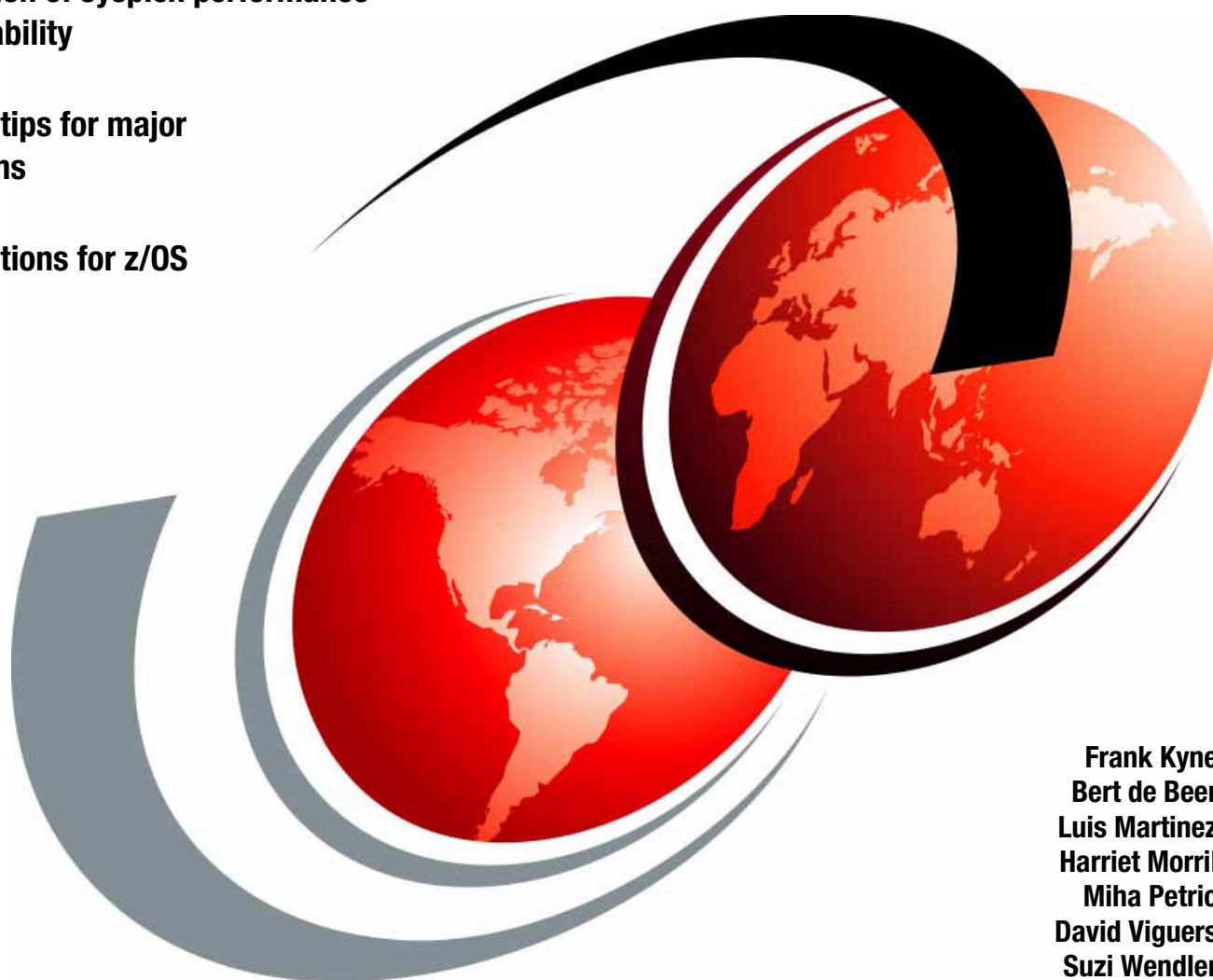


System z Parallel Sysplex Best Practices

Optimization of sysplex performance
and availability

Hints and tips for major
subsystems

Considerations for z/OS



Frank Kyne
Bert de Beer
Luis Martinez
Harriet Morril
Miha Petric
David Viguers
Suzi Wendler

Redbooks



International Technical Support Organization

System z Parallel Sysplex Best Practices

January 2011

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

First Edition (January 2011)

This edition applies to Version 1, Release 11, Modification of z/OS (product number 5674-A01).

