locks for a long time. Checkpoints are required when the programs require a restart capability. The major differences between a batch data sharing job and a BMP are the JCL testing methods.

Converting batch jobs into DLIBATCH

You may want to use DLIBATCH jobs because of one of the following reasons:

- If you have a large batch job accessing sequential datasets, DB2 data, or both, but non-IMS DB data, you may consider converting this batch job to a DLIBATCH. The principal benefit is that you have the ability to issue checkpoints, therefore will be able to restart from the last checkpoint in case of failure. The sequential datasets are treated as GSAMS. When you restart the job, they are repositioned in the point of the last checkpoint. The DB2 has the standard recovery.

If you are accessing only sequential datasets, you can issue a checkpoint, for example, every 10 minutes; this will assure that in case of failure, you will not lose more than 10 minutes of work. But if you access DB2 data, you must issue checkpoints with greater frequency to free the resources obtained.

- If a batch job must have access to DL/I databases, the recommendation is to convert it to a BMP.
- If a BMP performs a large number of database updates, even if it takes enough checkpoints, you can have contention with your online transactions and degrade the performance of the application. The recommendation is try to avoid the peak hours of

processing, and maintain it as a BMP. But you can convert it into a DLIBATCH if you have high control of all the recovery datasets that DBRC must take in account.

Converting DLIBATCH jobs into BMPs

You can find more information on this topic in *IMS V8 Application Programming: DB*, SC27-1286 and *IMS V8 Administration Guide: System*, SC27-1284.

DL/I Batch jobs should be converted to BMPs to take full advantage of the following:

- ▶ Utilizing IMS recovery and restart processing
- ▶ Accessing IMS message queues
- ▶ Opening up access to Fast Path DEDB database structures
- ▶ Reducing the number of logs to manage, since BMPs input to the IMS log dataset only

Data Base Recovery Control (DBRC) must track all logs associated with a database. Because each batch job writes its own log, DBRC must track the logs created. Any database recovery will require the log data from all IMS systems that updated the database, including the batch jobs. The recovery process is much simpler if all IMS batch jobs are BMPs, because BMP log data is included with the online control region.

Application changes

The changes required to convert a DLI Batch job to a BMP are well understood by most IMS installations; following is a brief summary of these changes.

Create an I/O PCB in the Program Specification Block (PSB): BMPs must have an I/O PCB. By specifying the compatibility (CMPAT) option in the PSB, you can obtain an I/O PCB in Batch. The same PSB can then be used for both BMP and DLI Batch jobs.

Issue frequent checkpoint calls: IMS obtains read locks and update locks for every block retrieved from the shared database. To prevent the batch job from obtaining so many locks that online operation is impacted, the BMP should release its locks by executing a checkpoint call. Frequent checkpoint calls will reduce the number and duration of the locks held by the batch job.

The frequency of checkpoint calls depends on how much work a BMP does. There is a trade-off between a long interval, which gets many locks, and a short interval, with the extra overhead of the checkpoint call. The minimum frequency between checkpoint calls should be a complete business transaction, or “unit of work.” All database position is lost at CHKP time. It is wasteful to have to reposition within the same database records after checkpoint.

Note that adding these calls is required even if the batch job is not converted to a BMP. DLI batch jobs accessing a shared database will get locks that can impact the performance of the online transactions.

You can provide an installation-specific program that indicates to the application the best moment to issue the checkpoint. Using such a program eliminates that complexity to the application developer, and if the policy of issuing checkpoint changes, the application programmer is not impacted.

To implement this solution, invoke the DL/I calls through macros. Inside each macro you call the module changing the value of a variable that indicates to the program whether it has to issue the checkpoint or not. The macros with the call to that module should be incorporated within a copy to the program. The applications can still increment the number of commits/checkpoints that the standard module indicates.

Be restartable from a checkpoint: By executing a checkpoint call, you are informing IMS that your program has reached a point of recovery (or syncpoint). Obviously, this means that the batch job must be able to restart at this point.

Batch IMS jobs must implement their own restart logic. Restart can be implemented in many ways, but it may be convenient to use the IMS Extended Restart (XRST) call. This reduces the overhead and time required to restart from the beginning of the job, but if large amounts of data are written to the log via the use of extended checkpoint facility, delays to online transactions and fast path syncpoints could occur trying to write to the log at the same time.

The IMS Program Restart Facility is an IBM tool that allows BMPs to be restarted on different IMS systems without any restriction.

A performance item that should be investigated is the use of the checkpoint message suppression facility, because when a BMP (or batch program) issues a CHKP call, a message is produced to be sent to the MTO and z/OS console. This can be a burden to the message queues at high checkpoint rates. If suppression of the messages is required, use of the Dependent Region execution parameter CKPTID='NOMSG' (or for system-wide control, DFSVSMxx OPTION statement ISSUE681=) should be investigated.

Application programs should be able to change the checkpoint frequency without having to recompile the program. Following are methods to change the checkpoint frequency for a program:

- ▶ Use a checkpoint parameter in a database or z/OS file that can be read by the BMP.
- ▶ Pass a parameter value in SYSIN data.
- ▶ Use the dependent region execution parameter APARM= to provide the BMP with checkpoint call frequency when the application issues an INQ DL/I call with the subfunction ENVIRON.

BMP locking considerations

Sometimes the additional locking that occurs when accessing a shared database can cause an unacceptable increase in a job's elapsed time. In this case, the job must be examined carefully to determine the cause of the increase.

If a PROCOPT level is more than is required, then change it. For example, do not use PROCOPT=A if the data is only ever read. The PROCOPT could be set to GO to avoid all locking.

If a read with integrity is required, after reading the record, release the lock by issuing a GU call for a non-existent root. If you use the X'FF' key, you may still have some contention in the X'FF' position, but it will be reduced.

In several installations, a long-running batch job was found to be sequentially retrieving every segment in the database, but updating very few. The application was changed to retrieve the segments through a PCB with PROCOPT=GO (so no locking was done), and to do the updates through a second PCB that had PROCOPT=GR or PROCOPT=A.

BMP execution and buffer pool resources

As more BMPs are scheduled to execute concurrently with the online system, the sizes of the PSB, PSBW and DMB (if all DMBs are not resident) have to be monitored and adjusted if necessary. If a particular database is often scanned sequentially by BMPs, then a private database buffer subpool should be assigned to it. These are items that you can review with your system and DB administrator.

Reduce or eliminate the batch window

The best way of reducing a batch window is to eliminate it, so consider whether you can execute an additional transaction to do the batch-type work after each online transaction. This solution may be costly, however, as you need more resources and capacity to handle this approach. Whether this is a desirable solution or not depends on how important 24x7 availability is for you. Also keep in mind that need to avoid degrading the response time of your online transactions.

What is needed is a technique to separate the online process from the batch process, so you may want to consider using WebSphere MQ as a solution; it has large capacity, runs asynchronously, and is being used with IMS in many installations. Because WebSphere MQ enables you to process messages asynchronously, you can either process them as they arrive or wait to process them to avoid peaks. For more about using WebSphere MQ in a sysplex, refer to Chapter 7, “WebSphere MQ application considerations” on page 131.

You can also consider processing messages in batch jobs that can be scheduled more frequently than only once per day.

Another consideration is that you can use this solution to reduce your existing batch, instead of eliminating it. With WebSphere MQ, you can process all the work you can do in advance, and maintain in your batch only the part that must be run at a specific time, perhaps related to the end-of-day event.

Other considerations are:

- ▶ How the application processes the end-of-day; is it compatible with the solution?
- ▶ If spreading batch processes over the day, be aware of locking problems that the batch can cause to the online transactions.

Parallelize the batch

If you need batch window reduction, consider making partitions of your databases (for example, increasing the number of areas of your DEDB database). Then you can execute the batch processing in parallel for each one of these areas.

Note: This solution must be evaluated carefully if you need to merge the results; in that case, this additional requirement can diminish the advantage of parallel processing.

Consider the enhancements in IMS versions

During planning, keep in mind the following enhancements.

High Availability Large Databases (HALDB): They have been implemented in IMS V7. Hierarchical direct access method (HDAM) and hierarchical indexed direct access method (HIDAM) databases can be partitioned using either the HALDB Partition Definition utility (DSPXPDDU) or DBRC commands and are then described as High Availability Large Databases (HALDBs).

After you partition an HDAM database, it becomes a partitioned hierarchical direct access method (PHDAM) database. After you partition a HIDAM database, it becomes a partitioned hierarchical indexed direct access method (PHIDAM) database.

HALDB allows Single Partition Processing where BMP, JBP, and batch-processing applications can process a single partition of a HALDB independently, thus allowing parallel processing across multiple partitions of one HALDB.

You can find more information on HALDB in *IMS V8 Administration Guide: DB*, SC27-1283.

Image copies: One requirement prior to running a batch process is to take image copies of the databases involved so you can restore them in case of failure.

IMS V7 has enhanced the Database Image Copy 2 utility to allow compression for Image Copy 2 copies. IMS V8 has also enhanced this utility, to allow multiple database data sets to be copied in one execution of the utility. For more information, refer to *IMS V8 Utilities Reference: Database and Transaction Manager*, SC27-1308.

6.2.6 Converting MSDBs

MSDBs have many restrictions that can inhibit an installation from achieving continuous application availability. For example, inserting or deleting records to a MSDB requires that IMS be shut down and restarted. Also, customers with applications updating MSDBs have been unable to exploit APPC/IMS. To remove these restrictions, we recommend that you convert all MSDBs to SVSO-option DEDBs. SVSO DEDBs supply all the functions of DEDBs with the performance of MSDBs.

During the database conversion from MSDBs to SVSO DEDBs, application and database developers must be aware of the differences in the structure and handling of MSDBs compared to DEDB organizations. Most differences can be made transparent to the IMS program with the selection of the Program Control Block (PCB) VIEW=MSDB option.

If a program relies on it, the appropriate PCB should be changed to include VIEW=MSDB. For example:

```
PCB TYPE=DB,NAME=MS2DEDB,PROCOPT=GR,KEYLEN=24,VIEW=MSDB
```

The MSDB to DEDB Conversion Utility, DBFUCDB0, simplifies the process of converting an MSDB to a DEDB.

MSDB - DEDB differences

Following are characteristics of MSDBs, with considerations for a programmer converting a program which was written to support an MSDB accessing a SVSO-option DEDB.

Root-only databases

MSDBs are simple root-segment (single record) databases. This is not a problem for an MSDB conversion, because all IMS databases can be defined in this way.

Fixed-length segments

MSDBs have fixed-length segments. Because DEDBs also support fixed-length segments, a program written to use an MSDB can run using a SVSO DEDB, if the DEDB is defined with a fixed-length segment.

Data accessible at the field level

MSDB data is accessible at the field level as well as at the segment level. Most updates to an MSDB segment are done with the IMS FLD call. DEDBs also support the FLD call.

Segment length of MSDB database

An MSDB segment and its prefix must be less than 32 KB in length, but the maximum size of a DEDB segment is usually much less. The maximum size of a DEDB segment is 120 bytes less than the size of the fast path buffer size, which is usually 4 KB or 8 KB.

Increasing the length of the DEDB CI is not normally an option. All fast path buffers are the same length, and most installations would not be willing to increase the size of the fast path buffer pool to accommodate a single database.

Applications using large MSDB segments may have to be rewritten to store their data into several DEDB segments.

Hexadecimal search arguments

IMS treats hexadecimal search arguments differently for MSDBs than for DEDBs. If you are processing hexadecimal data, the data in the search argument, either Segment Search Argument (SSA) or Field Search Argument (FSA), must be represented in character hexadecimal format. This means that the length of the data in the search argument is twice the length of the data in the field in the database.

For example, let us assume that a one-byte field called TYPE contains hexadecimal data and you wish to locate an occurrence of the field that contains hexadecimal x'C1' (the letter A).

- ▶ If the segment is in a MSDB, you must search for C1 (x'C3F1').
- ▶ If the segment is in a non-MSDB you must search for x'C1'.

The PCB VIEW=MSDB option on a DEDB will cause IMS to handle SSAs and FSAs in a fashion compatible to MSDBs, so applications do not need to be changed.

Arithmetic fields compared arithmetically

IMS treats comparisons of arithmetic fields differently for MSDBs than for other IMS database types.

If the SSA names an arithmetic field, type P (Packed), H (Hexadecimal), or F (Float) as specified in the database description (DBD), the database search is performed using arithmetic comparisons rather than the logical comparisons used for other databases. This can produce different results between MSDBs and non-MSDB databases. For example, a single-digit packed-decimal field with a value of "1" can be expressed in hexadecimal, as X'1F', or X'1C'. Comparing a X'1F' to a X'1C' would result in an equal condition when using an MSDB, and a not-equal condition for other database organizations.

The PCB VIEW=MSDB option maintains the MSDB database field compare logic for DEDBs that have been converted from MSDBs.

Keyed database

MSDBs with keys are loaded into virtual storage in key-sequence order. DEDBs, on the other hand, are stored on DASD according to the location determined by an IMS database randomizer exit routine. This order is rarely sequential.

Applications which perform only equality searches (key = value) will have no problems. Applications which perform inequality searches (key > value) may have problems locating the expected segment.

There are two possible solutions. One solution is to change the program to use sequence independent calls. The other solution is to implement a "sequential" randomizer for the DEDB. The randomizer must ensure that the segments are loaded in the order of ascending keys. This may or may not be a complicated task, depending on the application and the key structure of the database.

Different commit view

IMS does not apply segment updates to an MSDB until a commit point, or sync point. Until you reach the commit point, all requestors will continue to get the original value of the segment every time they request it.

In contrast, IMS presents the updated value for DEDBs and full-function databases on requests for the segment made after the update. This can cause problems with applications that were written to expect the original value of the segment.

The PCB VIEW=MSDB option maintains the original commit view associated with MSDBs that are converted to DEDBs. You may have to increase the size of the fast path buffer pool, as this function is implemented by keeping both the original and the updated copy of DEDB buffers until sync point.

Locking differences

There are differences in the way that IMS obtains locks for DEDBs and MSDBs. These differences depend both on the processing option (PROCOPT) defined for the database and the type of call made to the database manager. These differences may require you to change the application to get compatible operation.

The locking behavior of a Get Unique (GU) call for a database defined so that fields can be replaced (PROCOPT=GR) is different for a SVSO DEDB and an MSDB. A GU on a MSDB with PROCOPT=GR will acquire a read lock for the duration of the call, allowing for multiple concurrent accesses to the same segment.

With a SVSO DEDB, a GU with PROCOPT=GR will acquire an exclusive lock that will not be released until sync point. This causes all accesses to a specific segment in the SVSO DEDB to be single-threaded.

This difference between the way locking is handled is based on the current IMS DB design. If a PCB has the capability to update a DEDB segment, an exclusive lock will be obtained to prevent other users from having access to the uncommitted data. With MSDBs, updates are not made until sync point, so no exclusive lock is required during the unit of work (UOW). When the program reaches a commit point, IMS reprocesses the FLD VERIFY/CHANGE calls. If the VERIFY test passes, the change is applied to the database. If the VERIFY test fails, the changes made since the previous commit are undone, and the transaction is reprocessed.

If the difference in locking implementation causes excessive serialization in a Parallel Sysplex, then the application could be changed to define the database with two different PCBs, one with PROCOPT=GO, (so no locks will be obtained), and another with PROCOPT=GR to perform the update calls.

6.3 IMS Transaction Manager

From the IMS Transaction Manager point of view, an application can run without any changes to its code, but we provide recommendations that will help you to exploit the availability and capacity that Parallel Sysplex offers.

A well-designed IMS application will complete its work with a minimum number of DL/I calls and I/Os and free locked resources as quickly as possible.

6.3.1 Affinities

Applications can run in a Parallel Sysplex with affinities or dependencies that obligate the transaction to run in one, or only in some, of the available images. In this section, we discuss affinities to enable you to decide whether to avoid them or keep them (for example, sometimes affinities can be used to ensure that you can control specific transactions).

In any case, if an affinity restricts the ability to run an application in only one single image, you must be aware of how an outage on that image will affect you.

Use of data in storage

This affinity is caused when an application keeps information in a storage resident table. This table is available only to programs running on the same z/OS system, so it causes a transaction-to-system affinity. This information can be either read-only, seldom updated, or very volatile. Removing this affinity is required before the application can be run in parallel on multiple IMS systems.

If the data is primarily read-only, the table can be loaded in each IMS system, with no application changes required. This table can be loaded in each IMS, but you would then require a change to the administrative procedures that update the tables to ensure that the updates, when needed, will be made into each IMS system.

If data is accessed very frequently, another way to solve this problem is to convert the data into a reentrant module. Compile it and obtain a load module. Then you can fetch the data from programs through a standard call that you can provide within a copy. You can use the abilities of your installation to keep this module loaded in LPA memory, or preloaded in your MPR, so you can enjoy quick access to the data. The original source for the data can be a DB2 table, and you only need a process to be able to transform this data into a source program.

Very active storage-resident tables, such as journals or other summary counters that get updated many times a second, must be examined carefully to develop a design that will allow multiple copies of the application to be run in parallel. Often, the recommendation is to convert the data to a shared database, for example, a DEDB SVSO.

Scheduling constraints

Check your transaction and application definitions in the IMS generation input. Look for MAXRGN=1 or SERIAL=YES specified in the TRANSACT macro, or SKHDTYP=SERIAL on the APPLCTN macro. These scheduling options will prevent parallelism of execution.

Other scheduling constraints to look for:

- ▶ Defining a single message processing region (MPR) with a given transaction class
- ▶ Defining a fast path message processing region IFP type in only one of the images

These constraints may be required to satisfy the serialization needs of an application, but they limit the possibility for the application to run in parallel.

To benefit from Parallel Sysplex, try to find an acceptable way to remove the serialization constraints; here are some ideas and examples you may find applicable:

- ▶ The application requires receiving input in time sequence.

In this case, you can add a column with a time stamp for each record when it is processed or originated. In fact, this method is used for sharing Fast Path DEDB sequential dependent segments. It has added an 8-byte time stamp to each sequential dependent to indicate when it is created. The sequential dependents are not stored in sequence, but they can be ordered using the time stamp before they are processed.

From the application point of view, you can also obtain a time stamp from the terminal. This may be possible, depending on the software you have available on the terminal.

You can also customize a randomizing module in order to give sequence to the records in the database for IMS, or define a clustering index on that column for DB2.

- ▶ Sequencing of execution

A terminal can initiate an order to process a movement between two accounts. Then, initiate another operation to cancel the first one. You must be sure that the second transaction will be processed only after the first one has ended.

In this situation, you need knowledge in the second transaction to check that the first transaction was processed. If it was not processed, you must schedule the second transaction again before returning an error to the terminal.

Access to non-shared resources

Examples of non-shared resources available only on one image can be:

- ▶ Connections to external companies
- ▶ Cryptographic machines
- ▶ Access to MSDBs (refer to the IMS DB considerations section)
- ▶ Automatic Teller Machines (ATMs)

Sometimes the solution is just a question of money; you can buy more cryptographic machines or ATMs. In other cases, the solution can lie in the way you make the connections available in the network.

6.3.2 Transaction workload balancing

You can achieve transaction workload balancing in two ways:

- ▶ Through the use of Multiple Systems Coupling (MSC) routing. There are exit routines available in IMS transaction input processing that can be used to automatically adjust the MSC destination of transactions.
- ▶ Through the use of the shared queues facility. The text in the following section is excerpted from the IBM Redbook *IMS in the Parallel Sysplex Volume I: Reviewing the IMSplex Technology*, SG24-6908.

Shared queues overview

Many IMS users are looking forward to managing their increasing application workloads efficiently now and in the future. In addition, they want to develop strategies to improve their current service levels and move toward continuous operations while maintaining or improving acceptable response times.

IMS users recognize the requirement to efficiently distribute and balance the total workload across multiple systems. IMS shared queues assist users in meeting these requirements by:

- ▶ Providing dynamic load balancing among multiple IMS systems in a Parallel Sysplex environment
- ▶ Optimizing overall throughput across the sysplex, so that no single IMS remains under utilized while others are overloaded
- ▶ Achieving improvements in continuous availability for applications
- ▶ Improving reliability and providing better failure isolation, so that any remaining IMS can continue processing the shared workload
- ▶ Adding on new IMS systems as workload increases

Without shared queues, each IMS system has its own queues for both input and output messages, and its own expedited message handler for Fast Path messages. Messages are processed in the local IMS system, unless that IMS sends the message to another IMS system. With shared queues, all IMS systems in a Parallel Sysplex can share a common set of message queues stored in the Coupling Facility. Full function and Fast Path input and output messages can be processed by any IMS system that has access to the shared queues and is capable of processing the message.

Any IMS system running in a Parallel Sysplex can participate in processing application workload as defined in its IMS system definition according to available CPU capacity and resources. With a shared queues implementation, new IMS subsystems can easily be added to the Sysplex as workload increases or as extra load must be processed (scalability). The content of shared queues is not affected by hardware or software failures. Remaining IMSs continue to schedule their applications and process messages stored in the Coupling Facility.

There are really two sets of shared queues - those that support IMS full function messages, and another set that supports IMS Fast Path expedited message handler messages. The queues themselves are really lists of messages that are kept in a list structures in the Coupling Facility.

IMS registers interest in queues by the queue name. If a message must be processed by a particular IMS, the identifier of the IMS that is to process the message is placed in the queue name. No other IMS has an interest in this unique queue name.

A full function transaction is always passed to the Common Queue server (CQS) to be put on the shared queue structure. A full function transaction can be processed by the local IMS system when it is first received, or any of the IMSs in the shared-queues environment that have a registered interest in the queue can bid for the right to process the transaction. The exception to this rule is when the transaction is a SERIAL type, in which case it can only be processed by the IMS system which received it. With IMS Version 6 also the APPC and OTMA originated messages are exceptions to this rule. IMS Version 7 began shared queues support for asynchronous APPC and OTMA messages and IMS Version 8 supports all APPC and OTMA transactions.

IMS must have the following resources available to process a transaction:

- ▶ A message processing region (MPR) that is waiting for work. This MPR must be able to process the class for which that transaction has been assigned.
- ▶ The PSB that runs the transaction must be started.

When the local IMS system receives a full function transaction, IMS determines whether the transaction can be processed immediately in the local IMS system before it places the transaction on the transaction ready queue. The transaction can be processed immediately only if the resources are available. In that case, because the transaction is not placed on the transaction ready queue, other IMS systems that have a registered interest in this queue are not notified. If the local IMS system did not have the resources available to process the transaction immediately, CQS passes the transaction to the transaction ready queue. If there are no other entries on the queue for this transaction, all IMSs that have a registered interest in the queue are notified. One of them will process the transaction.

6.3.3 Transaction simplification

Normally, IMS output messages or responses are returned to the user at the IMS terminal. However, IMS TM programs can change the destination of their output with the IMS CHNG call.