© Copyright International Business Machines Corporation 2011. 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 book	ix
Now you can become a published author, too!	xi
Comments welcome	xi
Stay connected to IBM Redbooks	xi
 Chapter 1. Introduction	 1
1.1 Why Parallel Sysplex best practices	2
1.2 General hints and tips	2
1.2.1 Threes	2
1.2.2 Twos	3
1.2.3 Ones	3
1.2.4 Murphy's Law	4
1.2.5 When a backup will not help you	5
1.2.6 IPL frequency	5
1.2.7 Rolling IPLs	6
1.2.8 Sysplex separation	6
1.3 Layout of this document	7
 Chapter 2. z/OS sysplex best practices	 9
2.1 Introduction	10
2.2 XCF	10
2.2.1 Couple data sets	11
2.2.2 XCF signalling paths	25
2.2.3 Transport classes	32
2.2.4 Failure handling	34
2.3 Coupling Facilities	39
2.3.1 CF-level recommendations	39
2.3.2 Structure-level recommendations	54
2.3.3 How many sysplexes to have	67
2.4 Testing with copies of a sysplex	68
2.5 System Logger	70
2.5.1 Logger protection of log data	70
2.5.2 Types of Logger exploiters	71
2.5.3 General Logger guidelines	71
2.5.4 CF log stream guidelines	74
2.5.5 DASDONLY log stream guidelines	76
2.6 Avoiding single points of failure	76
2.7 z/OS Health Checker	78
 Chapter 3. CICS sysplex best practices	 81
3.1 Introduction	82
3.2 Affinities	83
3.2.1 Use of temporary storage pools	84
3.2.2 Named counters	86
3.2.3 CICS data tables	87

3.2.4	VSAM RLS	89
3.2.5	CICS ENQs	89
3.3	Effective CICS workload balancing	90
3.3.1	CICS topology and session balancing	90
3.3.2	Dynamic transaction routing	90
3.4	CICS subsystem connectivity	93
3.4.1	CICS-MQ connection	93
3.4.2	CICS-DB2 connections	93
3.4.3	CICS and Automatic Restart Manager	94
3.5	General suggestions	95
3.5.1	CICS log-stream-related suggestions	95
3.5.2	CICS use of XCF groups	97
3.5.3	CICS naming conventions and cloning	98
Chapter 4.	DB2 sysplex best practices	99
4.1	Introduction	100
4.2	Coupling Facility-related recommendations	100
4.2.1	Avoid single points of failure in CF configuration	100
4.2.2	Failure-isolated lock and SCA structures	101
4.2.3	Always use duplexing for DB2 GBP structures	102
4.2.4	Enable auto alter for all DB2 CF structures	104
4.2.5	Ensure that storage for the SCA structure is generous	107
4.2.6	Aim for zero cross-invalidations due to directory reclaims	107
4.2.7	Recommendations on GBP checkpoint frequency	108
4.2.8	Recommendations on PCLOSET and PCLOSEN	109
4.2.9	Synchronized SYNCVAL value	110
4.2.10	Best practices for GBPCACHE attribute	110
4.2.11	Tablespace to buffer pool relationship	110
4.3	Restart considerations	112
4.3.1	Use ARM to restart DB2 as quickly as possible after a failure	112
4.3.2	Planned restart	114
4.4	General recommendations	114
4.4.1	Minimize lock contention	114
4.4.2	TRACKMOD recommendation	118
4.4.3	CONDBAT recommendation	118
4.4.4	RETLWAIT system parameter	118
4.4.5	Tune system checkpoint frequency	118
4.4.6	Relative WLM importance of DB2 address spaces	119
4.4.7	Ensure that the system has sufficient free main storage	119
4.4.8	Workload balancing considerations	119
4.4.9	Use rolling IPLs methodology to upgrade your DB2s one at a time	122
4.4.10	Avoid single points of failure for critical DB2 data sets	122
4.4.11	Archive to DASD instead of tape	122
4.4.12	Changing DSNZPARM values without restarting DB2	123
4.4.13	DROP unused indexes	123
Chapter 5.	IMS sysplex best practices	125
5.1	Introduction	126
5.2	IMS data sharing	126
5.2.1	DBRC considerations	126
5.2.2	Database considerations	127
5.2.3	Database cache structures	128
5.2.4	IRLM	129

5.2.5 Other database-related considerations	131
5.2.6 Applications	133
5.3 IMS shared queues	133
5.3.1 CQS system checkpoints	134
5.3.2 CQS structure checkpoints	134
5.3.3 CQS shared queue structures	135
5.3.4 CQS and the MVS System Logger	136
5.3.5 False scheduling	137
5.3.6 Resource Recovery Services (RRS) considerations	137
5.4 IMS connectivity	138
5.5 Other miscellaneous considerations	138
Chapter 6. MQ sysplex best practices	141
6.1 Introduction	142
6.2 Location and sizing	143
6.2.1 Coupling Facility structures	143
6.2.2 Structure backup and recovery	146
6.2.3 Peer recovery	147
6.3 General recommendations	149
6.3.1 Which non-application queues should be shared	150
6.3.2 Shared queues and clustering	150
6.3.3 Inbound shared channels and clustering channels	150
6.3.4 What to do with non-persistent messages on shared queues after a Queue Manager restart	151
6.3.5 Use backout queue	151
6.3.6 Shared queue definitions	152
6.3.7 Intra-group queuing	152
6.3.8 Double hop avoidance	153
6.3.9 Increase MQ network availability	153
Chapter 7. WebSphere Application Server sysplex best practices	157
7.1 WebSphere Application Server configuration	158
7.1.1 Configure a mobile deployment manager	159
7.1.2 Cell and node isolation	160
7.2 Application server recommendations	160
7.2.1 Determine the optimal number of servant regions	161
7.2.2 Monitor JVM heap use	161
7.2.3 Exploit the rollout update option	162
7.2.4 Configure DB2 data sharing when using WebSphere solutions	162
7.2.5 Configure WMQ as the JMS provider and implement queue sharing	163
7.2.6 Configure EIS regions in same LPARs as WebSphere Application Server for z/OS	163
7.3 System z hardware-related recommendations	164
7.3.1 Use System z specialty processors	164
7.3.2 Exploit the Coupling Facility to improve logging performance	165
7.4 z/OS component recommendations	165
7.4.1 Logger offload data set sizes	165
7.4.2 Disable the RRS archive log	166
7.4.3 Exploit 64-bit operating mode	166
7.4.4 Use XCF communication for the HA manager	166
7.4.5 Use zFS rather than HFS	166
7.4.6 Determine which file systems to make sharable	167
7.4.7 Implement Automatic Restart Manager (ARM)	167

7.4.8 Tailor Workload Manager policies.	168
7.4.9 Use Fast Response Cache Accelerator	168
7.4.10 Use WebSphere Configuration Tool for z/OS	169
Appendix A. Important sysplex messages	171
Automation and critical messages	172
XCF messages.	172
XES messages.	174
System Logger messages	174
Related publications	177
IBM Redbooks publications	177
Other publications	177
Online resources	177
How to get Redbooks publications.	177
Help from IBM	178
Index	179

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

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

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

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


COPYRIGHT LICENSE:

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

Trademarks

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

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

CICSplex®	MQSeries®	System Storage®
CICS®	MVS™	System z10®
DB2 Connect™	OMEGAMON®	System z®
DB2®	OS/390®	Tivoli®
DRDA®	Parallel Sysplex®	VTAM®
DS8000®	PR/SM™	WebSphere®
ESCON®	RAA®	z/OS®
FICON®	RACF®	z/VM®
GDPS®	Redbooks®	z10™
HiperSockets™	Redbooks (logo)  ®	z9®
IBM®	Resource Measurement Facility™	
IMS™	RMF™	

The following terms are trademarks of other companies:

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

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

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

Preface

This IBM® Redbooks® publication pulls together diverse information regarding the best way to design, implement, and manage a Parallel Sysplex® to deliver the levels of performance and availability required by your organization.

This book should be of interest to system programmers, availability managers, and database administrators who are interested in verifying that your systems conform to IBM best practices for a Parallel Sysplex environment. In addition to z/OS® and the sysplex hardware configuration, this book also covers the major IBM subsystems:

- ▶ CICS®
- ▶ DB2®
- ▶ IMS™
- ▶ MQ
- ▶ WebSphere® Application Server

To get the best value from this book, readers should have hands-on experience with Parallel Sysplex and have working knowledge of how your systems are set up and why they were set up in that manner.

The team who wrote this book

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

Frank Kyne is an Executive IT Specialist and Project Leader at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of Parallel Sysplex and high availability. Before joining the ITSO 12 years ago, Frank worked in IBM Ireland as an MVS™ Systems Programmer.

Bert de Beer is a Senior IT Specialist who has worked for 35 years with IBM The Netherlands in many positions. He started his career in Customer Engineering providing hardware support, followed by a period providing software support for MVS. Bert later became an MVS Systems Programmer and worked in that position at customer locations. During that period Bert supported the introduction of the first Parallel Sysplex at a major bank in The Netherlands. Bert also worked in the IBM Education Center, teaching major z/OS classes, like Parallel Sysplex, RMF™, and WLM. Currently, Bert works with customers, specializing in the implementation of Parallel Sysplex and system performance.

Luis Martinez is a Team Leader in Central Systems at Iberdrola, one of the world's largest energy companies, based in Spain. He has more than 20 years of mainframe experience. His areas of expertise include z/OS, CICS, and Parallel Sysplex.