This can be done for functional purposes (for example, gathering information from other companies in other sites that are connected to the main site), or just for load balancing, because at some point in the past it was necessary to create a new IMS in order to be able to process all the transactions.

Some application logic is required to ensure that the transaction sent to the remote IMS system executed correctly, as there is no way to guarantee that the transaction successfully completed. It is the application's responsibility to ensure that all databases are in a consistent state after a system or application failure. Application programs that use this message switching technique often must provide logic to detect and handle a possible duplicate transaction, or to provide a transaction reversal.

With IMS data sharing, these kinds of applications may be simplified. Instead of using MSC routing to send a transaction to an IMS that has access to some data, the database can be accessed directly using the data sharing capabilities of the Parallel Sysplex. In effect, the database is available to all IMS transactions wherever they are entered and processed. This can greatly simplify application logic and database management. Some installations have found that this is the biggest single factor influencing the decision to implement Parallel Sysplex.

6.3.4 General IMS TM application performance considerations

Although the following performance considerations apply to both single IMS image and Parallel Sysplex configurations, they are very important in Parallel Sysplex environments:

- Message priming

During the scheduling of application programs, IMS obtains the input message and presents it to the dependent regions inter-region communication area.

If the first DL/I call of an online transaction is not a GU on the I/O PCB, then this primed message is lost and will have to be obtained from the queue again when this call is made. The GU should be issued first even if the application uses an INIT call.

- Scheduling batch processing during prime online periods

If batch processing includes sequential scans of shared databases or report writing to online terminals, there could be a negative performance impact during prime online periods. Current scheduling processes should be examined with this in mind.

Consider the use of MQ in your transactions in order to change the way you process your batch. Each transaction can send a message to a queue. Afterwards, each message is asynchronously processed in another transaction in order to process the batch work related with the first transaction. This results in availability advantage, but probably in a higher consumption of resources.

- Avoidance of large conversation Scratch Pad Areas (SPAs)

Scratch Pad Areas are placed in the message queues for recovery, so the use of very large SPAs could cause additional message queue and log I/O.

- Reduction in the use of the ROLL call

When a ROLL call is issued by an application program, IMS will perform application abend processing. This involves dynamic backout, issuing an ABENDU778 to terminate the program controller for the dependent region, performing the I/O to the log for the type 67FF abend records, reattaching the program controller, preloading of the non/reentrant preloaded programs and reinitializing the dynamic BLDL list.

The ROLB should be used, rather than the ROLL call, since it will only back out the transaction and then return control to the application which could then continue to make decisions on what option to take. The dependent region program controller will not be terminated. Of course, the use of the ROLL or ROLB call will probably be more efficient than scheduling another program to logically back out the original transaction.

- Using multiple transaction codes for the same program

Additional scheduling overhead occurs when there are multiple transaction codes per application program. Only one transaction code can be processed during application scheduling. Separate applications per transaction code, transaction subcodes within the input message text, or the use of parallel scheduling of the same application in another region for different transaction code are possible alternatives.

- Use of common MFS formats

If possible the installation conventions should stress the use of common message formats. This reduces the storage necessary to store formats and activity to the MFS format pool.

- Use of the MODE=SNGL rather than the MODE=MULT parameter

The transaction parameter of MODE=SNGL should be used whenever possible, rather than using MODE=MULT. MODE=SNGL causes altered data base records to be written back to the data base on each request for a new message via a GU call to the I/O PCB. MODE=MULT is the default and would normally tie up more database blocks and buffers and sending of message output (unless an EXPRESS PCB is used), since these activities would be delayed until program termination time.

- Program structure and use of preload

Attempt to utilize modular application designs where exceptional condition routines are outside of the mainline code. This keeps the mainline working set size as small as possible.

Highly used general modules, like those which providing service to many applications, or which provide special control data, can be considered to be preloaded in the message processing regions.

You must then be careful when upgrading to a new version of any such module, since you have to stop and re initiate the message processing regions. Some installations choose to stop and start the MPRs at some hour at night.

- Reducing the number of DL/I calls

The number of transaction manager-related calls from an application could be reduced in the following manner. (Of course it is much easier to invoke these recommendations during the design stages.)

Output messages should be shipped as a single ISRT call per message, screen or page using MFS rather than issuing multiple ISRT calls.

Where it is possible, the input should come in as a single segment message rather than a single GU followed by multiple GNs. Less application, MFS, IMS communication analyzer and message queue resources will be used this way.

6.3.5 Connectivity

Figure 6-3 on page 125 shows the components related to connectivity in IMS. In the upper part of the figure are the two main protocols used to connect to legacy applications: VTAM and TCP/IP.

Connector type	Connection type	Description
WebSphere MQ	ESS - OUT	Existing IMS runs applications which explicitly send messages through MQ calls
WebSphere MQ	OTMA	Existing IMS receives and sends WebSphere MQ messages through IMS Bridge using OTMA
Web Server	OTMA	Existing IMS receives transactions from the Web through WebSphere MQ or IMS Connect to OTMA
Web Server	APPC	Existing IMS receives transactions from the Web through an APPC connection.
TCP/IP Sockets	BMP	Existing IMS uses a BMP to provide a sockets interface.
TCP/IP	OTMA	Existing IMS uses IMS Connect for sockets connections.
DCE RPC	ISC	Existing IMS receives RPC requests through DCE Application Server's ISC interface.
DCE RPC	APPC	Existing IMS receives RPC requests through DCE Application Server's APPC interface.
DCE RPC	OTMA	Existing IMS receives RPC requests through DCE Application Server's OTMA interface.
ODBA	DRA	Provides z/OS batch application programs, DB2 Stored Procedures, and WebSphere EJBs access to IMS

VTAM Generic Resources (VGR) for SNA

VTAM Generic Resources (VGR) support allows terminals to log on using a generic name rather than a specific IMS APPLID when requesting a session with one of the IMSs in a Generic Resource Group. VTAM Generic Resource support selects one of the IMS subsystems in the generic resource group to which the session request will be routed. VTAM's session selection algorithm enables the session requests to be evenly distributed across the IMS subsystems to achieve network workload balancing while simplifying the terminal end-user interface to the IMSs in the Parallel Sysplex.

Sometimes you need to force the connection of a special group of terminals to a specific IMS image. This can happen, for example, when a bank is deploying a new version of terminal software that must be tested in functionality and capacity; you can decide which terminals will have access to the new application features implemented only in a specific logical partition (stepwise deployment). Another example is giving access to new levels of software products only available on one image.

VTAM Generic Resources and shared queues coexistence

The major difference between shared queues and VGR is that IMS shared queues balance the application workload across IMSs, while VGR balances the network workload between IMS and its logged-on users. Both are valid requirements, and shared queues and VGR can be used either separately or together.

In addition to application workload balancing, shared queues offer users an easy way to incrementally increase or decrease the capacity of a Parallel Sysplex without disturbing existing logged-on users. This may be done by just starting a new IMS when the workload gets heavy. The new IMS does not have to have network and users connected to it. When the workload decreases again, just quiesce and normally terminate that IMS; no end users are affected.

There are certainly some availability improvements with shared queues. When one IMS fails, the work that it put on the queue can be processed by other IMSs while the failed one is being restarted. And, if it is necessary or desirable to cold start one (or all) IMSs, the shared queues are not lost; they remain in the CF for access by the restarted IMSs when they are ready.

Web requests to legacy

Today, enterprises must increase the capacity of their system architecture and provide 24x7 availability because of the demands of the e-business environment. Applications that in the past could have enjoyed a large batch window now have to be available 24/7.

Figure 6-4 illustrates IMS connectivity being propagated to the Web world; a request is generated in a browser, and it can follow different paths through WebSphere until it reaches the IMS. The figure only shows one IMS, but you can extrapolate to an IMSplex.

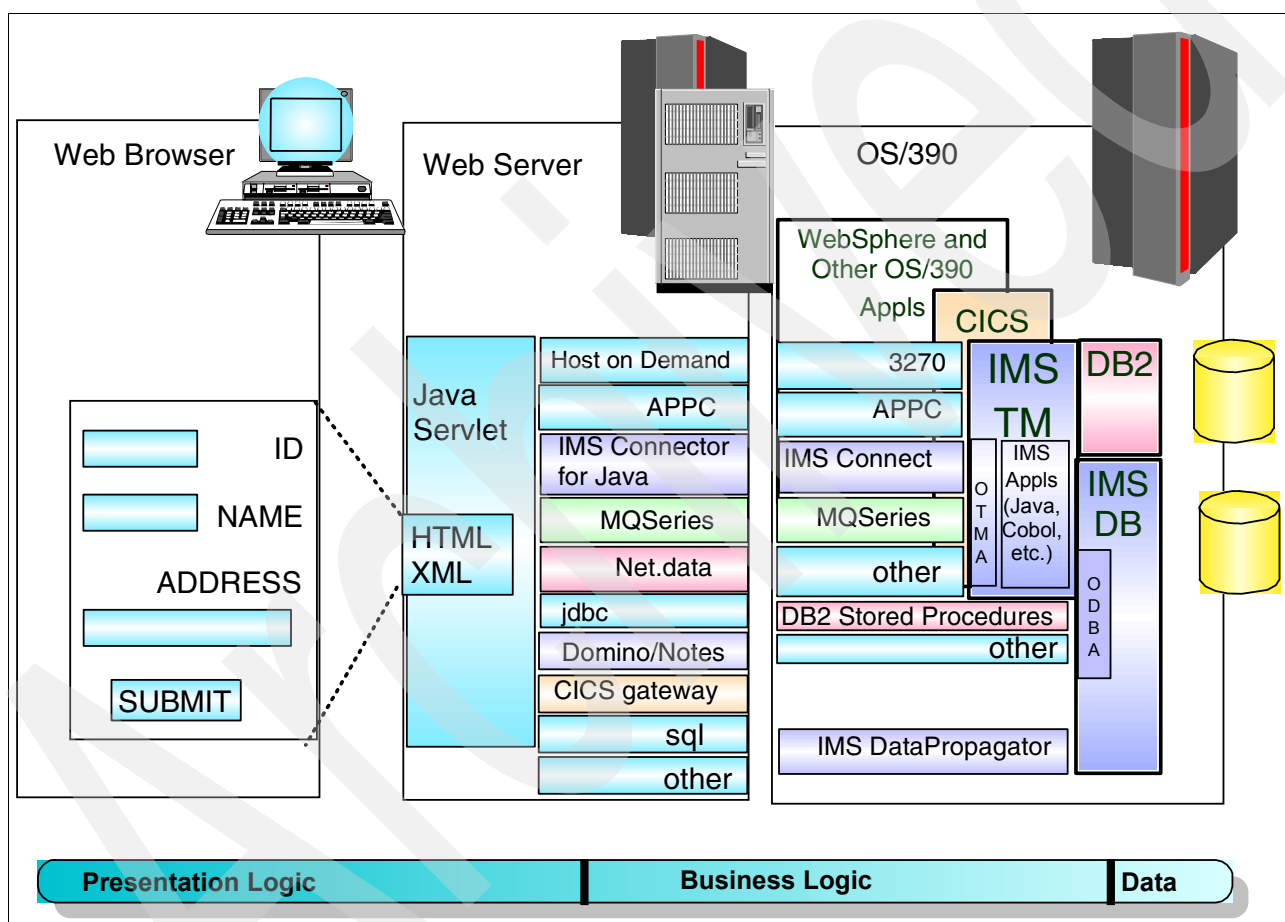


Figure 6-4 Connectivity Web-IMS

6.3.6 IMS configuration considerations

An IMSplex can be configured in several ways, depending on technical and business requirements. Here we present a few options for your consideration.

Cloning

One way to configure an IMSplex is to replace an existing IMS system with multiple IMS systems, each capable of providing the same services and functions as the original IMS; this is called *cloning*. When cloning is used, the IMSplex contains multiple servers, each able to provide similar functions to the end-user community.

The advantages of cloning are:

- ▶ Increased availability

If multiple servers exist, the loss of one server does not cause a complete loss of IMS service. In addition, preventive or emergency maintenance; either hardware or software, can be introduced into the IMSplex, a server at a time.

- ▶ Increased capacity

As the workload requirements of the enterprise grow, cloning allows additional IMS systems to be brought into the IMSplex, as required, in a non-disruptive manner.

- ▶ Ability to benefit from new hardware technology

Distributing the workload across multiple processors in the IMSplex, while preserving the image of a single IMS system to the end user community, allows the enterprise to gain the benefits (price and performance) of new processor technology without having to be overly concerned about the power of a single processor.

Multiple cloned IMS systems could be very similar but, perhaps not identical. Factors that might force systems to differ depend on several conditions:

- ▶ There may be affinities within applications, client file use, special operating processes or specific hardware attachments that might preclude complete cloning.
- ▶ Communication among IMS systems that use traditional queuing (for example, for workload balancing) can be achieved using MSC and/or ISC. These MSC and/or ISC definitions must be different and, therefore, cannot be cloned. Distributing workload among IMS systems can be automatically achieved with shared queues, thus eliminating the need for unique MSC and/or ISC definitions.

Joining

You can combine multiple, somewhat independent, IMS systems into an IMSplex. This situation might arise when these multiple IMS systems need to share a subset of the enterprise's databases. Combining multiple IMS systems is called *joining*. When joining is used, the IMSplex is not viewed as a single provider of service to the end-user community. Instead, it continues to be viewed as multiple, independent providers of IMS service.

An example of joining is a situation where there are three IMS systems on three different z/OS systems, each one handling the IMS processing requirements for three different manufacturing plants. The vast majority of the databases in each system are independent (unique to the specific manufacturing facility); however, a few databases represent corporate, rather than plant, data. Therefore, only limited cloning of functionality is required.

If systems are to be joined rather than cloned, you give up several advantages of a Parallel Sysplex environment:

- ▶ Application availability. When an application can only execute on a single system, the loss of that system stops all processing for that application until the failed system is restarted.

- Workload balancing. By restricting an application's processing to a single IMS, you cannot take advantage of excess processing capacity that might be available in the other IMS systems in the sysplex.

Front-end, back-end

Another configuration option is to replace an existing IMS system with one or more front-end and back-end systems. The design is that the front-end IMS systems have the IMS network connected to them, but do not have any dependent regions to process application programs. The back-end IMS systems have dependent regions to process application programs, but do not have the IMS network connected.

Transactions and messages flow between the front-end and back-end systems through MSC, ISC links, and shared queues, or any combination of these. VTAM Generic Resources can also be used, to balance the network workload across the joined systems while shared queues enable the transaction workload to be routed to the correct system for processing. An advantage of a front-end, back-end system is that an application abend cannot bring down the network (no U113 on the front-end).

Archived

WebSphere MQ application considerations

In this chapter we provide guidance for application programmers who want to create applications that run in WebSphere MQ for z/OS queue-sharing groups. We address new terminology and discuss application considerations when using queue managers that are members of a QSG.

This chapter contains the following:

- ▶ “A brief introduction to WebSphere MQ” on page 132
- ▶ “A standalone queue manager” on page 133
- ▶ “Queue managers in a queue-sharing group” on page 134
- ▶ “Coupling Facility list structures” on page 135
- ▶ “Channels in a queue-sharing group” on page 137
- ▶ “Intra-group queuing” on page 141
- ▶ “Benefits of using queue sharing groups” on page 143
- ▶ “Initial considerations related to the use of QSGs” on page 144
- ▶ “How applications can exploit WebSphere MQ shared queues” on page 147
- ▶ “Limitations and restrictions” on page 158
- ▶ “Sharing and clustering” on page 159
- ▶ “Some final considerations” on page 161

7.1 A brief introduction to WebSphere MQ

WebSphere MQ products enable applications to participate in message-driven processing. With message-driven processing, applications can communicate across the same or different platforms by using the appropriate WebSphere MQ products. WebSphere MQ products implement a common application programming interface (API) that applications can use to achieve the transfer of messages via queues.

The common API consists of the verbs listed in Table 7-1.

Table 7-1 API verbs

Verb	Description
MQCONN, or MQCONNX	Connects to a WebSphere MQ queue manager.
MQDISC	Disconnects from a WebSphere MQ queue manager.
MQOPEN	Opens a WebSphere MQ object for subsequent access.
MQCLOSE	Closes a WebSphere MQ object.
MQPUT	Places a message on a WebSphere MQ queue.
MQPUT1	Opens a WebSphere MQ queue, places a message on the queue and closes the queue.
MQGET	Retrieves a message from a WebSphere MQ queue.
MQCMIT	Commits a unit of work.
MQBACK	Backs out a unit of work.
MQINQ	Inquires on the attributes of a WebSphere MQ object.
MQSET	Sets selected attributes of a WebSphere MQ queue object.

A *message* is a string of bytes that is meaningful to the applications that use it. Messages are used for transferring information from one application program to another (or to different parts of the same application). WebSphere MQ messages consist of a message descriptor that defines control information such as the type and priority of the message, and the application data.

A *queue* is a WebSphere MQ object that is essentially a data structure used to store messages. Queues are managed by a queue manager.

Other types of WebSphere MQ objects include processes, namelists, and the queue manager attributes object. A queue manager is responsible for maintaining the queues it owns, for managing requests from applications as well as from subcomponents of the queue manager, for the storage and retrieval of messages from these queues, and for managing the set of resources that are used by WebSphere MQ. One such resource is the *channel initiator*.

The channel initiator provides and manages resources that enable the transfer of messages across a network between queue managers, or between client applications and queue managers. This process is known as *distributed queuing* or *intercommunication*. Relevant resources include message channels, MQI channels, and message channel agents (MCAs).

Message channels define the communications links and behavior used to transfer messages from one queue manager to another. Message channels can be of various types: sender, receiver, server, requester, cluster sender, and cluster receiver. A channel is established by defining one of these message channel types at one queue manager, and another compatible type at the other queue manager. Compatible combinations are sender-receiver, sender-requester, server-requester, server-receiver, and cluster sender-cluster receiver.

MQI channels are used to transfer messages between WebSphere MQ client applications and server queue managers. MQI channels can be of type client connection (to represent the client end) and server connection (to represent the server queue manager end). The only compatible combination is client connection-server connection. Essentially, the client issues WebSphere MQ API calls that are serviced by the server.

TCP/IP or LU6.2 is typically used to establish the physical communications link between the message channels.

Message channel agents (MCA) are the components of the channel initiator that control the sending and receiving of messages. There is one MCA at each end of a message channel. The MCA at the sending end is known as the sender (or outbound) MCA. The sender MCA retrieves messages from a special type of queue known as a transmission queue and places them on a communications link. The MCA that receives these messages and places them on to the target queue(s) at the remote end is known as the receiver (or inbound) MCA.

A WebSphere MQ client is a part of the WebSphere MQ product that can be installed on a separate machine from the server machine where the base product is installed and where the queue manager(s) runs. A WebSphere MQ application can be run on a WebSphere MQ client and it can connect to one or more queue managers on one or more WebSphere MQ servers. An MQI channel is established at the time of connection and it remains active until the time of disconnection.

Queue managers can be grouped to form a cluster. Queue managers in a cluster can make the queues that they host available to every other queue manager in the cluster. Any queue manager can send a message to any other queue manager in the same cluster without the need for many of the object definitions required for standard distributed queuing. Each queue manager in the cluster has a single transmission queue from which cluster sender MCAs can transmit messages to any other queue manager in the cluster.

WebSphere MQ for z/OS V5.2 (and above) exploits the Coupling Facility (CF) technology and a shared DB2 database to allow queue managers to be configured to form a cooperating queue sharing group (QSG). Within a QSG, queues may be defined as shared queues which are accessible to all queue managers in the QSG. Queues which remain accessible by individual queue managers only are non-shared or private queues. A new class of channels known as *shared channels* are also provided. These channels offer enhanced behavior and availability over non-shared or private channels

7.2 A standalone queue manager

On z/OS, WebSphere MQ runs as a z/OS subsystem. Within the subsystem, the queue manager is activated by a startup procedure. The subsystem and the queue manager have the same name of up to four characters. Applications connect to the queue manager using this name. The queue manager owns and manages the set of resources that are used by WebSphere MQ. These include:

- ▶ *Page sets* that hold the object definitions and message data
- ▶ *Logs* that are used to recover messages and objects in the event of queue manager failure

- *Processor storage*
- *Connections* through which different application environments (CICS, IMS, and Batch) can access the common API (via adapters that are part of the queue manager)
- *Channel initiator*

Figure 7-1 illustrates a queue manager, a channel initiator, and connections to the queue manager from different application environments.

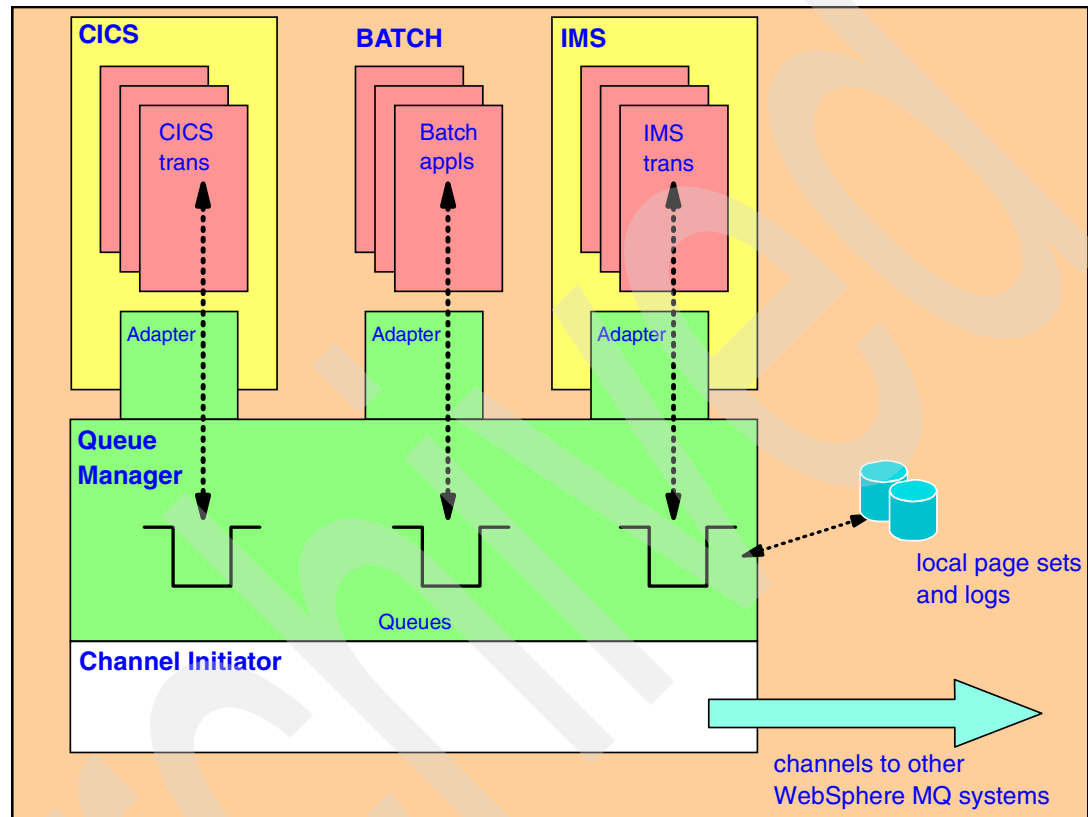


Figure 7-1 Components of a queue manager

7.3 Queue managers in a queue-sharing group

A group of queue managers that can access the same WebSphere MQ object definitions and that can share message data is called a Queue Sharing Group (QSG). A QSG has a name of up to four characters. QSG names within a network must be unique, and different from any queue manager names. QSGs on WebSphere MQ for z/OS consist of the following components:

- A number of z/OS queue managers (running within a single z/OS sysplex) along with their associated channel initiators
- One or more Coupling Facility structures (CF structures) that are used to hold messages on shared queues
- A shared DB2 database (also known as the *DB2 shared repository*) that is used to hold shared object definitions

Definitions of shared objects are held in the DB2 shared repository. Shared objects can be shared local queues or group objects with a common definition covering all the queue managers in the QSG. Each queue manager in the QSG connects to a DB2 subsystem that is a member of the same DB2 data sharing group. This allows each queue manager to access the definitions and create instances of them.