Harriet Morril is a Certified System z® Technical Support Specialist in Poughkeepsie, New York. She has 26 years of experience working in technical support for z/OS and related software, most recently at the IBM High Availability Center of Competency. She holds a master's degree in Economics from Stanford University and a Certificate of Advanced Study in Computer Science from Wesleyan University in Middletown, Connecticut. Her areas of expertise include System z high availability and best practices for Parallel Sysplex deployment and management. She recently served as Guest Editor, and an author for the *IBM Systems Journal*, Vol. 47, no. 4 which was devoted to continuous availability of systems

infrastructures. She also has published textbooks on programming with Little, Brown, and Company.

Miha Petric is a System Programmer with IBM Slovenia. His areas of expertise include MVS and its subsystems. He has worked in this field since 1978 and provides consultancy services. He also teaches classes on a variety of technical topics.

David Viguers is a Senior Technical Staff Member with the IBM IMS development team at the Silicon Valley Lab, working on performance, test, development, and customer support. He has 38 years of experience with IMS. Dave presents at technical conferences and user group meetings regularly and is also responsible for developing and delivering education on IMS Parallel Sysplex.

Suzi Wendler is a Consulting IT Specialist in the IMS support group of the IBM Advanced Technical Skills organization. Her major responsibilities have included areas such as customer support, critical situation analyses, IMS On Demand, SOA, and Connectivity projects. She has over 26 years of experience with IMS and spends much of her time participating in on-site consulting projects including implementations of connectivity designs and proofs of concept. Additionally, she has presented at several user groups, including Share. She has been involved in IBM-sponsored activities such as roadshows and technical conferences, has provided both technical and teaching support for the IMS education initiative, and has been a participating author in several IBM Redbooks.

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

Richard Conway
Robert Haimowitz
International Technical Support Organization, Poughkeepsie Center

Michael Ferguson
Feroni Suhood
IBM Australia

Bart Steegmans
IBM Belgium

Alan Murphy
IBM Ireland

Mario Bezzi
IBM Italy

Andy Clifton
John Campbell
Arndt Eade
Pete Siddall
IBM United Kingdom

Don Bagwell
Mark Brooks
Carolyn Elkins
Nicole Fagen
David Follis
Thomas Hackett
Gary King
Rich Lewis
Judy Ruby-Brown

Neil Shah
Gina Skelton
David Surman
Doug Zobre
IBM USA

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

<http://www.facebook.com/IBM-Redbooks>

- Follow us on twitter:

<http://twitter.com/ibmredbooks>

- Look for us on LinkedIn:

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

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



Introduction

This chapter introduces this book. It explains the rationale for creating the book, and describes the layout.

1.1 Why Parallel Sysplex best practices

One of the greatest strengths of Parallel Sysplex is its flexibility. Parallel Sysplex can be exploited by the largest corporations with huge sysplexes, and by small customers with just a single processor. However, this flexibility can also be a challenge. With so many options, how do you know what is the best way to configure a sysplex to achieve *your* objectives? And with so many demands for your time, how can you possibly keep up with all the latest recommendations for how to set up your sysplex?

This book makes it easier for you to obtain the latest guidance for your Parallel Sysplex in one succinct and easy-to-use place. Configuring your sysplex to adhere to these guidelines does not guarantee that you will never encounter an outage again, but it should result in a more resilient configuration that delivers improved levels of service.

Note: The guidelines and best practices described in this book are intended for production sysplexes and sysplexes that require production levels of availability and performance.

There might be valid reasons for test or development sysplexes not adhering to the guidance provided in this book. For example, it might not be possible to cost justify the investment required to meet certain guidelines provided in this book.

1.2 General hints and tips

Certain general considerations related to best practices and high availability apply across z/OS and all the subsystems, and we cover those here.

1.2.1 Threes

Before we go into detail about specific products or functions, we briefly discuss the avoidance of single points of failure. This discussion applies to both hardware and software. To an extent it applies to a monoplex environment, but it is more applicable to a data-sharing Parallel Sysplex.

When investigating the avoidance of single points of failure, people tend to think about having two of the resources in question. You might think about having two processors instead of one, two switches instead of one, two z/OS systems instead of one, and so on. However, if you have two z/OS systems and one is unavailable for some reason, you now have a new single point of failure (the remaining z/OS system).

Depending on how important high availability is to your business, you might decide that what you really need is three instances instead of two, so if one z/OS is down as part of a move to a newer release, and a second z/OS fails, you still have a running z/OS to provide service to your users. While you might be extremely unlucky and have a problem on the last z/OS, there is no doubt that having three or more instances of a critical resource should, on average, deliver better availability than having just two.