Figure 7-2 illustrates a QSG that contains two queue managers, QM1 and QM2. Each queue manager in the Queue Sharing Group QSG1 has an associated channel initiator and its own local page sets and log data sets, as well as access to the Coupling Facility and the DB2 shared repository.

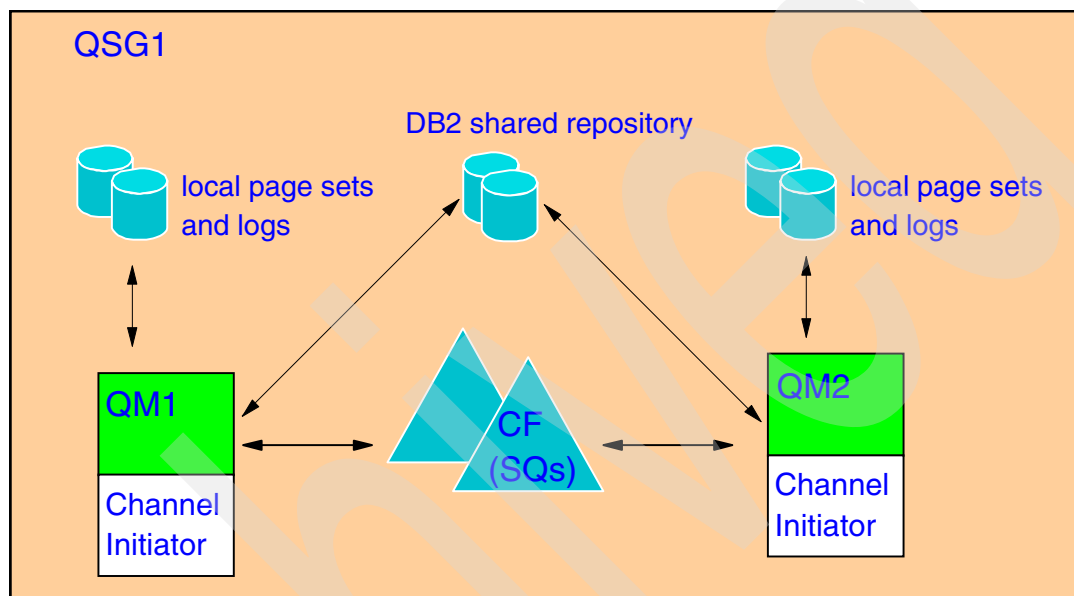


Figure 7-2 A queue-sharing group

A shared local queue can hold messages that can be accessed concurrently by any queue manager in the QSG. For example, an application connected to a queue manager in the QSG can put a message to a shared queue while another application, connected to a different queue manager in the same QSG, can get this message. This provides a rapid mechanism for communication within a queue-sharing group without the need to define and start WebSphere MQ channels between queue managers.

7.4 Coupling Facility list structures

The messages on shared queues are held in Coupling Facility *list structures*. Two types of list structures are used: *administration list structures*, for holding information required for unit of work recovery and for coordinating WebSphere internal activity across the QSG, and *application list structures*, for holding messages on shared queues.

Figure 7-3 on page 136 illustrates both types of list structures.

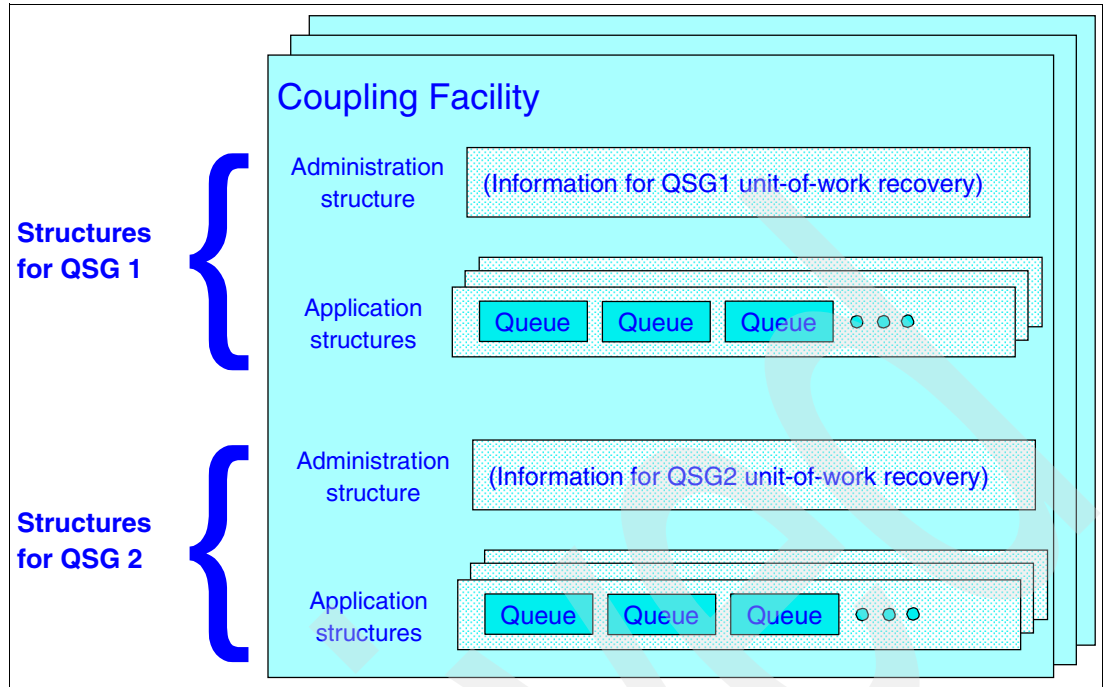


Figure 7-3 Coupling Facility list structures

Each Coupling Facility list structure is dedicated to a specific QSG, but a Coupling Facility can hold structures for more than one QSG. Each queue manager can only belong to a single QSG, and queue managers in different QSGs cannot share data. Up to 32 queue managers in a QSG can connect simultaneously to a Coupling Facility list structure.

Each queue manager can connect to up to 64 Coupling Facility list structures including the administration list structure. A single Coupling Facility list structure can contain message data for up to 512 shared queues. The size of a list structure limits the space available for message data. The size of a list structure itself is limited by the fact that it must lie within a single Coupling Facility, and that the available Coupling Facility storage may be shared between other structures for WebSphere MQ and other products.

Figure 7-4 on page 137 illustrates a Coupling Facility being shared between two QSGs. It is possible that the two QSGs represent production QSGs (set up for two suites of application programs that for reasons of sensitivity must be restricted from interacting with each other), or a production and a test QSG (due to a limitation of there being only one Coupling Facility available, for example). In the latter case, the same named shared queues can be defined to the list structures for both QSGs to assist in migration from test to production.

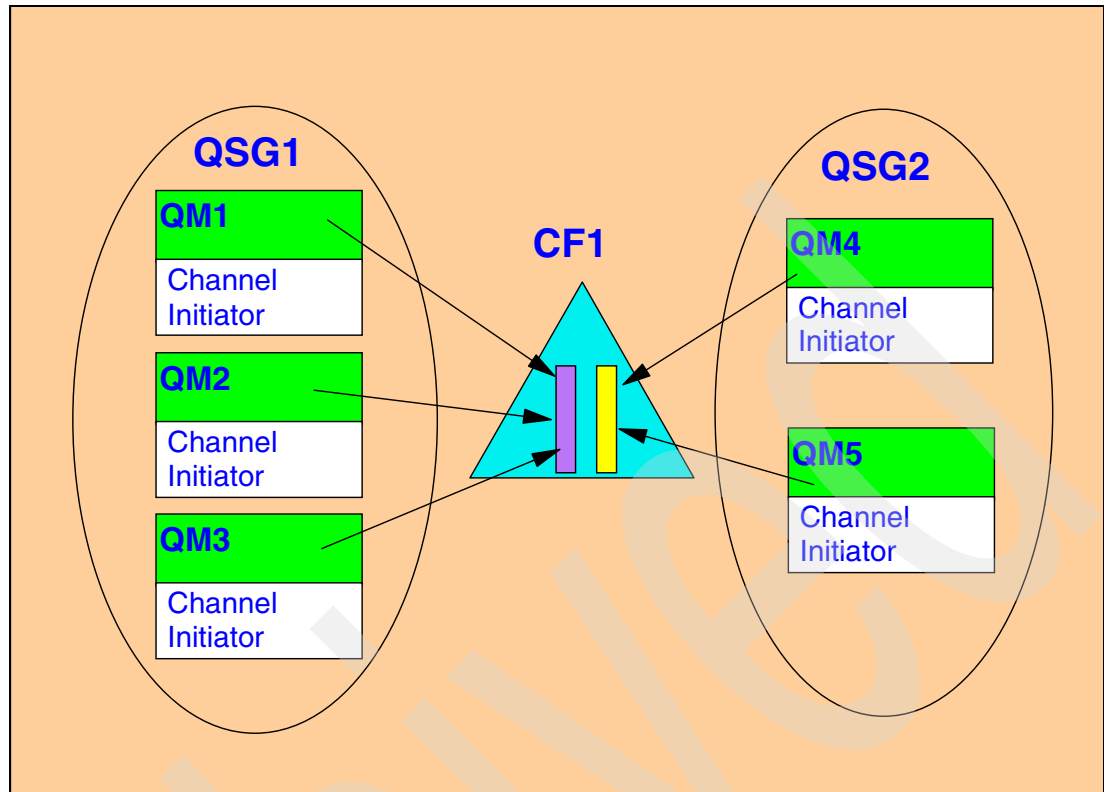


Figure 7-4 Sharing a Coupling Facility between two queue sharing groups

7.5 Channels in a queue-sharing group

In the following sections we take a closer look at channels in queue-sharing groups, discussing channel initiators, shared inbound channels, and shared outbound channels.

7.5.1 Channel initiators

Multiple channel-initiators are employed within a queue-sharing group to obtain capacity and availability improvements over stand-alone channel-initiators. The architecture and behavior of shared channels make this possible, while requiring few, if any, application changes.

There are two repositories that hold data relevant to running a channel. Firstly, the synchronization queue holds information about the last batch of messages transmitted. This queue is a regular MQ queue reserved for system use that holds the batch data in MQ messages. This data is used to enforce assured, once-only message delivery. Secondly, the channel status table stores information about the state of all active channel instances. It is used to enable correct channel behavior. For example, an already running channel must ignore a subsequent start request, whereas a stopped channel requires a start-command to be issued against it before it may be used to transmit messages.

Two classes of channels are available: *private channels* retain all the behavior of pre-QSG versions of MQ prior to WebSphere MQ 5.2 and are managed and run unchanged, and *shared channels* benefit from QSG-related improvements. The class of a channel instance is determined by the type of the transmission queue in the channel definition (for outbound channels) and the listener port used when a connection is made (for inbound channels).

To enable high availability, any channel initiator in the QSG must be able to run a shared channel, and the channel must be recoverable by any channel initiator following a failure. This requires that the state repositories for a shared channel must be accessible by any of the QSG channel initiators.

In order to achieve this, the synchronization queue used by shared channels is a shared queue (private channels continue to use private synchronization queues), and a subset of a shared channel's status resides in the DB2 shared repository (private channels continue to use a status-table held in the channel initiator's address space). The full status record for a shared channel is also stored in-memory by the channel initiator currently running the channel.

7.5.2 Shared inbound channels

The inbound channel architecture is shown in Figure 7-5 on page 139. An MQ channel initiator in a QSG may listen for inbound connections on two different kinds of port. In fact, these ports may actually be two different ports on the same TCP/IP address, two entirely separate TCP/IP addresses, or different LU6.2 logical units, but the term “port” is used here as shorthand.

A connection made through the private port (prvP) causes the receiving end of the channel to act as a private channel. If a connection is made through the group port (grpP), then the receiving end of the channel behaves as a shared channel. In addition, a generic port (genP) is assigned to the QSG as a whole. This port may be managed by any front-end routing hardware or software, for example Sysplex Distributor, WLM DNS, 2216 router, CISCO routers, VTAM Generic Resources (GR). Any connections to this port should be rerouted by the front-end hardware or software to the group port of one of the QSG channel initiators (the choice is determined by the front-end mechanism, not WebSphere MQ). If a failure should occur on one channel initiator or queue manager and the channel connection is broken, the front-end mechanism will establish a connection to another channel initiator, which will be able to continue processing of the shared channel, thereby providing high availability.

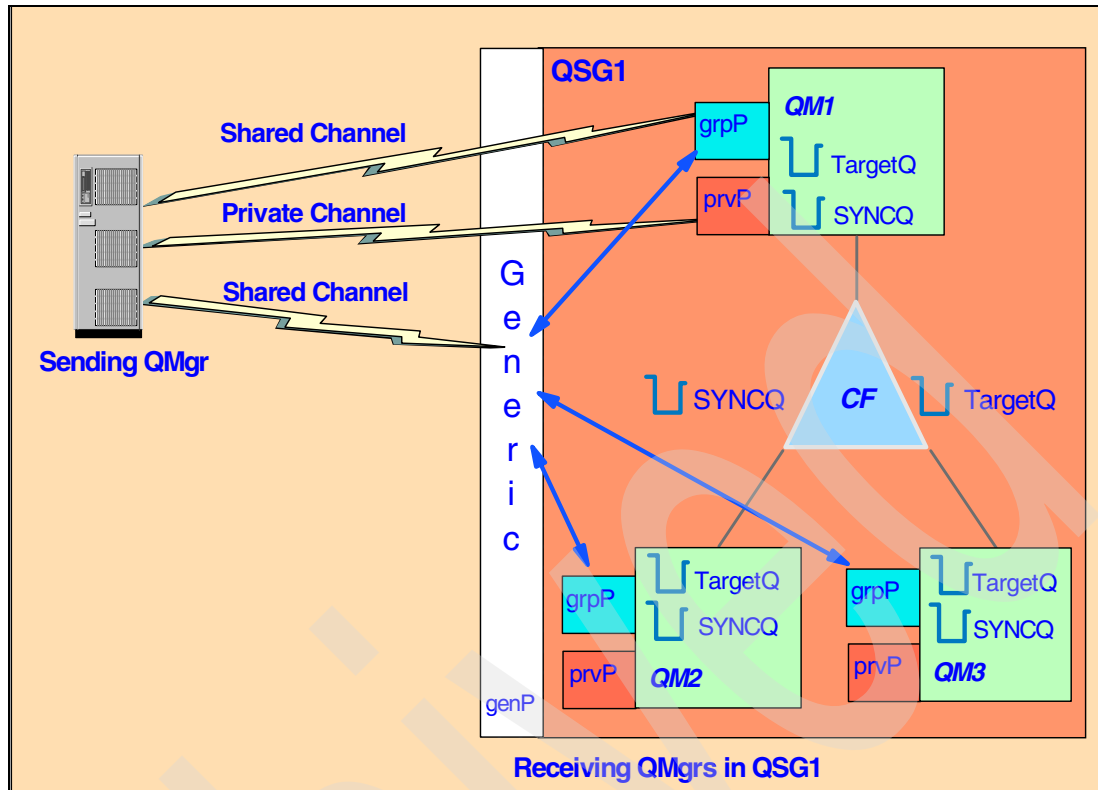


Figure 7-5 Inbound shared channel architecture

Notes:

- ▶ This technique for exploiting shared inbound channels is also feasible for MQI channels servicing WebSphere MQ clients.
- ▶ The type of port (private or group) that an inbound channel is connected to determines whether the receiving end of a channel is private or shared.
- ▶ The workload balancing of the channels is determined by the front-end mechanism and may or may not be dependent on the actual workload.
- ▶ Depending on the front-end mechanism, it may be possible to direct the connection through the generic port to a specific channel initiator (the primary channel initiator) and only connect to other (secondary) channel initiators if the primary channel initiator is not available. In this way you may control your primary channel flow and still exploit the availability.
- ▶ The distribution of the channel connections in the receiving end is completely transparent to the sending end. And the sending end only has to define one channel instance.

7.5.3 Shared outbound channels

The outbound channel architecture is shown in Figure 7-6 on page 140. A private channel transmits messages residing on a private transmission queue, whereas a shared channel transmits messages residing on a shared transmission queue.

When a start request is generated for an outbound shared channel, the channel initiator processing the request chooses the channel initiator that will run the channel via a workload-balancing algorithm.

Start requests may result from trigger-starting the channel, or from a command. The algorithm calculates a load-number for each channel initiator in the QSG based upon the number of channels a channel initiator is currently running and the maximum number of channels it has been configured to run. These numbers are stored in the CF for every QSG channel initiator, so that they are accessible by all the QSG members. The least loaded channel initiator is requested to run the channel. If the lowest load number occurs more than once a weighted-random choice is used to choose a particular channel initiator, with the weighting proportional to the headroom available (the current number of channels being run subtracted from the maximum number of channels it has been configured to run).

The load balancing algorithm aims to target start channel requests to the most appropriate channel initiator to keep the loads on each channel initiator approximately balanced across the QSG.

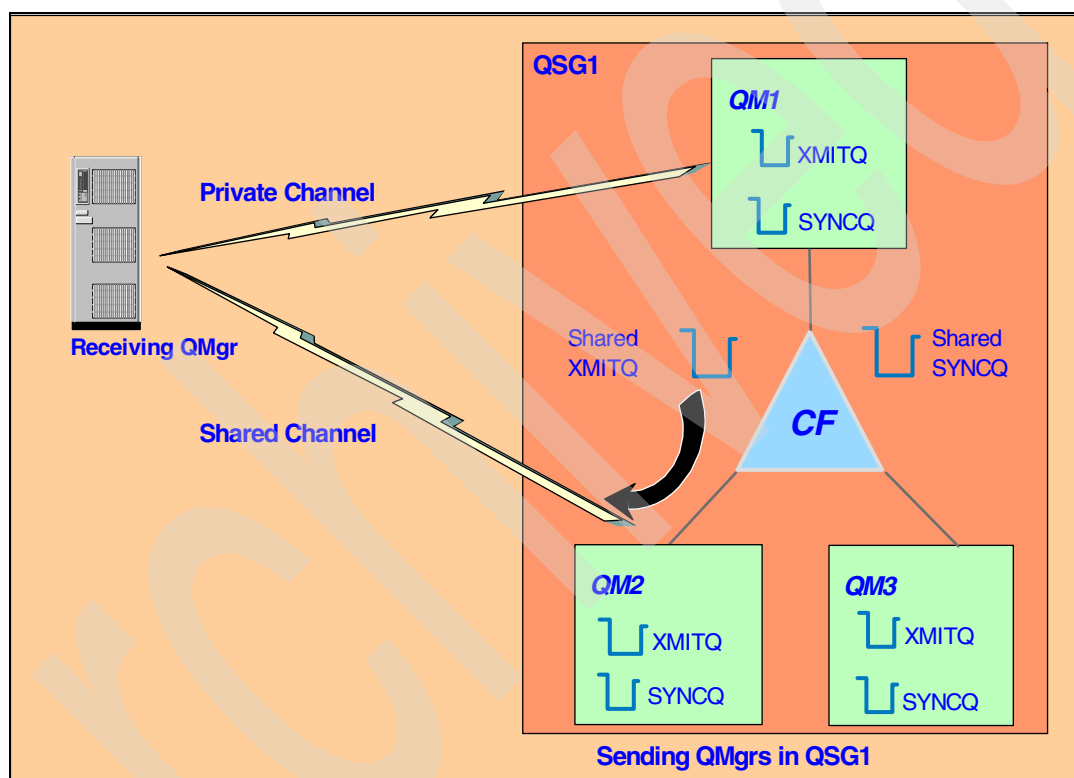


Figure 7-6 Outbound shared channel architecture

Notes:

- ▶ The type of transmission queue (private or shared) that the sending end of an outbound channel services will determine whether it is private or shared.
- ▶ A shared channel derives its benefits over private channels from the channel initiators in the QSG cooperating to provide enhanced behavior; this is unlike a shared queue, which is defined as a shared object and is usable concurrently by all queue managers in the QSG.
- ▶ The loadbalancing algorithm is simple and does not take into account the actual workload in the channel initiator and queue manager.
- ▶ It is not possible (as it is for shared inbound channels) to set up a primary channel initiator for the outbound channel and only use other (secondary) channel initiators as a failover possibility.
- ▶ While a shared channel is running, it is managed by a single channel initiator in the QSG.

- ▶ The fact that the transmission queue is a shared queue means that messages sent through shared outbound channels must apply to the rules and restrictions of shared queues (for instance, max message size).
- ▶ The distribution of the channel connections in the sending end is completely transparent to the receiving end. And the receiving end only has to define one channel instance.
- ▶ When starting shared outbound channels (assuming that your queue managers and their associated channel initiators in the QSG are running), an attempt will be made to load-balance start the channel on any one of the channel initiators. It is therefore important that you define your channels as group objects so that they are available on all queue managers. An omission of a channel definition on one queue manager does not exclude the channel initiator for that queue manager from being the target for start channel. So you will not be able to let a *subset* of your queue managers in the QSG; for instance, if you have queue managers connected to IP stacks in your DMZ, process your shared outbound transmission queues related to outbound channels to external companies.

There is a way of targeting a start channel request to a specified queue manager and “fixing” a channel to that queue manager. However, in the unlikely event of queue manager or channel initiator failure, since the channel is fixed to the queue manager, no attempt is made to restart the channel on another available queue manager; the failing queue manager must be restarted before the channel itself can be started. Nevertheless, fix-started channels can be useful if you have a need to only ever start an instance of a channel on one of your QSG channel initiators, for example.

7.6 Intra-group queuing

WebSphere MQ for z/OS V5.2 introduced a mechanism called Intra-group queuing (IGQ). With IGQ messages may be transferred between queue managers in a QSG without the need to define private channels between these queue managers.