Sometimes, for capacity reasons, you might decide to have more than three instances (CICS AORs, for example). Equally, you might determine that the cost of three instances of certain resources is more than would be the impact were you to encounter a complete outage. However, we encourage you to consider the concept of addressing single points of failure by having three instances rather than two.

1.2.2 Twos

Traditionally, in the days before data sharing and very powerful processors, people avoided having multiple members from the same sysplex on the same processor. The reasoning behind this was that if a processor had to be shut down, you did not want the outage to affect more than one production system.

However, now that processors are larger and more reliable, running multiple systems from the same sysplex on the same processor is becoming more common. In fact, we encourage such configurations. The reason for this is that if a system is down and it is the only member of that sysplex on that processor, then you have lost *all* the capacity of that processor from the sysplex. Conversely, if you had two or more members from that sysplex on that processor and they are defined with shared engines, then the surviving members should be able to use the capacity freed up by the unavailable system, thereby keeping the capacity of that processor available to the sysplex.

We also encourage you to extend this concept to software components within a system. So, for example, instead of having just *one* CICS TOR in a system, have two. This way, if you need to stop a TOR for any reason, you still have a CICS TOR on that system that can accept new logons, so you do not end up with all your CICS work vacating that system. The same concept can be applied to other software components. For example, you can have two members of the same DB2 data sharing group in the same system, so that if you need to stop one of them for any reason, you can still (transparently) run DB2 work on that system using the other DB2.

1.2.3 Ones

Sysplexes with high application availability requirements are usually designed to have no single points of failure. However, unforeseen changes or hardware failures can result in single points of failure being introduced to the configuration.

Anytime that a hardware resource (such as a channel, CF link, or sysplex time source) is lost, a message is sent to the console. However, in today's world, most large systems are operated from automation consoles. Those consoles typically only display messages that the installation has defined as being important. If you do not tell your automation which messages are important and should raise an alert, it is possible that a failure can go unnoticed, resulting in your system running for days, weeks, or even months with a single point of failure and you being unaware of it.

For this reason, we recommend three actions:

- ▶ Spend time identifying critical messages and ensure that they will generate an alert on the automated operations console. To help you get started, Appendix A, "Important sysplex messages" on page 171, contains some suggested messages.
- ▶ Assemble automation code that proactively checks your configuration on a regular basis to ensure that there are no single points of failure. This might consist of a set of D M=CHP, D CF, and D ETR commands, for example, and code to compare the response to the expected configuration.
- ▶ Create *and maintain* a CONFIGxx member in Parmlib. This can then be used together with the D M=CONFIG command to verify that the current configuration matches the expected configuration.

IOSSPOF service

z/OS 1.10 introduced a system service called IOSSPOF. This service can check for single points of service between a system and a data set (or pair of data sets) or between a system and a DASD device (or pair of devices). This service is used by the XCF_CDS_SPOF health check to look for single points of failure between the active couple data sets.

If you want to use this service to check for single points of failure between resources that are critical to you, you can use a batch interface that can be downloaded from the z/OS Tools and Toys website at:

<http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxalty2.html>

The IOSSPOFD batch interface lets you easily use the IOSSPOF service to check for single points of failure in your DASD configuration.

Because Coupling Facilities use a completely different communication protocol than DASD, the IOSSPOF service cannot be used to check for single points of failure in the CF configuration. However, the output from the D CF command reports whether there are single points of failure between the system the command is issued on and the CF that is being reported on.

1.2.4 Murphy's Law

Murphy's Law states that if something *can* go wrong, it probably *will*. And at the worst possible time.

The best way to defend yourself against Murphy's Law is to proactively search for and address single points of failure, before Murphy's Law can intervene and fail that last component on you.

We provide a small example. A data center was doing a reorganization of the computer room in preparation for building work. They had two sysplex timers and had to move both of them. Understanding that z/OS and the supporting hardware would switch to the other sysplex timer should either one stop, they randomly unplugged one of the timers. *Everything* stopped.

As they subsequently discovered, the other sysplex timer had failed months previously, and the systems had issued a message at the time, indicating that their signal from that time source had disappeared. The operators even saw the message. However, not knowing what it meant, and having determined that everything was still running fine, they simply deleted the message and carried on.

What could these people have done to protect themselves?

- ▶ They could have implemented automation that would generate an alert when the message saying that a sysplex timer had broken was issued. Ideally, this would also send an email, a text, or both to the responsible system programmers as well.
- ▶ They could have implemented automation that would regularly verify that both sysplex timers were still active.
- ▶ They could have checked that sysplex timer 2 was working before they pulled the plug on sysplex timer 1.

It is human to make mistakes. However, not implementing as much protection as possible to prevent mistakes from having such a large impact can result in outages such as this.