On each queue manager in the QSG an agent called the IGQ agent retrieves messages from a shared transmission queue called the `SYSTEM.QSG.TRANSMIT.QUEUE`. Messages that are put to this queue are tagged with a destination queue manager name. The agent running on the destination queue manager retrieves these messages and places them on the required target queue(s).

Figure 7-7 on page 142 illustrates intra-group queuing in use between two queue managers in a QSG.

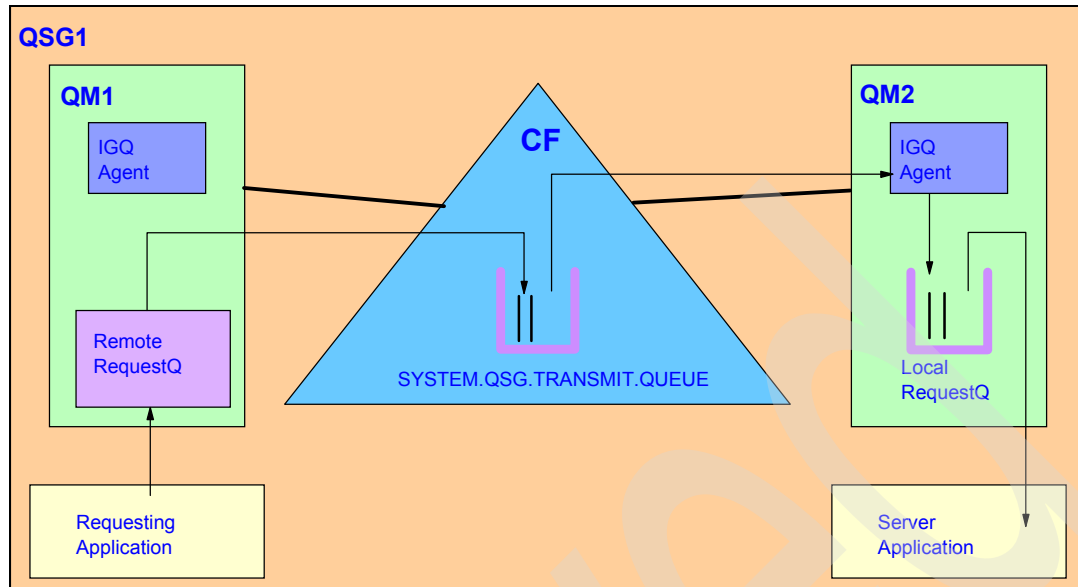


Figure 7-7 Intra-group queuing

With IGQ, administration costs are reduced since there is no need to define channels between the queue managers.

The flow is as follows:

1. The requesting application opens, and puts a query on to, the remote request queue (the remote request queue is defined with RNAME set to the name of the local request queue, RQMNAME set to QM2 and XMITQ set to blanks; assuming that IGQ has been enabled, name resolution resolves this to the SYSTEM.QSG.TRANSMIT.QUEUE).
2. During the put request, queue manager QM1 tags the message with the name of the target queue manager QM2.
3. The IGQ agent on QM2 retrieves messages that are tagged with queue manager QM2, and delivers them to the target queue. In this case, the local request queue.
4. The server application processes requests on the local request queue and generates a suitable response (though, the flow for the response message has not been shown in Figure 7-7).

The installation has to decide whether IGQ will be enabled or not.

- ▶ The usersids that are checked when IGQ is in use are controlled by the IGQAUT and IGQUSER queue manager attributes.
- ▶ If you have enabled IGQ, the queue manager will determine the eligibility of each individual message and send it to the SYSTEM.QSG.TRANSMIT.QUEUE or the usual transmission queue accordingly. Eligibility is decided by the restrictions for shared queues as the SYSTEM.QSG.TRANSMIT.QUEUE is a shared queue. So if you send a group of messages to a queue on another queue manager in the QSG, and it includes both eligible and non-eligible messages (for instance, with a message size varying around the max message size for shared queues), they will be split on two different transmission queues and transported through both IGQ and a channel.
- ▶ If IGQ is enabled, you cannot decide that all messages in a group must go either way.
- ▶ Depending on your needs for sending messages to private local queues on other queue managers in the QSG, you could face heavy load on this one shared queue in the

Coupling Facility. If you have these needs, you should instead consider migrating the private local queues to shared local queues.

- ▶ Non-persistent messages delivered by IGQ are always delivered out of syncpoint scope. IGQ does *not* support the equivalent of the function provided by the NPMSPEED(NORMAL) channel attribute.
- ▶ Non-persistent messages will be discarded if they cannot be delivered to either the destination queue or the dead-letter queue.

7.7 Benefits of using queue sharing groups

The use of queue sharing groups offers the following advantages.

High availability of queue managers

Because applications can connect to any queue manager in a QSG, and as all queue managers in a QSG can access shared queues, applications do not need to be dependent on the availability of specific queue managers. In the unlikely event of a queue manager failure, applications can continue to service shared queues by connecting to any remaining active queue manager in the QSG.

High availability and capacity of server applications

Cloned instances of server applications can be run on each z/OS image in the sysplex to provide higher server application availability and capacity. Each server instance can service request messages that are put to a shared request queue.

Scalability

New instances of server applications, or even new z/OS queue managers, can be easily introduced into the QSG to service shared queues and to run additional channels. Shared definitions, including group definition of non-shared objects, in the shared repository will be immediately available for any new instance or queue manager in the QSG.

Pull workload balancing

Based on the available processing capacity of each server instance in the QSG, “pull” workload balancing is naturally achieved for server applications. As each server instance can service request messages from a shared request queue, workload is balanced due to the immediate processing capability of the instance. No external workload balancing mechanism is needed.

Increased throughput of persistent messages

Persistent message throughput is limited by the rate at which data can be written to the log. Running multiple queue managers, each with its own log, can result in a significant increase in the throughput. By using shared queues, the increase in throughput can be made available through a single queue or through a set of queues.

Low cost messaging within the sysplex

It is not necessary to define and use WebSphere MQ channels to transfer messages between applications connected to different z/OS queue managers in the QSG. Instead, messages can simply be put to and retrieved from shared local queues.

Queue manager peer recovery

If a queue manager in the QSG fails and abnormally disconnects from the Coupling Facility, the remaining queue managers in the QSG are notified of this, and one of these remaining queue managers will complete any pending shared queue units of work for the failed queue manager. This maximizes the availability of messages and avoids the need to wait until restart of the failed queue manager.

Reduced administration

Because shared objects are stored in a DB2 shared database, after they are defined on one queue manager they become available to the entire group (hence, shared objects are also known as *group objects*). Similarly, any updates to shared objects need only be made on one queue manager in the QSG. And when a new queue manager is added to the QSG, shared object definitions are immediately available.

Increased channel availability

A shared channel is run by one of the channel initiators in the QSG. In the event of a failure affecting the communications subsystem, the channel initiator address space or the queue manager address space, the channel is restarted on one of the other QSG channel-initiators. It then re-synchronizes with its remote partner and continues message transmission.

Channel load-balancing

Outbound shared channels are run by a channel initiator chosen based on the number of channels each channel initiator is running, and the number of channels the channel initiator is configured to run. Inbound shared channels may be routed to one of the QSG channel initiators as determined by front-end routing software or hardware.

7.8 Initial considerations related to the use of QSGs

Before proceeding to exploit QSGs and shared queues, take into account the following considerations (additional issues are addressed in WebSphere MQ publications).

7.8.1 Technical setup

1. The number of QSGs you need to create for the application environment.

You may wish to set up different QSGs for production and test use, or define multiple production QSGs to achieve queue manager, and hence application, separation. By defining the appropriate level of access permissions at the QSG or queue manager level, you can control access to a QSG or to a given queue manager within a QSG.

2. As with queue manager names, you or your system administrator will need to derive a suitable naming convention for your QSG names.

Refer to *WebSphere MQ for z/OS Concepts and Planning Guide*, GC34-6051, for suggestions about naming conventions.

3. The number of queue managers in each QSG.

You will need to decide on the number of queue managers you need to run in each QSG to handle the application throughput.

As persistent message throughput is limited by the rate at which data can be written to the log, running multiple queue managers (each with its own log) in a QSG can significantly increase the message throughput.

The WebSphere MQ for z/OS V5.3 Performance Report (SupportPac™ MP1D) shows that a throughput rate of more than 11000 persistent messages (of size 1000 bytes) per second was achieved when running with four queue managers using a shared request queue and a shared reply to queue. This contrasts with a throughput rate of under 3000 persistent messages when running with a single queue manager in a QSG. Refer to the performance report for full details of the system configuration and design of the test application that was run to achieve these results.

4. Whether you want to have different classes of queue managers within a QSG, as described:
 - You may want to exploit queue sharing between queue managers connected to your internal LAN and queue managers connected to your demilitarized zone.
 - You may define one (or a subset) of your queue managers within a QSG to be gateway queue managers for communication to remote queue managers outside the QSG, by only running channel initiators on these queue managers.

5. The number of Coupling Facility list structures that the applications need.

A QSG requires a minimum of two list structures: the administration list structure, and an application list structure. However, for data isolation reasons, the application environment may demand that you define multiple application list structures, and Chapter 16 of *WebSphere MQ for z/OS Concepts and Planning Guide*, GC34-6051, lists other examples of why you may need more than one application list structure.

For performance reasons, it is recommended that you use as few CF application structures as possible.

If the applications issue MQGETs (say from a request queue) and MQPUTs (say to a reply to queue) within syncpoint, it is recommended that you use a single application structure for holding the messages associated with the two queues.

During tests, using a single queue manager with the request queue in a different application structure to the reply to queue, the unit CPU cost per reply/request was seen to increase considerably. However, with all MQGETs and MQPUTs out of syncpoint, the cost of using one or multiple application structures was seen to be the same.

6. The administration list structure has a fixed name of *qsg-name*CSQ_ADMIN (where *qsg-name* is the name of your QSG) but you will need to derive a suitable naming convention for the application list structures. They are also prefixed by *qsg-name*.

For example, you could use a name associated with the suite of applications that will use the shared queues whose messages are to be held in the list structure.

7. The size of your list structures.

It is recommended that the size of the administration structure is at least 10 MB.

The size of the application structures will depend on the size of your messages and on the maximum number of messages that will be concurrently held on the shared queues that map to the application structures. Of course, you will also be limited by the amount of available storage in the Coupling Facility; *WebSphere MQ for z/OS Concepts and Planning Guide*, GC34-6051, discusses this in more detail. You should also refer to the WebSphere MQ for z/OS V5.3 Performance Report (supportPac MP1D), which provides a simple formula for estimating the size of an application list structure.

8. After you have decided on the number and size of the application list structures, your systems administrator will need to define them to your Coupling Facility Resource Manager (CFRM) policy.
9. WebSphere MQ for z/OS V5.3 introduces CFSTRUCT objects that can be used to define the backup and recovery parameters for Coupling Facility application list structures. You can define/alter your CF structures to be recoverable, and use the BACKUP CFSTRUCT command

to back up persistent messages held in CF structures to logs. In the unlikely event of a CF failure, you can use the `RECOVER CFSTRUCT` command to recover persistent messages that had previously been backed up.

You can only back up those structures that are marked as recoverable. WebSphere MQ for z/OS V5.3 allows you to define/alter your CF structures to be recoverable. You should put persistent messages only to recoverable structures and non-persistent messages only to non-recoverable structures.

Refer to *WebSphere MQ for z/OS System Administration Guide*, GC34-6053, for further details.

10. You must decide and prepare the setup of shared channels and transmission queues.
11. You must decide and prepare the setup of initiation queues, and decide whether they should be shared; refer to 7.9.8, “Shared queue triggering” on page 154 for more information.
12. Other system-related queues:
 - You should not define your dead-letter queue (DLQ) as a shared queue because the maximum message size limitation for shared queues will prevent messages of length greater than 63 KB from being written to the DLQ.
 - You can define `SYSTEM.ADMIN.*.EVENT` queues as shared queues. Since the correlation identifier of each event message gives an indication of the originating queue manager, this may be useful in making some deductions about load balancing across the QSG.
 - You could define your `SYSTEM.DEFAULT.LOCAL.QUEUE` to be a shared queue so that subsequent queues based on the default are defined to be shared.
 - If you plan to use shared channels or intra-group queuing, you must define `SYSTEM.QSG.*` queues as shared queues.

Note: The asterisk (*) is used as a wild card. Refer to WebSphere MQ publications for details of the possible names for these queues.

7.8.2 Application development

1. Number of shared queues required by the applications, and the application list structure that each group of queues will map to.

Using a large number of queues in a single application list structure does not have any significant effect on the performance.

2. You will need to consider which of the following types of queues you want to define as shared queues:

- Request queues
- Reply queues
- Dynamic queues (possibly also being used as reply to queues).

For more information, see 7.9, “How applications can exploit WebSphere MQ shared queues” on page 147 and consider the items listed in 7.10, “Limitations and restrictions” on page 158.

3. You will need to consider the unit of work (UOW) scope and peer recovery. UOW scopes include
 - Local only UOWs - exist when messages are placed or retrieved, within syncpoint scope, from private local queues only.

- Shared only UOWs - exist when messages are placed or retrieved, within syncpoint scope, from shared local queues only.
- Mixed UOWs - exist when messages are placed or retrieved, within syncpoint scope, from both private local and shared local queues.

In the unlikely event of a queue manager failure, for:

- Local only UOWs, peer recovery is not possible, that is, another server cannot recover the UoW. The failing queue manager must be restarted to recover the local only UOWs.
- Shared only UOWs, peer recovery is possible.
- Mixed only UOWs, peer recovery is possible for shared messages only. To recover local messages, the failing queue manager must be restarted.

4. You will need to consider if the application has any message sequencing issues. For example:
 - Does the design of the applications allow you to run multiple instances of them against a shared queue?
 - Do the applications open queues for input exclusive use? Can they be changed to open queues for input shared so that multiple instances of them can be run against shared queues?
 - Do the applications issue keyed MQGET calls? Will they continue to work with shared queues given the limitations with shared indexed queues; see 7.10, “Limitations and restrictions” on page 158.
 - You may need to consider using serialized applications to preserve message sequencing; see 7.9.5, “Serialized applications” on page 151.
5. You will need to consider whether the application is to be cloned and your queue definitions should be identical, or if different class of service is required with different object attributes.
 - For cloned applications, you should define all objects as group objects so that the same definitions are made available to all queue managers within the QSG. This will enable the application instances to run against any queue manager within the QSG.
 - If different class of service is required, you may choose to vary the attributes of an object from queue manager to queue manager. This can be achieved by defining an object with the same name, but with different attributes, on each queue manager.

7.9 How applications can exploit WebSphere MQ shared queues

Existing MQSeries for OS/390 and WebSphere MQ for z/OS applications will run on a queue manager that is a member of a QSG without change. And they can even use shared queues and other group defined objects without changing the code.

However, to gain from the benefits of running queue managers in a QSG, you need to consider some of the issues highlighted by the following scenarios.

In the following scenarios, it is assumed that the applications follow a requester-server model. Requesting applications flow request messages into request queues and back end server applications perform database updates or inquiries and return response messages to the requesting applications via reply to queues. The requesting application waits for the response message to turn up in the reply to queue and picks it up when it arrives. Such a scheme works equally well whether the requesters:

- ▶ Have a direct connection to one of the queue managers in the QSG
- ▶ Use MQI channels to connect as clients directly into the queue manager
- ▶ Send the requests from queue managers outside the QSG using message channels

In any case, the server applications are running on z/OS or OS/390 logical partitions and connect directly to the QSG or one of the queue managers in the QSG.

7.9.1 Multiple cloned servers

In the simplest (and probably most often used) case, consider server applications that run as online applications (on MQGET, MQGMO=SYNCH parameter means that the message will be marked as not available to others but will be deleted from the queue only when you commit the MQGET). The server application flow could be:

- ▶ MQGET a request message from a request queue, in synch point (MQGMO=SYNCH).
- ▶ Perform a database update, in synch point.
- ▶ Construct and MQPUT a reply message to a reply to queue, in syncpoint.
- ▶ Issue a syncpoint with commit.

With one instance of a server application running in a logical partition, request messages would be serviced one at a time. With multiple instances of the server application running in the same logical partition more than one request can be serviced simultaneously but, with no increase in capacity this may not result in a significant improvement in the overall time to service requests. However, by making the following changes, this type of server application can very easily exploit the additional capacity, availability and dynamic workload balancing provided by QSGs and shared queues:

1. Define the request queue as a shared queue.
2. Run multiple cloned instances of the server application program on different logical partitions.

If a server, or queue manager instance should fail, messages on the shared request queue can still be serviced by the server application instances connected to the remaining queue managers in the QSG.

Compared to a local MQPUT to a private local queue, the shared queue MQPUT is slightly more expensive. Compared to a remote queue operation, where the request message should travel via a transmission queue and a channel, or even via Intra Group Queuing (IGQ), the shared MQPUT is much more efficient.

If the requester and server queue managers are in the same QSG, all of them can reach the shared queue directly.

You do not have to change the application code at all, whether you are using private or shared local queues, provided you are using ReplyToQ and ReplyToQmgr the right way:

- ▶ The requester must provide the desired ReplyToQ and leave the ReplyToQmgr blank in the message descriptor of the request message.

- The server must use the received ReplyToQ and ReplyToQmgr (now resolved to queue manager or QSG) in the object descriptor when sending the reply message. The server must not use remote queue definition for the reply to queue on the server queue manager.

If your reply to queue is a private local queue, the reply message will find its way back through a channel or IGQ according to the ReplyToQ and ReplyToQmgr settings.

Exactly the same thing will happen if your reply to queue is a shared queue and you do not change the ReplyToQ and ReplyToQmgr settings. But you may wish the reply message to be sent directly to the shared ReplyToQ. In order to make this happen, you must force the ReplyToQmgr to resolve to QSG name instead of the queue manager name. The technique is called “Reply to queue alias” and it works as follows:

- Define a QREMOTE (the reply to queue alias) with the reply to queue as the RNAME and the QSG name as the RQMNAME.
- Change (or develop) your requester so it uses this reply to queue alias as ReplyToQ and still leaves the ReplyToQmgr blank in the request message. WebSphere MQ will resolve according to the QREMOTE definition, thus supplying ReplyToQmgr with the QSG name.
- In order to get the reply message, the requester must (as usual) open the reply to queue. The reply to queue alias is a QREMOTE and can not be opened for get operations.

For a detailed example of migration, including considerations related to ReplyToQ and ReplyToQmgr, refer to 8.4.2, “Migration scenario” on page 172.

7.9.2 Replicated local queues with shared inbound channels

In the scenario presented in 7.9.1, “Multiple cloned servers” on page 148, the request queue was a shared queue. If you do not wish to convert a request queue to a shared queue, but you have your queue managers configured in a QSG, you could maximize on the capacity of your queue managers by running server applications on each queue manager. To achieve this:

- Define a private local request queue of the same name on each queue manager in your QSG. By defining the queue as a group object you only have to do it once and it will be available on any of the queue managers in the QSG.
- Define cloned server applications connecting to all (or a subset) of the queue managers.
- If the request messages arrive from message channels or MQI channels, configure your shared inbound channels to start through the generic port so that they are load balanced started across all (or a subset according to the presence of application clones) the available queue managers in the QSG, therefore allowing you to take advantage of the benefits of running shared channels.

Assuming that the inbound channels have been distributed across the available queue managers in the QSG, incoming connection requests could also be distributed across the available request queues.

Notes:

- ▶ A channel connection will stay on a particular queue manager until it is disconnected.
- ▶ You are able to define several shared inbound channels.

Depending on the setup of the front-end mechanism, you can choose to have different distribution patterns for different applications. And you may decide, for a given shared inbound channel, that it must only be directed to a subset of the available queue managers in the QSG because you have dependencies binding your server application to run on specific system images, or you want to restrict the execution of a particular application to a specific subset of system images for some other reason.

7.9.3 IMS Bridge

A particularly common type of server application makes use of the IMS Bridge. Requesting applications construct messages in a format suitable for running legacy IMS transactions, maybe using the MQIIH header to provide additional control of the transaction environment. These messages are placed on “IMS Bridge” queues.

IMS Bridge queues are linked by a WebSphere MQ storage class definition to a particular IMS control region. When messages arrive on the IMS Bridge queues, they are pulled off and transmitted via Cross System Coupling Facility (XCF) to the appropriate IMS regions by special IMS Bridge tasks inside the queue manager. The transactions run inside IMS and the responses that they generate are returned to the queue manager which places them on the appropriate reply to queues.

In a queue sharing group, the IMS Bridge can be configured for either capacity or availability, on a per queue basis.

Configuring an IMS Bridge queue for capacity

- ▶ Define the IMS Bridge queue to be a shared queue to allow it to be accessed by internal IMS Bridge tasks on each queue manager in the QSG.
- ▶ Define the queue with DEFSOPT(SHARED) and SHARE to allow multiple tasks to concurrently have it open for input.
- ▶ Ensure that the storage class definition on each queue manager targets a different IMS control region (ideally one which resides on the same logical partition as the queue manager).

Now each queue manager can run its own internal IMS Bridge task against the shared queue and pass messages from the shared queue to a back-end IMS in parallel with its peers. Within IMS, shared IMS message queues may be used, as required, to increase availability or solve issues relating to affinity of message processing regions to logical partitions.

Configuring an IMS Bridge queue for availability

For some applications it is not appropriate to run multiple servers against queues. This typically is the case where messages must be processed in strict sequence. The IMS Bridge can be configured for these types of applications as follows:

- ▶ Define the IMS Bridge queue as a shared queue, but with options DEFSOPT(NOSHARED) and NOSHARE. This restricts access to the queue, so that only a single task can open the queue for input at a time.
- ▶ Ensure that the storage class definition on each queue manager targets a different IMS control region.

Now only a single IMS Bridge task can run against the shared queue at a given time. But if the queue manager where the task is running should abnormally terminate, the other queue managers in the queue sharing group will be notified and another instance of the IMS Bridge task will be started to serve the shared queue.

7.9.4 Queue partitions

The issue of processing messages in strict sequence does not fit well in a shared environment where the greatest gains are made by having cloned servers working in parallel on a single queue. It is not possible to simply open the queue for `MQ00_INPUT_EXCLUSIVE` as that limits the servers on the queue to one, regardless of the number of logical partitions available.

A number of techniques are available to resolve this dichotomy, but each relies on the requesting application logically grouping the messages either by using a unique CorrelId for each group, or by using WebSphere MQ message grouping. Each server then works on a unique “partition” of the queue.

Partition by correlation ID

Both WebSphere MQ clustering and IGQ use the technique of partitioning by CorrelId.

In the case of clustering, the `SYSTEM.CLUSTER.TRANSMIT.QUEUE` is used to hold all messages outbound from the queue manager. Multiple sender channels start against the queue, and each only attempts to retrieve messages with a CorrelId that matches the channel name.