Another valuable lesson? If you are about to stop something that has a backup or an alternate, *always* check that the backup is working before you stop it.

1.2.5 When a backup will not help you

One of the rules of XCF is that if you want to dynamically switch to a new couple data set, the data set that is being added must be formatted to support at least as many objects as the one that it is replacing.

Let us say that you define a primary, alternate, and spare couple data set now, and you implement automation to automatically add the spare data set if either the primary or the alternate are lost. So far, so good.

Six months from now, your workload is growing, so you update the primary and alternate couple data sets to support a larger environment. However, you forget to update the spare. One month later, the device containing the primary CDS fails. XCF automatically switches to the alternate and your automation kicks in and tries to add the spare. However, because you forgot to update the spare, the add fails, and the result is that you now are running with just a single couple data set—a single point of failure.

The lesson from this example is that if you commit to create a backup for something, you must also commit to maintaining that at the same level as the entity that it is backing up. This principle applies to couple data sets and to subsystems (a V8 DB2 subsystem might not be an effective backup for a V9 DB2 if the V9 DB2 is using features that are not available in V8), and to processors (a 1000 MIPS processor cannot effectively back up a 5000 MIPS processor that is running at full capacity), and to many other things for which you might have a backup.

1.2.6 IPL frequency

System programmers are frequently faced with a challenge in relation to how frequently they can or should IPL their systems. On one side, you have the users, who want few, if any, application outages. On the other side, to deliver the levels of availability required by those same users, you have to keep your software up to date, apply preventive and HIPER service, and make other changes, all of which require an IPL. Also, even though many changes can be implemented dynamically (see *z/OS Planned Outage Avoidance Checklist*, SG24-7328, for more information), you might not want to run for extended periods without verifying that those changes have been accurately reflected in your system definitions.

In an ideal world, the frequency with which you IPL would be determined purely by technical requirements. For example, some systems tend to require service more frequently, whereas others might be very stable. Certain systems might be suffering from a memory leak problem that requires more frequent IPLs than other systems.

However, consider what would happen if you could IPL without impacting application availability. As long as their applications are available, users do not know or care what is happening in relation to system IPLs. You would then be in a position that the IPL frequency could be decided based on purely technical considerations. This allows you to be more responsive to user requirements (for new products or new functions), provide a more stable environment (by applying service in a timely manner), and still provide the levels of service that the business requires.

If you have implemented data sharing and dynamic workload balancing for all your business-critical applications, start trying to separate the topic of system IPLs from application outages. Management is naturally reticent about doing anything that can impact users. So if you can preface all IPL discussions by showing how the applications will continue to be available across the IPL (by exploiting data sharing and dynamic workload balancing), it should be possible to move to having IPLs when they are actually needed (which might be every month, or might be every six months).

Best practice for IPL frequency: Customers often ask IBM what is the recommended IPL frequency. The correct answer is “it depends.” The optimum IPL frequency varies not only from one customer to another, but from one system to another.

The optimum IPL frequency discussion should be broken in two points:

- ▶ How often you are *able* to IPL
- ▶ How often the system in question *should* be IPLed

The exploitation of data sharing and workload balancing lets you separate these questions from the issue of application outages.

1.2.7 Rolling IPLs

If your business is dependent on two of something (for example, two DB2s, two IMSs, two z/OSs, two CFs), then it is prudent not to change both things at the same time. If your applications support data sharing, and you find that a change to one IMS, for example, has introduced a problem, you can at least keep your applications up and running with the other IMS until you are able to schedule an outage window to back out the change.

For this reason, we suggest using the rolling IPL concept for any changes that you need to make. The change can be to the operating system, a CF, or one of your processors. The concept applies to any component that you have two or more of in your configuration. It assumes, that the before and after levels are compatible with each other. As long as the products in question meet that requirement, it should be possible to roll out a change to your least important system first and let it *cook* there for a while, then roll it out to the next most important system, and so on, until the change has been implemented on all systems. It is our experience that the majority of mainframe customers upgrade their systems in this manner, and we encourage the remaining customers to strongly consider this methodology as well.

As part of its normal testing of new releases and fixes, IBM runs a sysplex with three releases of z/OS and two releases of the major subsystems to ensure that running multiple releases concurrently in the sysplex does not raise any issues. For more information about the testing done by the Consolidated Service Test team, see:

<http://www.ibm.com/servers/eserver/zseries/zos/servicetst/mission/>

1.2.8 Sysplex separation

Most installations do not normally run test and production applications in the same CICS or IMS region. There are a number of reasons for this, but the most common is that you do not want a misbehaving test application to impact your production service.