In the case of IGQ, the `SYSTEM.QSG.TRANSMIT.QUEUE` is used to hold messages that are destined for target queue managers within the same QSG. Multiple IGQ agents (one per queue manager) serve the queue. The agent on each queue manager retrieves messages with a CorrelId that matches the queue manager name.

This is a simple technique and works best when there are a few, well-defined logical partitions that can be individually handled by different parallel servers

Partition by logical group

The requesting application constructs logical groups of messages. Each group is assigned a unique GroupId and messages are sequenced within the group. The server application retrieves messages in logical order. Firstly, it retrieves the first message in any group (applications can optionally choose to work on complete groups only, that is, ones where all the messages for the group are already available on the request queue). Next, it retrieves successive messages in the group until the last message in the group has been retrieved, when it returns to finding any “first in group” message.

This technique is more complicated to implement from an application programming standpoint, but works well where individual requests are composed of many messages, maybe from different sources, or which have travelled by different routes.

7.9.5 Serialized applications

Any application working in strict sequence on a queue, and using shared queues for availability may be subject to the following race condition

1. Application instance 1 runs on queue manager A.
2. Either queue manager A, application instance 1 or both, abnormally terminate, causing a rollback of the current unit of work.

3. An external monitor (for example, ARM) notices the abnormal termination and reschedules instance 2 of the application on queue manager B in the same queue sharing group.
4. Application instance 2 restarts and performs MQGET from the queue BEFORE the rollback of the unit of work that was created by application instance 1 has completed. Instance 2 of the application may now have retrieved a message out of sequence.

In the simple case, where there is a single server application, opening the queue for input exclusive may be sufficient. In the case of an application failure, an open queue will not be closed until the rollback has been completed.

In the case of a queue manager failure, the rollback of the in-flight unit of work on the shared queue is performed by peer level recovery in a different queue manager; it, too, only removes the open interest in the shared queue after the rollback is complete.

However, if the application is using two-phase commit and the unit of work is in doubt at the time of failure, both mechanisms release the queue to new openers even though there is an uncommitted unit of work, which may rollback at some future time.

A more general issue is where multiple servers are working on different queue partitions and it is not possible to lock the entire queue to an application by opening the queue for input exclusive. The solution is to use a *serialized application*.

The concept of a serialized application is described as follows:

- ▶ The application issues an MQCONN call to connect to the queue manager. The application passes a serialization token of its choice on the connect call.
- ▶ Any recoverable changes performed by the application against a queue (that is, MQPUT or MQGET in syncpoint) are tagged with the serialization token.
- ▶ The tag is only removed when the unit of work has either been committed or backed out through normal unit of work completion, abnormal application termination, peer level recovery, or in-doubt unit of work resolution.

A second connection attempt using the same serialization token will be refused while queues are still tagged with the token.

Therefore, serialized applications can be used to protect against the race condition described above, and to preserve the sequence in which messages are retrieved from a queue. By ensuring that both instances 1 and 2 of an application use the same serialization token, instance 2 can be prevented from starting until instance 1 has either successfully completed, or has been completely recovered.

7.9.6 Shared queues and clustering

You can define shared queues to be cluster queues. Such a queue is advertised as being hosted by each queue manager in the QSG which is also a member of the cluster.

7.9.7 Inbound channels and target queues

Table 7-2 on page 153 lists the front-end and back-end availability for various configurations of inbound channels.

In order for an inbound message to reach its intended target-queue, either any one of the QSG queue managers is sufficient, or a particular queue manager is required. This requirement varies depending on the type of the target queue, and the type of the port the inbound channel uses. The term front-end availability is used to describe this requirement; that is, what resource is required to put the message to the target queue.

Similarly, to retrieve a message from a target queue either any of the QSG queue managers is sufficient, or a particular queue manager must be used. This requirement depends on the type of the target queue. The term back-end availability is used to describe this requirement; that is, what resource is required to get the message from the target queue.

Table 7-2 Configurations of inbound channels

No	Target queue	Port	Type of channel started	Front end availability	Back -nd availability	Comments
1	PrivLQ	prvP	Private	QMgr	QMgr	Same behavior as pre WebSphere MQ for z/OS V5.2 channels
2	ShrdLQ	genP	Shared	QSG	QSG	Natural shared local queue
3	PrivLQ	genP	Shared	QSG	QMgr	Replicated private local queues
4	ShrdLQ	prvP	Private	QMgr	QSG	Front-end sufficient
5	PrivLQ	grpP	Shared	QMgr	QMgr	Shared synchronization
6	ShrdLQ	grpP	Shared	QMgr	QSG	Front-end sufficient, shared synchronization

The number listed in column (No) correlates to the following points:

1. A message being targeted to a private local queue, through the private port, gives the same behavior as channels in pre-QSG versions of WebSphere MQ for z/OS. The front-end and back-end availability is dependent on the targeted queue manager being available.
2. A message being targeted to a shared local queue, through the generic port, is using shared queuing and channels in the optimum way. The front-end availability is at a QSG level since any queue manager in the group can be targeted with the conversation. The back-end availability is also at a QSG level since any queue manager in the group can access the message on the shared queue.
3. A message being targeted to a private local queue, through the generic port, is a valid configuration if the replicated local queue model is in use. The front-end availability is at a QSG level, but the back-end availability is dependent on the selected queue manager remaining available.
4. A message being targeted to a shared local queue, through the local port, would be used when the front-end queue manager is sufficiently available, but the back-end availability is required at a QSG level.
5. A message being targeted to a private local queue, through the shared port, is choosing to use the shared synchronization mechanisms, but availability is as for 1.
6. A message being targeted to a shared local queue, through the shared port, is choosing to use the shared synchronization mechanisms for 4.

Shared request and reply to queues

The example migration scenario (see Chapter 8, “Implementation and migration” on page 163) addresses some of the issues that you might encounter if you attempt to convert private local request and reply to queues to shared local request and reply to queues.

7.9.8 Shared queue triggering

The queue manager defines certain conditions as constituting “trigger events”. If triggering is enabled for a queue and a trigger event occurs, the queue manager sends a trigger message to a queue called an *initiation queue*. The presence of the trigger message on the initiation queue indicates that a trigger event has occurred.

The program that processes the initiation queue is called a *trigger-monitor application*, and its function is to read the trigger message and take appropriate action, based on the information contained in the trigger message. Normally this action would be to start some other application to process the queue which caused the trigger message to be generated. From the point of view of the queue manager, there is nothing special about the trigger-monitor application; it is simply another application that reads messages from a queue (the initiation queue).

For trigger types FIRST and DEPTH, shared queue triggering makes use of Coupling Facility list monitoring services to monitor for transitions in headers, known as *list headers*, in Coupling Facility list structures. Transitions occur when messages are written to shared queues that map to the Coupling Facility list structures. For trigger types FIRST and DEPTH, the current depth of a shared local application queue is monitored for a transition from 0 to 1, and from N to the value of trigger depth, respectively.

For trigger type EVERY, when a trigger monitor opens an initiation queue, each queue manager maintains a list of application queues that name the initiation queue.

Triggering using private local initiation queues

Figure 7-8 on page 155 illustrates the use of private local initiation queues to trigger start instances of a server application to serve a shared application queue.

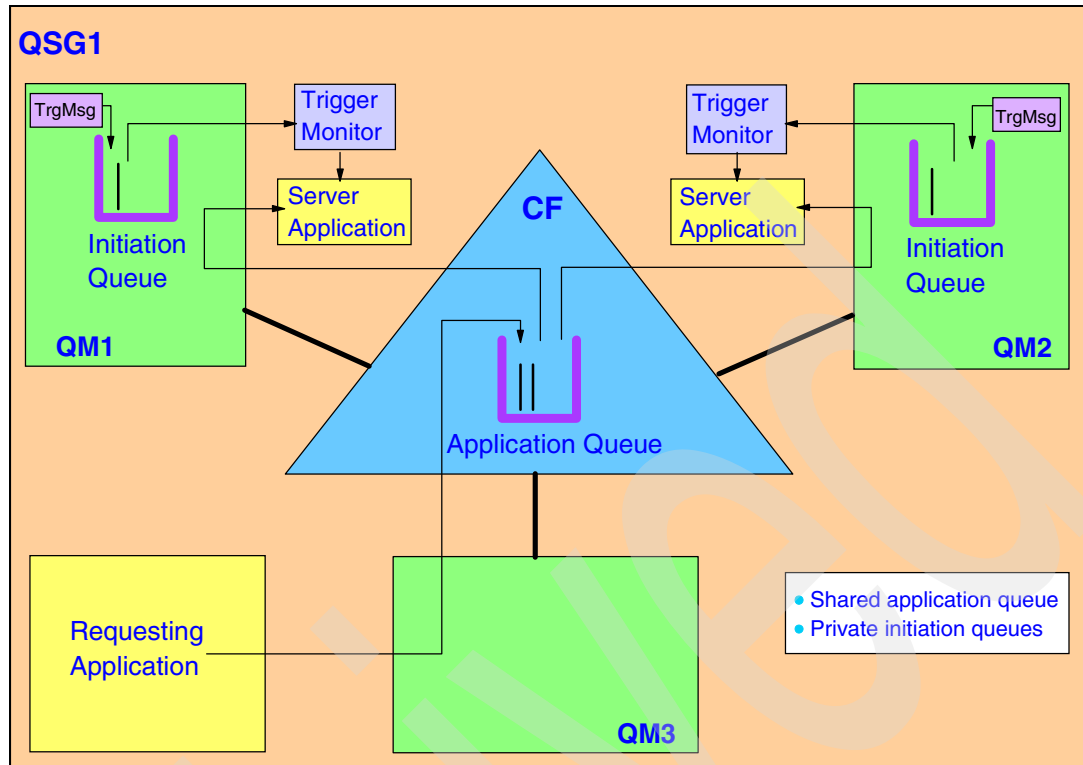


Figure 7-8 Using private local initiation queues to trigger start server application instances

Notes:

- For clarity, the initiation queue, trigger monitor, and server application have not been shown in the figure for queue manager QM3. The process objects on all queue managers are also not shown.
- The initiation queue and the process object can be defined as group objects.
- Keep in mind that many conditions need to be true in order for a trigger event to occur. The intention here is to explain the *general flow* for trigger starting applications to serve shared queues; providing explanations for all conditions is beyond the scope of this redbook.

A full list of the conditions can be found in WebSphere MQ publications.

- There are also special cases to consider when using triggering. Again, these are also described in WebSphere MQ publications.

The flow for trigger type FIRST or DEPTH is as follows:

1. The trigger monitor for each queue manager opens the private initiation queue for input. As part of the open request, each queue manager turns on list header monitoring for those shared local application queues that name this initiation queue.
2. An application puts a message on a shared local application queue.
3. Assuming that the put of this message results in a transition, each queue manager generates a trigger message and places it on the associated private initiation queue.

So, with trigger type FIRST or DEPTH, n trigger messages are generated, depending on the number of trigger monitors started.

4. Each trigger monitor retrieves the respective trigger message and starts an instance of the server application. Depending on the design of the server application (for example, cloned or not), one or more instances of the server application will be started.

The flow for trigger type EVERY is as follows:

1. The trigger monitor for each queue manager opens the private initiation queue. As part of the open request, the queue manager builds a list of those application queues that name this initiation queue.
2. An application puts a message on to a shared local application queue.
3. The queue manager on which this put is issued determines where trigger monitors are running and nominates one of the queue managers to generate the trigger message.
4. A trigger message is put to the private initiation queue of that queue manager.

So, with trigger type EVERY, one trigger message is generated.

5. The trigger monitor retrieves the trigger message and starts an instance of the server application.

If you do not want to trigger start an instance of the server application on one of the queue managers in the QSG, then you can simply stop the trigger monitor on that queue manager.

Note: For a shared queue defined with trigger type EVERY, when an application puts the first message on to the application queue, the queue manager that the application is connected to nominates one of the queue managers in the QSG to generate a trigger message. Only one trigger message is generated and it is put to the private initiation queue defined on the nominated queue manager.

If the trigger monitor serving the initiation queue, or the nominated queue manager, should fail before processing the trigger message, then the trigger message will remain unprocessed, or be lost (since trigger messages are non-persistent). An instance of the server application will not be started and a message will be left on the application queue.

To ensure that the trigger message is processed, and an instance of the server application started, you should consider using a shared initiation queue. Provided that a trigger monitor instance has been started on more than one queue manager in the QSG, in the event of failure, one of the trigger monitors should be able to process the trigger messages on the shared initiation queue.

The rules for trigger starting applications to serve shared application queues using a shared initiation queue are as described above when using private initiation queues. A point to note though is that for trigger type FIRST or DEPTH, if one trigger monitor instance is busy, this leaves the potential for less busy trigger monitors to process more than one trigger message from the shared initiation queue. Hence, multiple instances of the server application may be started against a given queue manager.

Note that these multiple instances are started as a result of processing multiple trigger messages. Ordinarily, for trigger type FIRST or DEPTH, if an application instance is already serving an application queue, another trigger message will not be generated by the queue manager that the application is connected to.

Notes:

- With trigger type FIRST or DEPTH, n trigger messages are generated and n server transactions are started depending, on the number of trigger monitors started against the initiation queue - even if there are less messages to process. With a low frequency of messages put on the application queue, you may have more server transactions than

messages to process. The application must be able to handle the situation where no messages are available.

Also consider the size of the wait interval of the MQGET carefully. It should be sufficiently large to keep the server application waiting for the next message, if it arrives within a reasonable time compared to the cost of starting a new application server instance.

- ▶ In CICS, you may have operational reasons for having only one trigger monitor (CKTI) running in each CICS subsystem instance. And the trigger monitor can listen to only one initiation queue. In this case you must decide if you want your initiation queue to be a private or a shared local queue. If most of the applications need trigger FIRST (or DEPTH), then you would probably choose a private initiation queue to support a better distribution of work between the system instances.

Another important consideration is that a private local initiation queue can also be used for private local application queues. So applications using shared queues can co-exist with applications using private (local or group defined) queues, cloned or not.

- ▶ A shared queue can refer to only one initiation queue name. Consequently, the initiation queues (if defined as private local queues) used by an application must have the same name in all the queue managers which participate in processing the application queue. The initiation queue may be defined as a group object.
- ▶ Process definitions may be defined as group objects.
- ▶ You must ensure that your transactions always reach MQOPEN of the triggering application queue. If your transaction ends normally or abnormally without opening the queue, a new trigger message will not be generated immediately to resume workload processing. This instance will be out of service and the rest of your clones must do the work until a new trigger condition is met on the particular queue manager. Whether from an expired trigger interval or because the queue depth reached zero, be aware that there will be a period with reduced workload processing—and it may be difficult to detect.

Service initiation layer (SIL) transaction

To avoid the shortcomings of trigger EVERY, but yet be able to trigger one transaction per message, consider establishing a SIL transaction with following characteristics:

- ▶ Initiated by standard trigger FIRST.
- ▶ Gets all messages in the request queue until it is empty. The MQGET must have a “balanced” wait interval, ensuring that it is kept running to process plenty of messages and that it will end when it is likely that there will be no work for a while.
- ▶ Starts a server transaction for each message, passing the complete message content to the transaction.

This offers the following advantages:

- ▶ You obtain full-blown pull workload balancing, thus ensuring that all system instances participate in the work.
- ▶ You do not have to define a queue for each transaction that you want to trigger. In fact, you can span from having one queue for the complete workload and to a very granulated degree, even with more queues per transaction.
- ▶ You may design (and protect) it to be able to start the transaction with the userid from the message descriptor, context information or just propagate the trigger monitors userid.
- ▶ You can extend the SIL transaction with more functionality (for instance, generating messages for tracing, debugging, problem determination and statistical purposes).

Note, however, the following preconditions and limitations:

- ▶ You must provide a header in each message saying which transaction to start (and possibly provide other information too, depending on the added functionality you want).
- ▶ You must develop (or change) the application to retrieve the message content from the communications area instead of getting a message.
- ▶ The transaction start implies a new UoW. Therefore, you cannot include the message get in the same UoW as the server transaction.

7.10 Limitations and restrictions

When exploiting sharing, you must be aware of a number of limitations and restrictions:

- ▶ With WebSphere MQ for z/OS V5.2, you can only write non-persistent messages to shared queues. With WebSphere MQ for z/OS V5.3, you can write both persistent and non-persistent messages to shared queues.
- ▶ As with z/OS queue manager names, QSG names are limited to up to 4 characters.
- ▶ You can define up to 32 queue managers to be members of a single QSG.
- ▶ Coupling Facility structure names can be up to 12 characters long and must begin with an alphabetic character.
- ▶ You can define a maximum of 512 Coupling Facility list structures per Sysplex.
- ▶ Each queue manager in a QSG can connect to up to 64 Coupling Facility list structures. One of these is the administration list structure so this leaves the possibility for up to 63 application list structures.
- ▶ A Coupling Facility list structure can hold data for up to 512 shared queues.
- ▶ The maximum message size for shared queues is 63 KB.
- ▶ You should not define a private local queue and a shared local queue with the same name. If you do manage to do this and you attempt to open either the private or shared version of the queue, then the MQOPEN (or MQPUT1) request will fail with reason code MQRC_OBJECT_NOT_UNIQUE. You will need to delete one of the definitions before you can successfully access the remaining definition.
- ▶ An MQINQ call for the IPPROCS and OPPROCS count of a shared local queue returns the count of applications that have the queue open for input or output, respectively, on the queue manager that issued the call. To obtain the count of applications that have the shared queue open for input or output throughout the QSG, you need to issue the MQINQ call on each queue manager in the QSG and essentially “total up by hand”.
- ▶ The models for shared dynamic queues have a DEFTYPE of SHAREDYN. Shared dynamic queues are created and destroyed in the same way as permanent dynamic (PERMDYN) queues. That is to delete them, you have to MQCLOSE them with the MQCO_DELETE* options.
- ▶ Queue indexing is limited for shared queues. If your application needs to retrieve messages by MsgId or CorrelId, then you must specify the corresponding index type for the shared queue. If you do not define a queue index, it is still permissible to select a message by message identifier but an attempt to select a message by correlation identifier will result in a failure with return code MQRC_CORREL_ID_ERROR.

Additionally, if your queue is indexed by GroupId, then you can only retrieve specific messages by specifying a GroupId. If you specify a MsgId and a GroupId, for example, then you will receive the next available message with a matching GroupId (the MsgId may well be different to that which was requested).

- ▶ You should not start a shared channel instance between two z/OS queue managers within the same QSG. If you attempt this then you may end up corrupting the shared channel status for this channel in DB2. Also, for load balanced starts, depending on which queue manager processes the start request, you may end up starting the sending end or the receiving end. This defeats the purpose of having shared channels.
- ▶ The maximum batch size (though this is not policed) that can be specified for a shared channel is 2650. This is due to the maximum size of message that can be written to the `SYSTEM.QSG.CHANNEL.SYNCQ` (which is defined as a shared queue). The typical batch size in use is 50 so, as it happens, having a limit of 2650 for shared channels should not be a cause for concern.

If you set the batch size for a shared channel to a value greater than 2650, then your channel may terminate abnormally with an error message that indicates that the MCA was unable to put a message to the `SYSTEM.QSG.CHANNEL.SYNCQ` for reason 2030 (MQRC_MSG_TOO_BIG_FOR_Q).

- ▶ Group attach is only supported for batch applications. So, your CICS and IMS transaction management systems should continue to connect to a specific, named queue manager that is on the same logical partition as the transaction management system.
- ▶ There is only one IGQ agent per queue manager.
- ▶ Non-persistent messages delivered by IGQ are always delivered out of syncpoint scope. IGQ does *not* support the equivalent of the function provided by the `NPMSPEED(NORMAL)` channel attribute.

7.11 Sharing and clustering

Exploiting queue sharing and shared channels does not prevent you from using clustering, and vice versa; these are two different technologies and are not mutually exclusive. There may be cases where you will use sharing, cases where you will use clustering, and cases where you will use both.

Do not consider clustering as an *alternative* to sharing. If you have a Parallel Sysplex, you should under any circumstances implement QSGs and a front-end routing mechanism to enable the generic port.

However, clustering offers advantages that are unavailable with QSGs alone:

- ▶ Several kinds of participants in the cluster

An MQ cluster can span queue managers on different processor types running different operating systems. And a particular queue manager can participate in several clusters. Queue sharing and shared channels are only available with QSGs on z/OS and OS/390, and are restricted to system images where a common DB2 sharing group is available.
- ▶ Easier serialization

With the “bind on open” feature, you can ensure that all messages put will go to the same queue on one queue manager. And if this queue is a private local queue, you can also ensure that the server application will process the message on the same queue manager and system image. Providing a similar functionality with shared queues demands a larger development effort, as described in 7.9.5, “Serialized applications” on page 151.
- ▶ Saved work defining channels

You do not have to define channels between all the queue managers in the cluster in order to be able to send messages to private local queues on other queue managers.

On the other hand, all this intra-cluster traffic flows through a single transmission queue on each queue manager, which may introduce a bottleneck. Alternatively, with sharing you could put directly to the shared queue, or use the IGQ to send the messages to another queue manager, if the messages are eligible due to the shared queue restrictions. Shared inbound channels between the queue managers in the QSG are also able to transport any kind of message, but this arrangement demands a larger effort to define and set up.

Clustering provides a simple workload balancing mechanism:

- ▶ If you put messages to a cluster queue and the queue is available locally to the putting application, the messages are put locally and the local server application clone will process the messages. This is good for performance, but it may not provide the workload balancing you want.
- ▶ If the cluster queue is *not* local to the putting application, the messages are put to one of the other clones based on availability and a simple round robin principle; it does not pay attention to the actual workloads on individual system instances. But you may develop a user exit to provide a more advanced algorithm.

QSGs provide a more advanced workload balancing mechanism, as follows:

- ▶ With shared inbound channels to a generic port you may, depending on your front-end mechanism, provide more advanced workload distribution algorithms at the front-end. And with pull workload balancing on shared queues in the back-end, you can let the actual workload in the back-end servers decide the optimal distribution.

Comparing clustering and QSGs is not a matter of private local queues versus shared queues. Clustering and QSGs can work on both:

- ▶ With shared inbound channels you can exploit workload balancing and still use private local queues containing messages of any kind or size.

If you use private local queues, you do not have the same level of availability for the application as for shared queues in case a failure should occur on a system image. The applications running on the failing image cannot recover immediately. You will have to wait until the system instance and the queue manager becomes available, or else forget about the transactions stuck with this queue manager.

- ▶ You can define a shared queue as a cluster queue. If you have a cluster consisting of for instance queue managers on different platforms, you can consider the QSG as one server by using a shared queue. This will be a very reliable server in the cluster. And you can even have more than one QSG in the cluster.

With shared queues, if a failure occurs on one system image and the queue manager is not available, then another queue manager can take over and resolve the UoWs (peer recovery), and users will not be affected.