For similar reasons, we suggest that you consider segregating your production work into a dedicated Parallel Sysplex if possible. Of course, there are customers who run both production and development work in the same sysplex. But the best practice is to try to keep your production systems separate from your other environments.

Separate DASD pools

As an extension to the concept of separating the production systems into their own sysplex, we also suggest that the DASD belonging to the production sysplex not be accessible to any system outside the sysplex. One obvious reason is security—you do not want someone on a system that is using one security database accessing volumes that are intended to be protected by another security database.

In fact, you might even want to go a step further than simply varying the production DASD offline to any system outside the sysplex. You might want to have further separation by removing the production volumes from the OS Config in the IODF of any other systems. One excellent reason for doing this is to eliminate the possibility of someone accidentally damaging a production volume from a non-production system. While such events are rare, they are possible if another system has physical access to a production volume. It is a little extra work to set up a separate OS Config for the non-production systems, but well worth the investment if it saves you from a costly mistake.

1.3 Layout of this document

The remainder of this book provides information about ways to maximize the availability of z/OS and the major IBM subsystems.

Most installations' technical support teams are structured by subsystem, with system programmers having an area of specialization. For that reason, we structured this paper along similar lines.

This first chapter contained concepts that apply equally across all the subsystems. Chapter 2, "z/OS sysplex best practices" on page 9, addresses z/OS. That chapter will be of interest, not only to the z/OS system programmers, but also to the subsystem programmers because of the close relationship between z/OS setup and the subsystems.

The remaining chapters are dedicated to one subsystem each. We hope that this structure allows our readers to obtain the maximum useful information from the book, in the minimum time.



z/OS sysplex best practices

This chapter provides the latest guidance and best practices for the components of z/OS to optimize performance and availability when operating in a Parallel Sysplex.

2.1 Introduction

The target audience for this chapter is skilled systems programmers who have hands-on experience with operating and managing a Parallel Sysplex.

If you would like background information about the topics discussed here, an excellent starting place is *z/OS MVS Setting Up a Sysplex*, SA22-7625.

z/OS consists of many separate components. Some of these behave in exactly the same way regardless of whether they are run in a multi-system sysplex environment or in a monoplex. However, some (XES, for example) *only* operate in a Parallel Sysplex environment, whereas others (XCF, for example) can deliver additional function when run in a Parallel Sysplex. This chapter contains best practices for the major z/OS components that support or exploit the Parallel Sysplex function.

The IBM Health Checker for z/OS provides many checks to verify that your configuration conforms to IBM best practices. IBM recommends that all checks applicable to your environment are enabled and that any exceptions reported by the checks are addressed as soon as possible. In the specific best practices in this chapter, we highlight any checks related to that best practice. The Health Checker is discussed in more detail in 2.7, “z/OS Health Checker” on page 78.

GDPS® customers: Because of its specific sysplex configuration requirements and the functions that it provides, the recommended settings for GDPS customers are sometimes different from the *normal* best practices. If you are a GDPS customer, *always* check the GDPS product documentation to ensure that any changes that you might consider making are in line with the GDPS recommendations or requirements.

2.2 XCF

The Cross System Coupling Facility (XCF) component of z/OS plays a fundamental role in a Parallel Sysplex. If XCF encounters a severe problem, this could impact the entire sysplex. Because of the role XCF plays in communication within the sysplex, it is important that XCF is configured to deliver the best possible performance.

XCF is designed to be very resilient, with the ability to automatically recover from the loss of one or more of its resources. While this resiliency helps XCF deliver very high levels of availability, there is a potential downside—it is possible to find that you are running with a single point of failure in your XCF configuration, not because you intended it to be that way, but rather because a component failed and the XCF recovery was so good that no one noticed.

There is another concern related to XCF's resiliency. Because XCF tends to just tick over quietly, automatically handling problems that it encounters, it typically does not receive much attention. Even in a configuration that is suboptimal, XCF still functions and the fact that its performance might not be as good as it could be might go unnoticed.

Therefore, the best way to avoid XCF problems and to deliver the best performance is to build a configuration that adheres to IBM best practices for XCF. This section describes those best practices.

2.2.1 Couple data sets

At a very high level, a Parallel Sysplex can be viewed as sort of a z/OS of z/OSs. That is, just as a single z/OS system consists of multiple address spaces, all under the control of a single operating system, if you move that up a level to a Parallel Sysplex, you might equate systems in a Parallel Sysplex to address spaces in a single z/OS. Just as the operating system maintains a consistent view of the activity of all address spaces under its control, similarly, in a sysplex, there is a need for coordination across all members of the sysplex. The primary mechanism for providing that level of coordination is the various XCF-managed couple data sets. For this reason, the performance and availability of the couple data sets are critical.