In any case, to take advantage of clustering as well as sharing disciplines, you may not have to make changes to the WebSphere MQ specific code calls in the application. Yet, a precondition is that you are using ReplyToQ and ReplyToQmgr in the standard way. You must specify your ReplyToQ and leave ReplyToQmgr blank.

Conclusion and recommendations

As this redbook covers considerations exploiting the Parallel Sysplex, the following recommendations presumes that a Parallel Sysplex is provided and feasible:

- ▶ Establish one or more QSGs.
- ▶ Implement a front-end routing mechanism to distribute work dynamically between the queue managers using the generic port concept.

- ▶ Enable the applications to run several clones in parallel and support continuous availability. You probably need this regardless of whether you are using sharing and/or clustering. With respect to WebSphere MQ, you may not have to change your code at all.
- ▶ Implement shared inbound and shared outbound channels according to your needs.

Consider using clustering for the following reasons:

- ▶ You have several queue managers on different platforms and you want them to participate in the workload balancing and availability.
- ▶ You wish to simplify the channel definitions between related queue managers.
- ▶ You have not (and do not intend to) implemented a front-end routing mechanism, and you want to workload balance transfer of messages that are not eligible for shared queueing.

If you have already implemented clustering:

- ▶ If most or all of your queue managers are on z/OS or OS/390, consider whether it would be advantageous to replace clustering functionality with sharing functionality, or possibly exploit a combination of both.

7.12 Some final considerations

- ▶ Because the Coupling Facility is resilient to software failures, and can be configured to be resilient to hardware failures and power outages, messages stored in the Coupling Facility are highly available.

Non-persistent messages on private local queues are deleted following a restart of a queue manager. Non-persistent messages on shared local queues are not deleted from the CF list structures following a restart of a queue manager. Although most applications will probably view this as an advantage, if the applications have a dependency on non-persistent messages being deleted following a restart of a queue manager, you will need to write a clean up jobs to deal with these unprocessed non-persistent messages.

Also, it may not simply be a matter of clearing the queue. If the queue also has persistent messages on it, then you may not want these to be deleted.

One way of ensuring that you remove the unwanted non-persistent messages safely is to disable the queue for puts (this will prevent the arrival of any new messages on the queue), get all messages off the queue with the MQGMO_SYNCPOINT_IF_PERSISTENT option (this will ensure that all persistent messages are retrieved within syncpoint scope), and issue a backout (this will restore all persistent messages to the queue). You will of course need to ensure that the number of messages in your unit of work does not exceed the value of the MAXSMGS attribute for the queue manager that your clean-up application is connected to.

Another way is to disable the queue for puts, browse a message on the queue with the MQGMO_LOCK option, and destructively get the message if it is non-persistent. By repeating this process for all messages, you will be able to remove all unwanted non-persistent messages from the queue. Of course, if the clean-up application is the only application running (and there is only one instance of it) against the queue, then you do not necessarily need to use the MQGMO_LOCK option.

However, these solutions require a time interval where access to the queue is disabled, and they are inhibitors to 24x7 continuous availability. Try to avoid this situation, as follows:

- Try not to mix persistent and non-persistent messages in the same queues. Then you are able to decide on a per queue basis whether a queue may be emptied or queued.

- Use expiry in your non-persistent messages. But be aware that expired messages on indexed queues will stay on the queue until an indexed get with the same message id or correlation id is issued on the queue. WebSphere MQ for z/OS V5.3 provides a feature to automatically remove expired messages with optional time intervals (EXPRYINT). Before version 5.3, you must establish a process issuing a non-indexed get against each of these queues.
- Ensure that any shared queue with a queue depth greater than zero has an active server instance open to process the messages and empty the queue.
- ▶ Unlike triggering for private local queues, triggering for shared local queues works on committed messages only.
- ▶ If you do not already have a development and change management tool in place covering your WebSphere MQ objects, consider acquiring or developing one. It is important to have tools support to ensure that the applications setup will be efficient and consistent.
In addition to the usual benefits of a change tool, it may help to keep shared queue definitions within restrictions/limitations, exploit the advantages of group definitions, and create “helping objects” as reply to queue aliases.
- ▶ Manage CF usage. CF structures are a shared resource. No messages should rest in the list structures. And the max message size and the max queue depth must be held low. It is a good idea to keep track of max allocated space in the CF compared to max potential usage (max message size times max queue depth) for all shared queues. Naturally, you would probably never reach the max potential usage, but it may be convenient to keep track of the percentage and perhaps even state a max percentage before you will intervene.
- ▶ Access authority to shared queues is maintained in RACF the same way as for private local queues. When you are accessing a queue directly, your local userid’s authority is checked. If you send a message to a private local queue on another queue manager through a channel or via IGQ, you may not be able to check your local userid depending on your authorization settings.
- ▶ RACF authorization can be maintained on queue manager level, QSG level or both depending on the security checking profiles in the MQADMIN class.
 - If you have enabled queue manager level checking, it works exactly as if each queue manager was not part of a QSG. You must maintain your profiles queue manager by queue manager. But if you have a well-implemented naming standard for your queue managers, you may also use generic profiles where all or some of your queue managers are maintained through common, generic profiles.
 - If you have enabled QSG level checking, your security profiles must be maintained on QSG level. And you can avoid having any profiles on queue manager level.
 - If you have enabled both, both queue manager and QSG level checking will be done. The first check is done on queue manager level, and if you are not authorized access on this level, a check on QSG level will be done. It may rise an administrative problem that a RACF-rejection occurs (on queue manager level) each time you access a resource you are authorized to access (on QSG level).

Implementation and migration

In this we chapter discuss the migration issues for applications you already have up and running in a sysplex or non-sysplex environment, but which you have not yet migrated to exploit Parallel Sysplex capabilities.

We introduce a migration plan and discuss the approaches of a step-by-step migration, and of a complete migration at one time.

We cover the following topics:

- ▶ “Migration planning” on page 164
- ▶ “CICS” on page 164
- ▶ “DB2 data sharing implementation” on page 166
- ▶ “WebSphere MQ” on page 170

8.1 Migration planning

This section provides a high-level definition of the migration planning for selected environments, and also for migration of the test Parallel Sysplex.

8.2 CICS

Here we describe the stages required to migrate from a single CICS region running an application suite, to the application running in a sysplex environment and exploiting the features and benefits this offers.

The approach is to create the sysplex environment before exploiting it. We use an application to demonstrate how a sysplex environment can be created through a series of small steps.

Note: Your requirements may dictate that some of the steps we describe be executed in parallel with other steps, executed in a different order, or perhaps not executed at all.

8.2.1 Create the CICS sysplex environment

Before applications can run in a sysplex environment, the CICS environment must be a sysplex one. The following steps require no changes to the application code, or even investigation into its design. All of the changes detailed in this section can be made without change to any running application.

Prepare a TOR - AOR environment

Establishing a TOR - AOR environment is straightforward:

1. Create a new region that will become the TOR for the new environment, and then define CICS connections between it and the existing region. Because this region will not be physically running the applications, it does not require access to user application programs or additional resource definitions to those supplied by CICS (apart from any terminal definitions or models).
2. Define dynamic routing on the TOR to route all requests to the original region. This is usually achieved by using a product like CPSM, but a user-written routing exit could also be used (keep in mind, however, that this will result in a higher maintenance overhead as the environment changes).

Start using the new TOR-AOR Environment

By changing the applid used by the original region and reusing it as a generic VTAM Applid on the TOR, users will now log onto the TOR rather than the original region (which is now operating as an AOR). By reusing the original applid in this way, the method of accessing the system does not need to be updated, thus making the change transparent to end users. Also, using it as a generic applid rather than a specific one means that additional TORs can added later without updating this one.

The user-perceived increase in availability is achieved because should the AOR fail, the only users affected are those attempting to run transactions at the time of failure, or while the region is restarting. Because they are accessing the AOR via a TOR idle, users are not required to reaccess CICS through an AOR failure and so are unaware the restart has taken place.

RLS-enable files

To enjoy the most reliable Parallel Sysplex environment, VSAM files used within the CICS regions need to be RLS-enabled so that multiple regions can update the same file without compromising the integrity of the data. To achieve this, an SMSVSAM server must be defined and running on each image in the sysplex, and the VSAM base files must be SMS-managed in a data class allowing RLS access. The AOR region needs to be started with the System Initial is at ion Parameter “RLS=YES” and the CICS file definitions need the attribute “RLS=YES”.

RLS support is not available in CICS/ESA® 4.1 and earlier releases, so for those releases you must use a Resource Owning Region (ROR) to access the files.

Sysplex-enable the database environment

This is covered in other sections so is not replicated here.

Create CICS Coupling Facility servers

Information on how to define and run these servers is defined in *CICS System Definition Guide*, and one server of each type is required on each image. Following is a brief description of how to exploit them without knowledge of the applications:

- ▶ Coupling Facility Data Tables (CFDT)

Because data held in a CFDT is not reflected back to any underlying dataset, these are similar to Use Maintained Tables. However, they are not forward recoverable and have a smaller maximum key length.

- ▶ Shared Temporary Storage

Without knowledge or analysis of applications, it is impossible to determine which queues need to be shared and which must not be shared.

- ▶ Named Counter server

Unless applications are already using a Named Counter Server, this server cannot be utilized until the applications are modified.

Create Resource Owning Regions

Create an additional CICS region that has connections from the AOR region. Because this type of region is a single point of failure in a sysplex environment, you should consider using CICS eXtended Recovery Facility (XRF), with the primary and alternate running on different images.

For resources held by a Shared Temporary Storage server, without knowledge of applications the use of Temporary Storage or Transient Data Queues defining them as remote on the AOR and held on the Resource Owning Region cannot be undertaken. However, non-RLS files can be moved to this region without application knowledge.

This step will have to be performed as a single step unless there are known to be no affinities in the application.

Clone the TOR

Because the existing TOR is simply routing work to the AOR, one or more identical copies of this region (apart from its VTAM Applid and CICS Sysid) can be created and added to the environment without regard to the application design. An additional TOR with a VTAM generic applid will mean that users can log onto what appears to be the same image, with VTAM balancing the sessions and potentially reducing network congestion.

Clone the AOR

Cloning the AOR region completes the final stage in the sysplex enablement of the CICS environment. Until the application has been analyzed and the potential for dynamically balancing transaction requests has been established, these additional regions will not be processing any work.

8.2.2 Exploit the sysplex environment with the applications

- ▶ An affinity analysis, CICS region topology, and CPSM implementation
- ▶ CICS/VSAM Record Level Sharing includes a detailed implementation

Before any migration or changes to the environment or the application are attempted, the application needs to be analyzed to determine affinities and to determine whether the different areas of the application can be migrated to a sysplex environment at different times or by different teams of developers. Tools such as the CICS Affinities Utility, along with documentation and papers mentioned previously, can assist with this task.

8.3 DB2 data sharing implementation

Here we present a high-level description of the steps involved in implementing a DB2 data sharing environment. For a more detailed implementation plan, review *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417.

8.3.1 Plan a naming convention

Because you must name a large number of items, and might have to add new members or move existing members to different systems in the future, managing all these items is easier if you choose and maintain a consistent naming convention. A consistent naming convention can also help you avoid confusion, eliminate operational error, and simplify problem determination.

Note: Keep in mind that, after installation, you cannot change some names (such as the group name and member names).

You must provide names for the data sharing group, the members of the group, and the IRLM group.

Data sharing group names

The following names are shared by all members of the group:

- ▶ DB2 group name
- ▶ ICF catalog alias
- ▶ Group attachment name
- ▶ SQL port number
- ▶ Location name
- ▶ Generic LU name
- ▶ Parallel Sysplex domain name

Member names

The following names must be unique within the data sharing group:

- ▶ Member name
- ▶ DB2 subsystem name (must be unique within the Parallel Sysplex)

- ▶ LU name
- ▶ Member domain name
- ▶ Resynchronization port
- ▶ Command prefix
- ▶ Work file database (DSNDB07 is not shared)

Each member will have its own active log data sets, archive log data sets, and BSDS data sets. These data sets require a master catalog alias as the high-level qualifier (HLQ) to point to the catalog where the data sets are cataloged.

One common convention is to begin the data set names with the ICF catalog alias and to use the member name as qualifiers, such as *catalias.memname.xxxxx*. If you choose the ICF catalog alias to be the group name, then the format would be *grpname.memname.xxxxx*

IRLM names

- ▶ Group name
- ▶ Subsystem name
- ▶ IRLM procedure name
- ▶ IRLM member ID

Coupling Facility structure names

There is a strict naming convention for Coupling Facility structures that use the DB2 group name as the high-level qualifier:

- ▶ Lock structure name - *groupname_LOCK1*
- ▶ Shared communications area - *groupname_SCA*
- ▶ Group buffer pool names - *groupname_GBPxxxx*

GBPxxxx is the name of the group buffer pool. Your DB2 for z/OS V8 data sharing group will have at least GBP0, GBP8K0, GBP16K0, and GBP32K, as these are required for the catalog and directory.

Naming convention suggestions and a naming example are provided in *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417.

8.3.2 Planning for availability

The Coupling Facility allocates storage for the group buffer pools, the lock structure, and the SCA. These structures are defined in the coupling facility resource management (CFRM) policy for the Parallel Sysplex. The CFRM policy is managed by the z/OS systems programming staff, as it is a key component of the Parallel Sysplex and contains several other structures that are for use by facilities other than DB2.

Sizing the DB2 structures

Each structure in the CFRM policy should be defined with an INITSIZE parameter, which indicates the size of that structure at initial allocation. There should also be a SIZE parameter which limits the maximum size to which the structure can be dynamically modified.

The sizes of the structures are based on the particular characteristics of your DB2 data sharing group. Refer to *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417, which has several formulas and general guidelines to help you determine reasonable sizes. Some of these characteristics include the amount of data sharing between the members and the amount of update activity that is expected. Another factor that affects GBP sizing is the GBPCACHE value for the table spaces, indexes, and partitions that will be cached in that GBP.

XES Auto Alter function

If the Coupling Facilities are at CFCC level 12 or higher and there is adequate storage in the CFs, the parameter ALLOWAUTOALTER(YES) should be considered in the GBP structure definitions in CFRM. This provides dynamic adjustment of directory to data page ratios without rebuilding the structures as the workload changes. It can increase the size of directory entries, data elements, or both, if a shortage of storage exists.

Duplexing structures

Duplexing is key for high availability, as it eliminates a single point of failure. GBP duplexing allocates two copies of the same GBP in two different Coupling Facilities. Changed data pages are written to both structures, but only the primary structure is used to read data pages, register pages, and perform cross-invalidation of pages in the local buffer pools of the members of the data sharing group. Data pages are cast out to DASD only from the primary structures. The secondary structures only store copies of the changed data pages, so there is minimal overhead. Storage should not be an issue, since simplex structures require that you have enough space in the surviving CFs to be able to allocate the structures of any failed CF. Duplexed structures are using that reserved storage to hold the secondary group buffer pools.

The lock structure and SCA can also be duplexed, but this is not usually recommended. These structures can be quickly rebuilt, so duplexing does not provide a significant availability improvement. In addition, in most cases, the data sharing overhead of duplexing the lock structure and SCA will outweigh the availability benefits. The simplex lock structure and SCA should be allocated on outboard CFs. This can be an ICF on a zSeries machine that holds no members of the data sharing group or a stand-alone CF. The failure of a machine that has both an LPAR with a data sharing member and an ICF that has these two structures will cause a group outage and require a group restart to bring back all the members.

Automatic restart and DB2 Restart Light

The locks held by a failed member of the data sharing group will become retained locks that could affect the applications that are executing on the surviving members. The process of releasing the retained locks is accomplished by restarting the failed DB2 member, either on the z/OS system on which it failed, or on another system in the same Parallel Sysplex. Therefore, it is important to implement the component of z/OS that manages automatic restarts, Automatic Restart Manager (ARM).

If you are restarting a failed DB2 on a different z/OS image, you may not have the necessary resources to support that member's storage and processing requirements. Your goal is to have the failed member perform the restart process in order to release the retained locks, but not to execute any new work.

DB2 Restart Light provides an option to restart with minimal-sized buffer pools, no EDM or RID pools, reduced number of service tasks, and several other optimizations. If there are in-doubt units of recovery (UR) at the end of the Restart Light, the DB2 member will not shut down, but remain running so that the in-doubt URs can be resolved. It will shut down after the in-doubt URs have been resolved. That DB2 member can then be restarted normally.

8.3.3 Migrating DB2 applications

SQL used in applications in non-data sharing will run when those applications are moved into a data sharing environment.

However, for a data sharing environment to run efficiently, applications must be well-designed and well-behaved. Chapter 3, "DB2 application considerations" on page 33 describes the application design and coding techniques that should be used in all new applications that will

run in a data sharing environment. Existing applications should be reviewed and appropriate enhancements made before migrating them to a data sharing environment.

8.3.4 Monitoring DB2 data sharing

There are several monitoring tools and DB2 commands which should be used to confirm that the data sharing group is performing within acceptable guidelines. Here we provide brief descriptions about some of these facilities. Detailed information and examples can be found in *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417.

RMF reports

Many DB2 staff members have limited experience with the reports available from the resource management facility (RMF). However, the following reports contain important system information and should be reviewed:

- ▶ The Coupling Facility Activity Report provides information that should be reviewed by both the DB2 staff and the z/OS systems programmers.
- ▶ The Storage Allocation Report allows you to make sure that the structures are allocated on the correct Coupling Facility and that there is sufficient storage available should one CF fail.
- ▶ The Structure Activity Report shows the response time for requests and, for the lock structure, the rate of total contention as well as false contention.
- ▶ The Subchannel Activity Report confirms that you have at least two links from each LPAR to each CF and indicates whether additional links should be added based on the number of delayed requests.

DB2 Performance Expert (DBPE)

Traditional reports and traces for a non-data sharing environment are now called *member scope* when reporting on an individual member of a data sharing group. *Group scope* reports and traces merge instrumentation data from the individual members, and presents it for the entire data sharing group. Details about locking and group buffer pool activity are available in the group scope statistics report.

For current information on DB2 PE, refer to *IBM DB2 Performance Expert for z/OS Version 2*, SG24-6867.

DISPLAY GROUPBUFFERPOOL command

Output from this command can be at the summary level or detailed level for either the group or individual members. The syntax and descriptions of the options for this command are available in *DB2 UDB for z/OS Version 8 Command Reference*, SC18-7416.

Two key values to review in the output from this command are:

- ▶ Write failures due to lack of storage
- ▶ Cross-invalidations due to directory reclaims

A non-zero value for either or both of these statistics indicates that the group buffer pool is too small and should be enlarged.

DISPLAY GROUP command

The output from this command includes:

- ▶ The DB2 group name and group release level
- ▶ The member names and release levels

- ▶ The group attachment name
- ▶ The IRLM subsystem names to which members are connected
- ▶ The command prefix for each member
- ▶ The status of each member
- ▶ The MVS system names where the member is running
- ▶ The procedure names of the connected IRLMs
- ▶ The SCA structure size in kilobytes and the percentage currently in use
- ▶ The lock structure size
- ▶ The maximum number of lock entries possible for the lock table
- ▶ The maximum number of modify lock list entries
- ▶ The number of modify lock list entries currently in use

Logical page list (LPL) recovery

The logical page list (LPL) contains a list of pages (or page range) that could not be read or written for some reason, such as transient disk read and write problems that can be fixed without redefining new disk tracks or volumes.

Specific to data sharing, the LPL also contains pages that could not be read or written for must complete operations, such as a commit or a restart, because of some problem with the Coupling Facility. The LPL is kept in the SCA and is thus accessible to all members of the group.

DB2 will automatically attempt to recover pages added to LPL, with the following exceptions:

- ▶ DASD I/O error
- ▶ During DB2 restart and end_restart time
- ▶ GBP structure failure
- ▶ GBP 100% loss of connectivity

To check for the existence of LPL entries, use the `DISPLAY DATABASE RESTRICT` command.

LPL recovery will begin when you start the object by using the `START DATABASE` command.

8.4 WebSphere MQ

In the following sections, we describe WebSphere MQ considerations.

8.4.1 Migrating your existing applications to use shared queues

In order to migrate your existing applications to use shared queues, you will need to do the following:

1. Migrate your existing private local queues to WebSphere MQ for z/OS V5.2/V5.3 shared queues, or migrate your existing WebSphere MQ for z/OS V5.2 shared queues to WebSphere MQ for z/OS V5.3 shared queues.

If you have messages on your existing queues, then you will also need to move these. However, given the maximum message size and queue index restrictions that apply to shared queues (see 7.10, “Limitations and restrictions” on page 158), you may not be able to move the messages on some of your private local queues.

The procedure for moving messages from private local queues to shared local queues is described in *WebSphere MQ for z/OS System Administration Guide*, GC34-6053.

2. Modify your existing applications (if required) to check for and handle any new reason codes that can be issued on WebSphere MQ API calls. Some new reason codes that may be issued on WebSphere MQ for z/OS are:

- MQRC_STORAGE_MEDIUM_FULL (but this has the same value as MQRC_PAGESET_FULL),
- MQRC_DB2_NOT_AVAILABLE,
- MQRC_OBJECT_NOT_UNIQUE,
- MQRC_CF_NOT_AVAILABLE
- MQRC_CF_STRUCTURE_NOT_AVAILABLE
- MQRC_WRONG_CF_LEVEL,
- MQRC_CF_STRUC_FAILED

Refer to the description of each WebSphere MQ API call, as provided in *WebSphere MQ Application Programming Reference*, to determine the full list of new reason codes.

3. Identify affinities (if any) that applications may have to the following:

- a. A given queue manager

For example, you may have batch applications that are coded to connect to a given queue manager, or that connect to the default queue manager.

If you are planning to run multiple cloned instances of the batch application, consider changing the applications to connect to the QSG name, or consider changing the name of your default queue manager to be the name of a default QSG. Be aware, however, that if you have more than one queue manager on the same system image (and in the same QSG), connecting to the QSG will direct you to either of these queue managers.

The reason for having more than one queue manager on the same system image could be that you need to reach both your internal LAN and the DMZ—and that cannot be accomplished from the same queue manager.

- b. Resources owned by a given queue manager

For example, the applications may be using private WebSphere MQ object definitions defined to a given queue manager.

If you want to ensure that the applications can run on any queue manager in the QSG, then you should ensure that the WebSphere MQ object definitions required by the applications are available on all the queue managers in the QSG. One way of achieving this is to use group objects.

- c. The way resources owned by a given queue manager are accessed

For example, you may have server applications that open private local request queues with the MQ00_INPUT_EXCLUSIVE option.

If you are planning to run multiple, cloned instances of your server applications simultaneously, then ensure that your server applications open shared local request queues with the MQ00_INPUT_SHARED option.

Of course, if you want to ensure that only one server instance can access a shared local request queue at a time, then your server application could open the request queue with the MQ00_INPUT_EXCLUSIVE option. Alternatively, you need to ensure that you only ever start one instance of the server application at a time (refer to 7.9.5, “Serialized applications” on page 151).

- d. A transaction manager, like CICS or IMS, that connects to a given queue manager

If you are planning to create a QSG with cloned queue managers, then you should ideally have “cloned” transaction managers that connect to each of these queue managers. So, for example, it should be possible to run instances of your transactions on each transaction manager. Additionally, if you are using triggering, then you should have trigger monitors started on each transaction manager.

A shared queue implementation may require application changes to address any such affinities and take advantage of parallel processing.

4. Identify affinities that your channels may have to:

- Private transmission queues

If you wish to benefit from running shared sender channels, then you will need to ensure that your sender channel definition refers to a shared transmission queue. If you have converted your private transmission queue to be a shared transmission queue, then this will not be an issue.

- Specific port numbers

If you have sender, server, or fully qualified requester channels that today specify a local port in the connection name field of the channel definition, then you will need to change this to refer to the group port or to the generic port if you want to take advantage of running shared inbound channels.

5. Ensure that the applications do not generate messages that are greater than the maximum length of messages for shared queues. Otherwise, puts to shared queues will clearly fail.

6. Make note of all the shared queues limitations; see 7.10, “Limitations and restrictions” on page 158.

8.4.2 Migration scenario

The following scenario illustrates how a simple query application can be converted to use queue managers in a QSG and shared local queues, in place of distributed queuing and private local queues.

Figure 8-1 on page 173 illustrates an simple query/response scenario. The scenario consists of a client application connected to a front-end z/OS queue manager QM1, and a server application connected to a back-end z/OS queue manager QM2. QM1 and QM2 are connected by WebSphere MQ channels. The client submits a query to the server. The server processes the query and returns a response to the client.

Notes:

- ▶ For simplicity, the client connects directly to QM1. The client may well be connected to a remote queue manager, QM3, running on a distributed server (for example, AIX®), with QM3 being connected to QM1 via WebSphere MQ channels.
- ▶ This example assumes prior knowledge of WebSphere MQ. Refer to WebSphere MQ documentation for specific details.

The queues defined on QM1 are:

- ▶ QUERY.REQUEST, an alias queue definition, with TARGQ(QUERY.REMOTE).
- ▶ QUERY.REMOTE, a remote queue definition, with RNAME(QUERY.REQUEST), RQMNAME(QM2), XMITQ().
- ▶ QM2, a private local queue definition, with USAGE(XMITQ) - the transmission queue that is used to transfer queries to queue manager QM2. Transmission queue QM2 is served by a WebSphere MQ sender channel that has a partner receiver channel on QM2.
- ▶ QUERY.RESPONSE, an alias queue definition, with TARGQ(QUERYRESP).
- ▶ QUERYRESP, a private local queue definition.

The queues defined on QM2 are:

- `QUERY.REQUEST`, an alias queue definition, with `TARGQ(QUERYREQ)`.
- `QUERYREQ`, a private local queue definition.
- `QM1`, a private local queue definition with `USAGE(XMITQ)` - the transmission queue that is used to transfer responses to queue manager `QM1`. Transmission queue `QM1` is served by a WebSphere MQ sender channel that has a partner receiver channel on `QM1`.

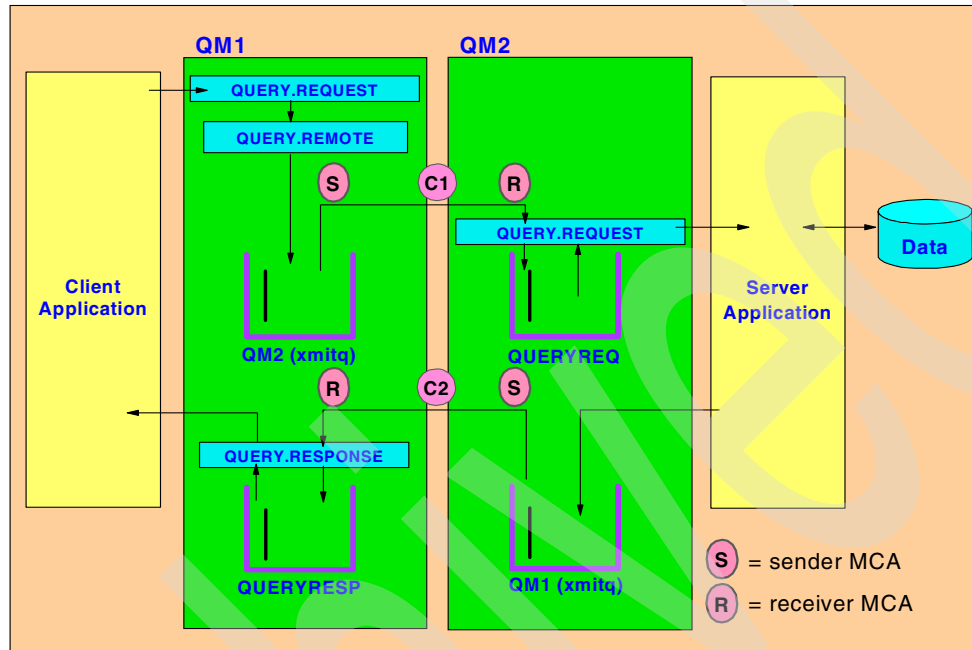


Figure 8-1 Simple query/response scenario

The query/response flow is as follows:

1. The client application puts a query on to `QUERY.REQUEST` (name resolution resolves this to transmission queue `QM2`).
On the put request the client application sets the `ReplyToQ` and `ReplyToQMgr` fields of the MQMD to `QUERY.RESPONSE` and blanks, respectively. Hence, the queue manager sets the `ReplyToQMgr` field to `QM1`.
2. The sender MCA for channel `C1` transmits the query to the partner receiver MCA which puts it on to `QUERY.REQUEST` (name resolution resolves this to `QUERYREQ`).
3. The back-end server application retrieves the query, processes it, generates a response, and places the response on the associated reply to queue (name resolution resolves this to transmission queue `QM1`).
4. The sender MCA for channel `C2` transmits the response to the partner receiver MCA which puts it on to `QUERY.RESPONSE` (name resolution resolves this to `QUERYRESP`).
5. The client application retrieves the response from `QUERY.RESPONSE` (name resolution resolves this to `QUERYRESP`).

Figure 8-2 on page 174 illustrates the same scenario, but with `QM1` and `QM2` in a QSG and with shared local queues. Notice that there is no longer a need to define channels between the queue managers, and that there is no longer a need to use remote queues and transmission queues. Also note that both the request and response queues have been converted to shared queues. If the client has been using a private dynamic queue for a reply to queue, then this could have been converted to a shared dynamic queue, or alternatively just left as a private dynamic queue.

The queues defined on QM1 and QM2 are:

- ▶ QUERY.REQUEST, an alias queue definition, with TARGQ(QUERYREQ). Defined as a group object.
- ▶ QUERYREQ, a shared local queue definition.
- ▶ QUERY.RESPONSE, an alias queue definition, with TARGQ(QUERYRESP). Defined as a group object.
- ▶ QUERYRESP, a shared local queue definition.

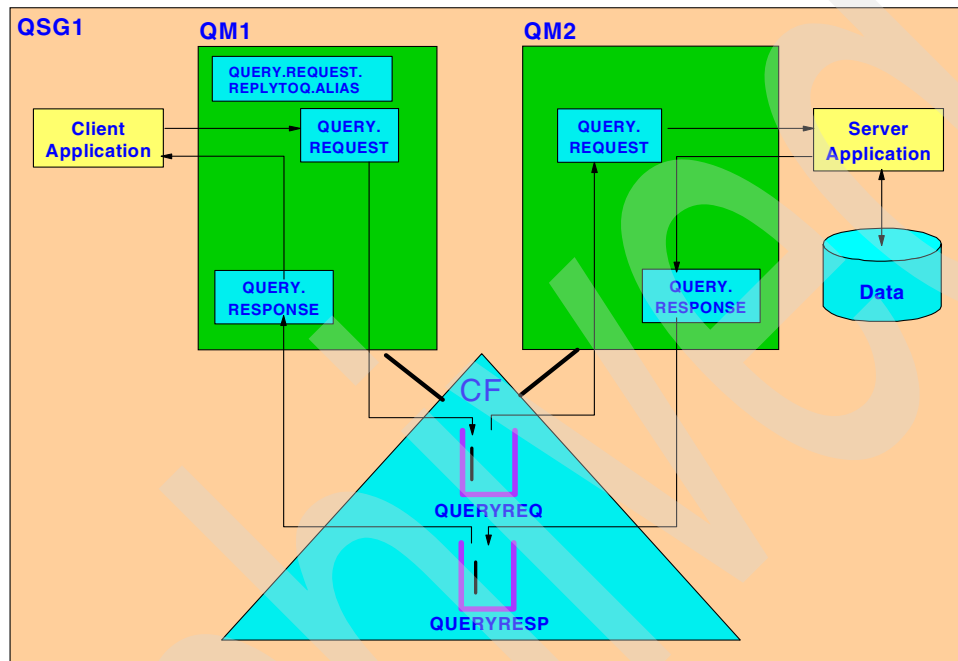


Figure 8-2 Simple query/response scenario using shared queues

The query/response flow is as follows:

1. The client application puts a query on to QUERY.REQUEST (name resolution resolves this to QUERYREQ).
On the put request, the client application sets the ReplyToQ and ReplyToQMGr fields of the MQMD to QUERY.RESPONSE and blanks, respectively. Hence, the queue manager sets the ReplyToQMGr field to QM1.
2. The back-end server application retrieves the query, processes it, generates a response, and places the response on the associated reply to queue i.e. QUERY.RESPONSE at QM1. Name resolution would resolve this to transmission queue QM1 but, we no longer have transmission queue QM1. Ideally, the server application should put the response to QUERY.RESPONSE at QM2 since name resolution would resolve this to QUERYRESP. This in turn would avoid any unnecessary distributed queuing. There are a number of alternative ways of achieving this:
 - a. Change the client application so that it specifies ReplyToQ QUERY.RESPONSE and ReplyToQMGr QSG1. The server would then place the response on QUERY.RESPONSE at QSG1, which would result in the response being put to shared queue QUERYRESP.
However, this is not ideal as many applications are probably (and should be) designed to leave the ReplyToQMGr blank so as to allow the queue manager to fill in the field.
 - b. Define a queue manager alias definition (which is really a remote queue definition) on QM2 that maps QM1 to QM2. I.e. Define QREMOTE(QM1) with RNAME(), RQMNAME(QM2)

and XMITQ(). So that, when the server puts the response to QUERY.RESPONSE at QM1, this actually resolves to QUERY.RESPONSE at QM2.

However, if there was a need to define private channels as well as to configure queue managers in a QSG, then this method would not allow you to define a transmission queue called QM1 on QM2.

- c. Define a reply to queue alias definition (which is really a remote queue definition) on QM1 that maps the ReplyToQMgr to the QSG name. Define QREMOTE(QUERY.REQUEST.REPLYTOQ.ALIAS) with RNAME(QUERY.RESPONSE), RQMNAME(QSG1) and XMITQ().

Change the client application so that it specifies ReplyToQ QUERY.REQUEST.REPLYTOQ.ALIAS and a ReplyToQMgr of blanks. In this way, the reply to queue information in queries is resolved at the time the client issues the put request to QUERY.RESPONSE at QSG1. Now when the server puts the response to QUERY.RESPONSE at QSG1, the response is actually put to QUERYRESP.

The client application would continue to process responses from QUERY.RESPONSE on QM1.

This method is better as it resolves the reply to queue information at the source. However, the client application must specify different reply to queue information in the query, and in the request to open the reply to queue for subsequent processing. If client applications currently make use of ReplyToQ alias definitions, this should not be much of a problem. Otherwise, it will be necessary to change the client application to avoid unnecessary distributed queuing (via private channels) between queue managers QM1 and QM2.

3. The client application retrieves the response from QUERY.RESPONSE (name resolution resolves this to QUERYRESP).

Note: To benefit fully from increased capacity and availability, the server application should be enabled to run on queue manager QM1, so that instances of client applications can connect into either queue manager for service.

This scenario demonstrates how shared queues could be used to avoid the need to define transmission queues and channels. If the front-end queue manager QM1, shown in Figure 8-1 on page 173, had been running on a distributed server (for example, an AIX server), and the back-end queue manager QM2 had been running on z/OS, then an increase in capacity at the back-end could be exploited as illustrated in Figure 8-3 on page 176.

Figure 8-3 on page 176 illustrates how the simple query/response scenario can exploit the back-end capacity and availability. The client application connects to QM1 on AIX and submits a query to the back-end QSG. The cloned server application on one of the QSG queue managers processes the query and returns a response.

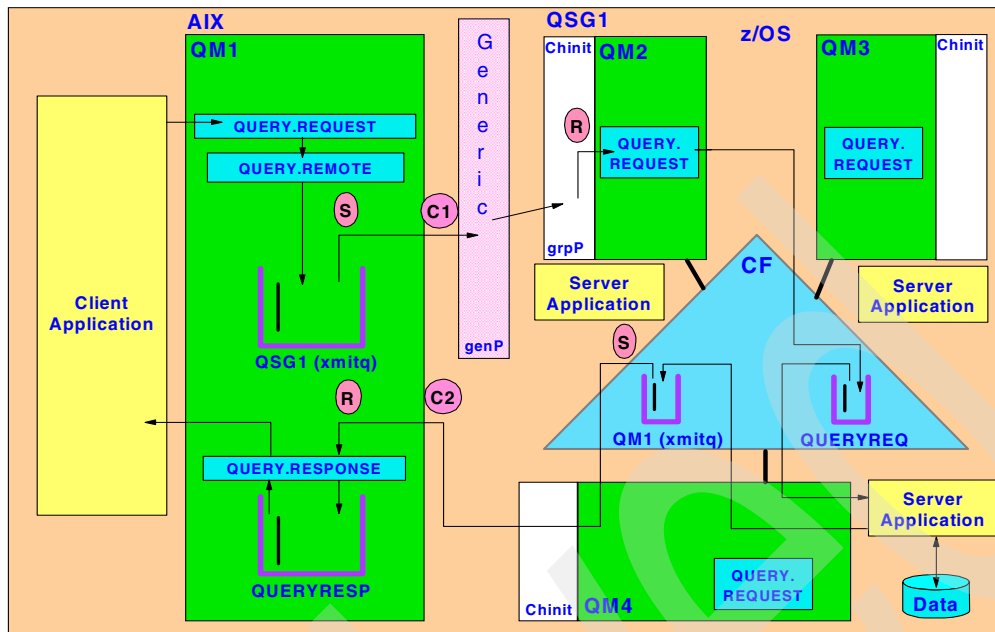


Figure 8-3 Using QSGs to exploit back-end capacity and availability

The queues defined on QM1 are:

- ▶ QUERY.REQUEST, an alias queue definition, with TARGQ(QUERY.REMOTE).
- ▶ QUERY.REMOTE, a remote queue definition, with RNAME(QUERY.REQUEST), RQMNAME(QSG1), XMITQ().
- ▶ QSG1, a private local queue definition with USAGE(XMITQ) - the transmission queue that is used to transfer queries to the QSG. This queue is served by a sender channel.
- ▶ QUERY.RESPONSE, an alias queue definition, with TARGQ(QUERYRESP).
- ▶ QUERYRESP, a private local queue definition.

Queue managers QM2, QM3 and QM4 are cloned queue managers with:

- ▶ QUERY.REQUEST, an alias queue definition, with TARGQ(QUERYREQ). This queue can be defined as a group object.
- ▶ QUERYREQ, a shared local queue definition.
- ▶ QM1, a shared transmission queue.

Channels C1 and C2:

- ▶ Sender channel C1 establishes a session with the partner receiver channel via the generic port. Hence, the inbound receiver is load-balanced started on one of the queue managers (e.g. on QM2) in the QSG.
- ▶ Sender channel C2 is a shared channel as it serves shared transmission queue QM1. This too is load-balanced started on one of the queue managers in the QSG (for example, on QM4).
- ▶ Receiver channel C1 and sender channel C2 can be defined as group objects.

The query/response flow is as follows:

1. The client application puts a query on to QUERY.REQUEST (name resolution resolves this to transmission queue QSG1).

The client application sets the `ReplyToQ` and `ReplyToQMgr` fields of the `MQMD` to `QUERY.RESPONSE` and blanks, respectively. Hence, the queue manager sets the `ReplyToQMgr` field to `QM1`.

2. The sender MCA for channel `C1` transmits the query to the partner receiver MCA which puts it on to `QUERY.REQUEST` (name resolution resolves this to `QUERYREQ`).
3. The back-end server application on one of the queue managers (for example, `QM4`) retrieves the query, processes it, generates a response, and places the response on the associated reply to queue (name resolution resolves this to transmission queue `QM1`).
4. The sender MCA for channel `C2` transmits the response to the partner receiver MCA which puts it on to `QUERY.RESPONSE` (name resolution resolves this to `QUERYRESP`).
5. The client application retrieves the response from `QUERY.RESPONSE` (name resolution resolves this to `QUERYRESP`).

Notice that in the QSG, the inbound receiver channel was started against queue manager `QM2`, while the outbound sender channel was started on `QM4`—though they may well have both started against the same queue manager.

It is also worth noting the changes that were made on the front-end queue manager `QM1`:

- ▶ The `RQMNAME` attribute in remote queue `QUERY.REMOTE` was changed to name `QSG1`.
- ▶ The name of the transmission queue on `QM1` was changed from `QM2` to `QSG1`.
- ▶ Sender channel `C1` was changed to serve transmission queue `QSG1`.

If you have many distributed queue managers connecting into the QSG, then you will clearly need to make such changes on all these queue managers. However, this kind of impact may not be acceptable, especially if you need to ask other companies to change their setup. Some possible solutions are:

1. Make no changes on the distributed queue managers and the QSG, and exploit the advantages of shared queues.

On the distributed queue managers, keep the old transmission queue name and `RQMNAME` reference (`QM2`) and let the sender channel go on serving this transmission queue. You may use a private channel to `QM2` or a shared inbound channel through the generic port to `QSG1`.

In both cases, you are still dependent on the specific queue manager (`QM2`) to be available to deliver the messages to the shared queue (`QUERYREQ`), but you can enjoy the workload balancing and availability of the server application.

This solution gives you the opportunity to make the changes on the distributed queue managers to fully exploit the sharing capabilities, one queue manager at a time. Or you may decide not to migrate on specific distributed queue managers.

2. Change the name of your QSG and the queue managers.

A technique that has been considered is to give the QSG the same name as the back-end queue manager that the QSG is intending to replace. So, the QSG could be named `QM2`, and the queue managers within the QSG could be called `QM2A`, `QM2B`, and `QM2C`, for example. This avoids having to make the changes listed above at the front-end queue manager. Additionally, if you have any batch applications that connect to `QM2`, they would now connect to the QSG (that is, they benefit from using the group attach facility).

This may be a major change in your QSG setup and you may have other dependencies to the existing naming, so consider these points carefully before you move.

8.4.3 Recommendations

- ▶ The processing of large messages in a WebSphere MQ network often necessitates special design considerations (for example, separate channels) that should be reviewed before transferring an application architecture to a parallel implementation. For example, in an image archiving system, immediate message availability may not be of prime importance.
- ▶ WebSphere MQ applications designed to gather messages for later processing (that is, batch-oriented applications) can store large amounts of message data before batch processing is started. Bearing in mind some of the limitations for shared queues, and given that this type of application is not response time-critical, delivering these messages via private channels may be more appropriate.
- ▶ For real-time applications, the amount of message data that may arrive during an application outage (for update or due to a failure, for example), and the fact that messages may be trapped, need to be considered. A solution that uses shared queues and supports parallel servers may be more suitable in such cases.
- ▶ Solutions that are running successfully from response, availability, and reliability perspectives should be lower priority for migration.
- ▶ If significant application changes are required to remove affinities, it may be appropriate to leave these unchanged.
- ▶ Clustering (or private channels) can handle larger messages than shared queues. If the maximum size of a reply message from a legacy application cannot be guaranteed to be less than 63K, it may be better to use clustering (or private channels) than to redesign the application.
- ▶ A shared queue solution assures that messages are not trapped. The availability of queue managers in a QSG allows server applications to continue serving shared queues, and removes the need to wait for a failing queue manager to be restarted.

Glossary

abend. Abnormal end of task, usually referring to a CICS task.

applet. A small Java program, downloaded from a Web server to a Web browser. Applets differ from fully fledged Java applications in that they can be restricted from performing certain operations on the local computer.

application-owning region (AOR). A CICS region in a MRO environment that “owns” the CICS applications, and invokes them on behalf of remotely attached terminal (or Web) users. See also *TOR* and *listener region*.

Application Programming Interface (API). A set of calling conventions defining how a service is invoked through a software package.

Advanced Program-to-Program communication (APPC). An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

business logic interface (BLI). An externally callable interface provided by CICS Web support. It allows a client to invoke the business logic in a CICS application. It is implemented by the module DFHWBBLI. It provides a mechanism for implementing Web-aware presentation logic in the “converter”. The converter provides Decode and Encode routines to receive and send the HTTP presentation logic.

browser. An application that displays World Wide Web documents, usually referred to as a Web browser.

Central Electronic Complex (CEC). This is the physical machine that contains main storage (memory), central processing units and connections to devices. Used primarily in the S/390 environment, and also known as Central Processing Complex.

Central Processing Unit (CPU). Known also as an engine or processor, this is the part of the computer that executes the program instructions. There may be one or many CPUs in an S/390 CEC. Each CPU in the CEC may access the main storage (memory) in that CEC. If there are multiple CPUs in a CEC, then multiprocessing (or simultaneous execution of two threads of control) is possible.

Customer Information Control System (CICS). A distributed on-line transaction processing system designed to support a network of many terminals. The CICS family of products is available for a variety of platforms ranging from a single workstation to the largest mainframe.

CICS Transaction Gateway (CTG). A set of software components that provide the ability for CICS transaction and programs to be invoked from Java programs.

CICS Web support (CWS). A set of resources supplied with CICS TS V1.3 that provide CICS with functionality similar to a real Web server, and allows both CICS programs and 3270 transactions to be invoked from the Internet. See also *CICS Web Interface*.

CICS Web Interface (CWI). A set of resources supplied with CICS/ESA V4.1 and CICS TS V1.2 that allows CICS programs to be invoked from the Internet. Superseded by *CICS Web support*.

client. As in client/server computing, the application that makes requests to the server and, often, handles the interaction necessary with the user.

client-server computing. A form of distributed processing, in which the task required to be processed is accomplished by a client portion that requests services and a server portion that fulfills those requests. The client and server remain transparent to each other in terms of location and platform. See *client* and *server*.

COMMAREA. A buffer or data area used by a CICS program to hold input and output data when LINKED (called) as a subroutine from another CICS program.

commit. An action that a transaction processing monitor takes, to make permanent the changes it has made to recoverable resources during a logical unit of work.

Common Connector Framework (CCF). The IBM common client programming model for connectors. These interfaces allow VisualAge® for Java's Enterprise Access Builder for transactions to easily build Java applets or servlets to access programs or transactions in a CICS region. The CCF also provides a common infrastructure programming model for connectors, which gives a component environment such as WebSphere a standard view of a connector.

Common Gateway Interface (CGI). A defined standard that describes how a Web server can communicate with another application (the CGI program) on the same machine. Usually a CGI program is a small program that takes data from a Web server and does something with it, like execute a database query.

Common Object Request Broker Architecture (CORBA). The Object Management Group's (OMG) standard for communication between client and server ORBs.

conversational. A communication model where two distributed applications exchange information by way of a conversation; typically one application starts (or allocates) the conversation, sends some data, and allows the other application to send some data. Both applications continue in turn until one decides to finish (or deallocate). The conversational model is a synchronous form of communication. See also *pseudo-conversational*.

cookie. A piece of information sent by a Web Server to a Web browser that the browser software is expected to save and to send back to the Server whenever the browser makes additional requests from the Server. Depending on the configuration, the browser may or may not accept the cookie. Cookies can be stored on the hard disk of the Web browser and can be reused until they expire. Cookies often contain identification information such as registration information, or user preferences.

Coupling Facility (CF). A special zSeries logical partition that provides high-speed caching, list processing, and locking functions between systems in a S/390 Parallel Sysplex. The Coupling Facility can run in a zSeries logical partition or in a dedicated Coupling Facility processor.

Cryptographic Coprocessor Feature. An additional hardware unit for a S/390 system that contains dual cryptographic module chips protected by tamper-detection circuitry. It can be used to offload cryptographic operations from the main processors.

distributed program link (DPL). This enables an application program executing in one CICS system to link to (or call) a program in a different CICS system. The linked-to program executes and returns a result to the linking program in a buffer known as a COMMAREA. This function is similar to a remote procedure call (RPCs) as provided by client-server architectures.

Distributed Transaction Processing (DTP). This enables a transaction running in one CICS system to communicate synchronously with transactions running in other system using the LU6.2 protocol. The transactions are designed and coded specifically to communicate with each other.

Data Language/I (DL/I) The IMS data manipulation language, a common high-level interface between a user application and IMS. DL/I calls are invoked from application programs written in languages such as PL/I, COBOL, VS Pascal, C, and Ada. It can also be invoked from assembler language application programs by subroutine calls. IMS lets the user define data structures, relate structures to the application, load structures, and reorganize structures.

Domain Name System (DNS). System for resolving TCP/IP hostname to IP addresses, and vice versa.

External Call Interface (ECI). An application programming interface (API) provided by the CICS client that enables an application on a workstation to call a CICS program as a subroutine. The client application communicates with the server CICS program using a data area called a COMMAREA.

External CICS Interface (EXCI). An application programming interface (API) provided by CICS on z/OS that enables a non-CICS application running on z/OS to call a CICS program as a subroutine. It is similar in function to the ECI but is available only on z/OS.

External Presentation Interface (EPI). An application programming interface (API) provided by the CICS Client that allows a 3270-based CICS transaction to be driven programmatically.

External Security Interface (ESI). An application programming interface (API) provided by the CICS Universal Client that utilizes the APPC PEM support in a CICS region to verify and update userids and passwords.

External Security Manager (ESM). A external security product, such as RACF, used by CICS for authentication and authorization of users.

extranet. A combination of TCP/IP networks of different companies connected with a secure connection, perhaps using Virtual Private Network technology (VPN).

function shipping. A CICS intersystem communication protocol that enables an application program running in one CICS system to access resources owned by another CICS system. In the resource-owning system, a mirror transaction is initiated to perform the necessary operation; for example, to access CICS files or temporary storage, and to reply to the requester.

gateway. Software that transfers data between normally incompatible applications, or between networks.

Graphical user interface (GUI). A style of user interface that replaces the character-based screen with an all-points-addressable, high-resolution graphics screen. Windows® display multiple applications at the same time and allow user input by means of a keyboard or a pointing device such as mouse, pen, or trackball.

Go Webserver Application Programming Interface (GWAPI). API provided by the Domino® Go Webserver for OS/390.

Hierarchical File System (HFS). z/OS UNIX® System Services filing system.

host. (1) In a computer network, a computer providing services such as computation, database access, and terminal control functions. (2) In a multiple computer installation, the primary or controlling computer. Often used to refer to by an IBM S/390 processor.

Host Access Class Library (HACL). This is provided by Host On-Demand software from IBM as a set of Java classes and methods that allow the development of platform-independent applications that can access host information (such as CICS 3270 terminals) at the data stream level.

Host On-Demand (HOD). Terminal emulation software from IBM, downloadable “on-demand” from a Web server.

hypertext. Text that activates connection to other documents when selected.

Hypertext Markup Language (HTML). Standard scripting language used to create hypertext documents. Widely used for documents on the World Wide Web.

Hypertext Transmission Protocol (HTTP). Standard Internet client/server communications protocol.

Interface Definition Language (IDL). A definition language used to define an interface between client and server objects, including the operations that can be performed by the server. Used by, among others, the CORBA standard and the DCE protocol.

intranet. An internal local area network, private to a corporation or other organization.

internet. A collection of interconnected networks, compare with *Internet*.

internet. The global TCP/IP network that forms the backbone of the World Wide Web.

Internet Connection Application Programming Interface (ICAPI). OS/390 Web server application programming interface, now replaced by GWAPI.

Internet Inter-ORB Protocol (IIOP). An industry standard that defines formats and protocols to provide client/server semantics for distributed object-oriented applications in a TCP/IP network. It is part of the CORBA architecture.

intersystem communication (ISC). Intercommunication between separate CICS regions (systems) by means of SNA networking facilities or the application-to-application facilities of VTAM. ISC can also be used to connect other APPC-based system such as the CICS Universal Client or native APPC applications. Compare with *multi-region operation* (MRO).

Java Virtual Environment (JVM). The virtual environment within which a Java application runs. The JVM provides for such features as platform neutrality and multi-threading.

Java Native Interface (JNI). Interface provided by the Java language to enable native (non-Java) code to be invoked.

listener region. A CICS region that listens for incoming requests and routes them onto to an attached application-owning regions. A CICS system can listen for request from HTTP or CORBA clients using the CICS TCP/IP listener, from LU6.2 clients using CICS ISC support, or from other z/OS address spaces (such as the z/OS Web server) using the EXCI protocol. See also *application-owning region* and *terminal-owning region*.

local area network (LAN). A localized computer network usually based on ethernet or token-ring technology. See also *intranet* and *WAN*.

logical partition (LPAR). A logical subset of the S/390 CEC hardware. CEC resources, CPUs and main memory can be shared between logical partitions. Each logical partition is capable of running an instance, or image, of the z/OS operating system.

LU type 6.2 (LU 6.2). A type of SNA logical unit that offers multiple (parallel), half duplex sessions between independent (peer-to-peer) devices. Used extensively by CICS intersystem communication. APPC is the application programming interface provided by an LU6.2 logical unit.

Multi Region Operation (MRO). Intercommunication between CICS regions in the same processor (or Sysplex) without the use of SNA network facilities. This allows several CICS regions to communicate with each other, and to share resources such as files, terminals, temporary storage, and so on. Contrast with *intersystem communication*.

Network Address Translation (NAT). A protocol used to provide a mapping between internal IP addresses and external or public (globally unique) IP addresses. Often used when connecting an intranet to the Internet. Since the internal addresses are not advertised outside the intranet, NAT can be used when they are private (globally ambiguous) addresses, or when they are public (globally unique addresses that a company wishes to keep secret).

Object Management Group (OMG). The consortium of software organizations that has defined the CORBA architecture.

Object Request Broker (ORB). A CORBA system component that acts as an intermediary between the client and server applications. Both client and server platforms require an ORB; each is tailored for a specific environment, but support common CORBA protocols and interfaces.

On-line Transaction Processing (OLTP). A style of computing that supports interactive applications in which requests submitted by terminal users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. An on-line transaction processing system supervises the sharing of resources to allow efficient processing of multiple transactions at the same time.

Operating System 390 (OS/390). IBM operating system for the S/390 platform. Previously known as MVS.

Parallel Sysplex. A sysplex that uses one or more Coupling Facilities.

pipe. An EXCI pipe is a one-way communication path between a sending client process and a receiving CICS region. Each pipe maps onto one CICS MRO session. See also *EXCI* and *MRO*.

proxy. A software gateway between connecting networks that allows communication between the two networks, by acting as both a client and a server. A popular usage of a proxy is a HTTP proxy server, which allow Web browsers in a private intranet to connect to Web servers on the Internet, but restricts all other network communications between the two networks.

pseudo-conversational. A type of CICS application design that appears to the user as a continuous *conversational* flow, but actually consists of multiple interconnected CICS tasks. Such a design is inherently more scalable than a *conversational* design, since a CICS task only executes when a user requests data, and not while the transaction is waiting for user input.

Request for Comments (RFC). A set of standards for TCP/IP and Internet protocols, managed by the Internet Engineering Task Force (IETF).

Secure Sockets Layer (SSL). A protocol designed by Netscape to enable encrypted, authenticated communications across the TCP/IP networks. It is most widely used between Web browsers and Web servers. A URL that begins with the “https” protocol indicate that an SSL connection will be used for the HTTP datastream.

servlet. A Java program that executes within a JVM on a Web server and “serves” HTML to the Web browser. Similar in concept to a CGI program.

Socket Secure (SOCKS). A proxy gateway that allows compliant client code (client code made socket secure) to establish a TCP/IP session with a remote host via means of the SOCKS gateway.

Standard Generalized Markup Language (SGML). The standard that defines several markup languages, HTML included.

System 390 (S/390). IBM mainframe hardware platform.

Systems Network Architecture (SNA). IBM networking protocol, that offers connections between nodes termed Physical Units (PUs), and offers programming services from Logical Units (LUs). See also *LU6.2*

sysplex. A set of z/OS or OS/390 systems (also called z/OS images or logical partitions) that communicate using multi-system hardware components and software. Systems in a Sysplex share disk storage.

TCP62. A communication protocol available when using the CICS Universal Client. It utilizes the AnyNet® function of the IBM Communication Server to provide LU6.2 communication over TCP/IP (LU6.2/IP). It also provides for dynamic configuration of the client SNA node.

TCP/IP listener. In CICS, the function provided by the CICS Sockets listener task (CSOL) that handles incoming HTTP and IIOF requests.

terminal-owning region (TOR). A CICS region that “owns” the network of attached terminals and routes the request to an attach application-owning region (AOR).

transaction. (1) In CICS, a unit of processing consisting of one or more application programs initiated by a single request. A transaction can require the initiation of one or more tasks for its execution. (2) A unit of work that transforms one or more resources from one consistent state to another consistent state.

transaction processing. A style of computing that supports interactive applications in which requests submitted by users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. A transaction processing system supervises the sharing of resources for processing multiple transactions at the same time.

transaction routing. A CICS intercommunication protocol, that enables a terminal connected to one CICS region to run a transaction in another CICS region. It is common for CICS/ESA, CICS/VSE®, to have a terminal-owning region (TOR) that “owns” a network of terminals.

Transmission Control Protocol/Internet Protocol (TCP/IP). The networking protocol that forms the basis of the Internet.

URL (Uniform Resource Locator). Format used to describe the resources located on the World Wide Web. A URL is of the form protocol://server/pathname/document. It consists of the document's name preceded by pathname where the document can be found, the Internet domain name of the server that hosts the document, and the protocol (such as http or ftp) by which the Web browser can retrieve the document from the server.

Virtual Telecommunications Access Method (VTAM). Networking software for the z/OS operating system that implements the SNA network protocol. Now part of the IBM Communications Server for z/OS.

Web-aware. A term used to denote a CICS application that contains logic to send and receive HTTP datastreams as opposed to 3270 datastreams.

World Wide Web (WWW). The collection of resources on the Internet including HTTP servers, newsgroups, and ftp sites.

Archived

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 186. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129
- ▶ *DB2 for z/OS Application Programming Topics*, SG24-6300
- ▶ *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know... and More*, SG24-6079
- ▶ *DFSMStvs Application Migration Guide*, SG24-6972
- ▶ *DFSMStvs Overview and Planning Guide*, SG24-6971
- ▶ *Distributed Functions of DB2 for z/OS and OS/390*, SG24-6952
- ▶ *IBM DB2 Performance Expert for z/OS Version 2*, SG24-6867
- ▶ *IMS in the Parallel Sysplex Vol. 1: Reviewing the IMSplex Technology*, SG24-6908
- ▶ *Locking in DB2 for MVS/ESA Environment*, SG24-4725
- ▶ *OS/390 Parallel Sysplex Configuration, Volume 1: Overview*, SG24-5637
- ▶ *OS/390 Parallel Sysplex Configuration, Volume 2: Cookbook*, SG24-5638
- ▶ *OS/390 Parallel Sysplex Configuration, Volume 3: Connectivity*, SG24-5639
- ▶ *Parallel Sysplex - Managing Software for Availability*, SG24-5451
- ▶ *TCP/IP in a Sysplex*, SG24-5235

Other publications

These publications are also relevant as further information sources:

- ▶ *DB2 UDB for z/OS Version 8 Data Sharing: Planning and Administration*, SC18-7417
- ▶ *DB2 UDB for z/OS Version 8 Application Programming and SQL Guide*, SC18-7415
- ▶ *DB2 UDB for z/OS Version 8 SQL Reference*, SC18-7426
- ▶ *DB2 UDB for z/OS Version 8 Administration Guide*, SC18-7413
- ▶ *DB2 UDB for z/OS Version 8 Command Reference*, SC18-7416
- ▶ *CICS Resource Definition Guide*, SC34-5990
- ▶ *CICS Application Programming Guide*, SC34-5993
- ▶ *CICS Application Programming Reference*, SC34-6026
- ▶ *DB2 UDB for z/OS Version 8 Installation Guide*, GC18-7418
- ▶ *IMS Version 8: Administration Guide: DB*, SC27-1283
- ▶ *IMS Version 8: Administration Guide: System*, SC27-1284
- ▶ *IMS V8 Application Programming: DB*, SC27-1286

- ▶ *IMS Version 8: Utilities Reference: Database and Transaction Manager*, SC27-1308
- ▶ *WebSphere MQ for z/OS Concepts and Planning Guide*, GC34-6051
- ▶ *WebSphere MQ for z/OS System Administration Guide*, GC34-6053
- ▶ *IBM Systems Journal Volume 36, Number 2, 1997*
- ▶ *In Search of Clusters: The Coming Battle in Lowly Parallel Computing*, Pfister, G.F., Prentice Hall, Upper Saddle River, NJ, 1995.
- ▶ "Restart Services for Highly Available Systems," *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, N. S. Bowen, C. A. Polyzois and R. D. Regan, October 1995.

Online resources

These Web sites are also relevant as further information sources:

- ▶ Parallel Sysplex for OS/390 and z/OS
<http://www.ibm.com/servers/eserver/zseries/psa>
- ▶ z/OS Integration Test
<http://www.ibm.com/servers/eserver/zseries/zos/integtst/>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications, and Additional materials, as well as order hardcopy Redbooks or CD-ROMs at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

- affinities 21
 - types of 21
- affinity-safe programming techniques 92
- affinity-suspect techniques 92
- affinity-unsafe programming techniques 92
- application considerations for DB2 system administrators
 - backup and recovery 62
 - BIND options 57
 - DB2 installation options 59
 - DDL options 58
 - deadlock and time-out detection 61
- application considerations for developers
 - additional DB2 considerations 55
 - lock avoidance 41
 - locking 40
 - programming techniques for lock avoidance 43
- application design
 - applications in a sysplex 18
 - other considerations 29
- application restart 70
- application-owning region (AOR) 179

B

- batch application considerations
 - sharing files in the batch job structure 70
 - unit of recovery 69
- batch considerations 51
- batch performance 70
- batch window 25
- BSDS 39
- business logic interface (BLI) 179
 - converter 179
 - decode 179
 - encode 179

C

- channel initiators 137
- CICS application considerations 81
 - CICS TS application checklist 92
 - how to exploit sysplex in CICS TS applications 82
 - introduction to CICS TS 82
- CICS Coupling Facility 165
- CICS Transaction Affinities Utility 87
- CICS TS application checklist 92
- CICS Web Interface (CWI) 179
- CICS Web support 179
- Commarea 89
- Common Connector Framework (CCF) 179
- Common Gateway Interface (CGI) 179
- Common Work Area (CWA) 89
- continuous availability 19
- cookie, HTTP 180

- CORBA 180
- Coupling Facility (CF) 179–180
- CREATE SEQUENCE 59
- Cryptographic Coprocessor Feature, S/390 180
- CSOL, sockets listener transaction 182
- customer account key 62

D

- data sharing
 - programming guidelines 54
- DB2 application considerations
 - how data sharing works 38
 - introduction to DB2 data sharing 34
- DFHWBBLI, *See business logic interface*
- DFSMStvs 67
 - batch application considerations 69
 - introduction 68
 - programming changes 77
 - transactional recovery 71
- distributed access to a data sharing group 53
- distributed program link (DPL) 180
- DLIBATCH 113

E

- ECI
 - definition of 180
- EDMPOOL 61
- Enterprise Access Builder (EAB) 179
- EPI
 - definition of 180
- ESI
 - definition of 180
- Example application
 - Extraordinary processes 65
 - Parallelizing sequential processes 64
- example application
 - batch processing 63
 - parallelizing sequential processes 64
- EXCI
 - definition of 180
- extranet 180

F

- Fast Database Recovery (FDBR) 101
- function shipping 180

G

- GETMAIN SHARED 89
- Go Webserver API (GWAPI) 180
- group buffer pools 38
- GRPCACHE 59

H

- Hierarchical File System (HFS) 180
- high availability large database 102
- Host Access Class Library (HACL) 181
- Host On-Demand (HOD) 181
- how to exploit sysplex in CICS TS applications
 - CICS transaction affinities utility 87
 - programing facilities/techniques and their implications on affinities 89
 - what happens if you have to create an affinity 92
- HTML
 - definition of 181
- HTTP
 - cookies 180
 - definition of 181

I

- implementation and migration 163
 - CICS 164
 - create the CICS sysplex environment 164
 - exploit the sysplex environment with the applications 166
 - DB2 data sharing implementation 166
 - planning for migration 164
 - WebSphere MQ 170
 - migrating your existing applications 170
 - Migrating your existing applications to use shared queues 170
 - migration scenario 172
 - recommendations 178
- IMS bridge 150
- IMS Database Manager
 - Locking recommendations 107
- IMS database manager 103
 - batch considerations 113
 - converting MSDBs 117
 - data sharing integrity overview 106
 - single points of failure 112
 - supported IMS databases 103
- IMS transaction manager 119
 - affinities 119
 - connectivity 124
 - general IMS TM application performance considerations 123
 - IMS configuration considerations 128
 - transaction simplification 122
 - transaction workload balancing 121
- Interface Definition Language (IDL) 181
- Internet 181
- internet 181
- Internet Inter Orb Protocol (IIOP) 181
- intersystem communication (ISC) 181
- intra-group queuing 141
- intranet 181

J

- Java
 - applets 179
 - Java Native Interface (JNI) 181

Java Virtual Machine (JVM) 181

L

- listener region 181
- Local Area Network (LAN) 181
- lock structure 38
- locking 24
- LOCKMAX 58
- LOCKSIZE 58
- Logical Partition (LPAR) 181
- logs and BSDS 39
- LRDRTHLD 60

M

- managing commit frequency in batch 52
- MAXROWS 58
- MEMBER CLUSTER 59
- migration scenario 172
- multiple cloned servers 148
- multi-region operation (MRO) 181

N

- Network Address Translation (NAT) 181
- NUMLKTS 60
- NUMLKUS 60

O

- Object Management Group (OMG) 181
- online transaction processing (OLTP) 12

P

- parallel processing 9
- Parallel Sysplex
 - data sharing in a sysplex 5
 - introduction 1
 - managing work 14
 - parallel processing 8
 - summary 5
 - sysplex hardware 2
 - sysplex software 2
 - what is a sysplex 2
 - why use a sysplex 3
- performance 26
- programming changes 77
 - application program migration steps 79
 - how to commit 78
 - how to perform a backout 79
- pseudo-close 61

Q

- queue partitions 151

R

- Redbooks Web site 186
 - contact us xii
- REQID 90

Request for Comments (RFC) 182

S

- sample application
 - extraordinary processes 65
- SCA 38
- serialized applications 151
- service account key 63
- shared inbound channels 138
- shared outbound channels 139
- shared queue triggering 154
- shared queues and clustering 152
- single points of failure 25
- SOCKS 182
- sysplex parallelism 60

T

- task synchronization 90
- TCP62 182
- TCTUA 89
- temporary storage queues 91
- TRACKMOD 59
- transaction managers 12
- transactional recovery 71
 - batch application restart 71
 - conventional batch job structure 72
 - using DFSMStvs - shared input 75
 - using DFSMStvs - unique input 73

U

- URCHKTH 60
- URLGWTH 60

V

- versioning 27
- Virtual Private Network (VPN) 180

W

- Web-aware, CICS application 183
- WebSphere MQ application considerations 131
 - a standalone queue manager 133
 - an introduction 132
 - benefits of using queue sharing groups 143
 - channels in a queue-sharing group 137
 - coupling facility list structures 135
 - final considerations 161
 - how applications can exploit shared queues 147
 - initial considerations related to the use of QSGs 144
 - limitations and restrictions 158
 - queue managers in a queue-sharing group 134
 - sharing and clustering 159

Archived



Parallel Sysplex Application Considerations



Redbooks

**Introducing
architecture mindset
to leverage z/OS
sysplex applications**

**Designing
applications to
benefit from Parallel
Sysplex**

**DB2, DFSMStvs, CICS,
IMS, WebSphere MQ
applications**

This IBM Redbook introduces a top-down architectural mindset that extends considerations of IBM z/OS Parallel Sysplex to the application level, and provides a broad understanding of the application development and migration considerations for DB2, DFSMStvs, CICS, IMS, Transactional VSAM, and WebSphere MQ applications.

Parallel computing and data sharing technologies are playing a major role in e-business computing. Despite their compelling capabilities, however, these technologies are virtually unknown within the application architect and developer communities and are deployed more like operational enhancement technologies, with limited application involvement. Parallel Sysplex deployment models often expressly attempt to segregate the application environment from the underlying environment.

But today's software engineering principles recommend that you view applications to be designed as the drivers for availability and data sharing requirements for any technology. This publication explains how to achieve this objective by designing applications to exploit the capabilities of Parallel Sysplex.

**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**

SG24-6523-00

ISBN 0738424064