Every CDS should have a primary, an alternate, and a spare

XCF provides support for a primary and alternate version of each couple data set (CDS) that it manages. In the event of a primary CDS becoming unavailable for any reason, XCF automatically switches to the corresponding alternate CDS.

Before XCF accepts the addition of a new alternate couple data set, it checks that the new data set is formatted with at least the same features as the current primary CDS, and also that it is formatted to support the same or a larger number of objects. If the new data set meets these criteria, it is added as the new alternate.

However, if an existing primary CDS becomes unavailable and XCF automatically switches to the alternate, the sysplex is now operating with only a primary CDS, and therefore has a single point of failure. For this reason, you should have a predefined spare for every CDS.

And, just as the alternate CDS should be formatted to support the same number of objects as the primary CDS, the spare CDS should be formatted in the same way. If the spare CDS is formatted for a smaller number of objects, the attempt to add the spare as the new alternate fails. Also, because the spare CDS is not known to XCF, there is no way to find out how it was formatted. Therefore, to ensure that your spare CDSs are always compatible with the in-use ones, alter your procedures to ensure that every time that you format a new primary and alternate CDS, you also format a new spare CDS at the same time.

Health check: The XCF_CDS_SPOF health check (introduced in z/OS 1.10) issues a warning message if it discovers that there is only one of any of the in-use CDS types (Sysplex, CFRM, LOGR, and so on). However, because the *spare* CDSs are not defined to XCF in advance of being added as new alternate CDSs, there is no way for the health check to verify whether you have defined spare CDSs. Note that as well as checking for just a single instance of each CDS, it also checks that there are no single points of failure in the I/O configuration for each CDS.

Enable automation to add the spare couple data set

Having defined a spare CDS, you want to be sure that if XCF ends up running with just a single CDS, the spare is added as the new alternate as quickly as possible. If XCF detects that it does not have an active alternate for a CDS, it issues an IXC267E message (Example 2-1). The message informs you that XCF is now running with a single CDS for that function. It also tells you the type of CDS (SYSPLEX, in this example). You can use the information from this message to have your automation product issue a SETXCF command to add your spare CDS as the new alternate CDS.

Example 2-1 Message indicating that XCF has a CDS with no alternate

```
*IXC267E PROCESSING WITHOUT AN ALTERNATE 481
COUPLE DATA SET FOR SYSPLEX.
ISSUE SETXCF COMMAND TO ACTIVATE A NEW ALTERNATE.
```

The use of a well-structured CDS naming convention makes it easy to create generic automation code that parses the CDS type from the IXC267E message to handle the loss of any CDS. For example, using CDS names such as `hlq.plexname.cdstype.CDSx`, where:

hlq	Is the HLQ for all CDSs.
plexname	Is the sysplex name (to minimize the chances of a CDS being accidentally added to the wrong sysplex).
cdstype	Is the CDS type. The values used for this qualifier should match the names used in the output from the D XCF,C command (ARM, BPXMCDS, CFRM, and so on).
CDSx	Indicates whether this is the primary (CDS1), alternate (CDS2), or spare (CDS3/4) CDS.

While XCF does not include a function to automatically add spare CDSs, IBM Tivoli® System Automation *does* include this capability, and presumably other vendors' automation products also have this ability. We suggest that you enable this function in your automation product if it exists. If not, it is easy to write your own automation to provide this function.

We also suggest that the same automation routine that is responsible for reacting to the IXC267E message also include certain process (such as an email) for ensuring that the system programmer (and his or her backup) responsible for the CDSs is made aware that a spare CDS had to be used. The trigger event for the IXC267E message should be identified and the CDSs brought back to their normal configuration as soon as possible.

It might also be worthwhile to have proactive automation that checks that the CDSs in use are the CDS1 (normal primary) and CDS2 (normal alternate). If it finds that CDS3 or CDS4 (the spares) is being used, it should email the responsible system programmers so that they can determine why the spare CDS is being used and bring the configuration back to normal.

Keep primary, alternate, and spare separate from each other

Having gone to the trouble of defining primary, alternate, and spare CDSs, you want to do your best to ensure that a single failure does not make more than one instance of a given CDS type unavailable. There is little point in having a spare CFRM CDS on the same volume as the primary CFRM CDS, as a single failure affecting that volume can make both data sets unavailable. Therefore, attempt to distribute the CDSs across as many physical storage subsystems as possible or reasonable. Figure 2-1 shows one possible example of how CDSs can be allocated over a number of volumes. This configuration assumes that none of the CDSs are part of a synchronous remote copy configuration. See “Multi-site sysplex considerations” on page 13 for more information about mirroring CDSs.

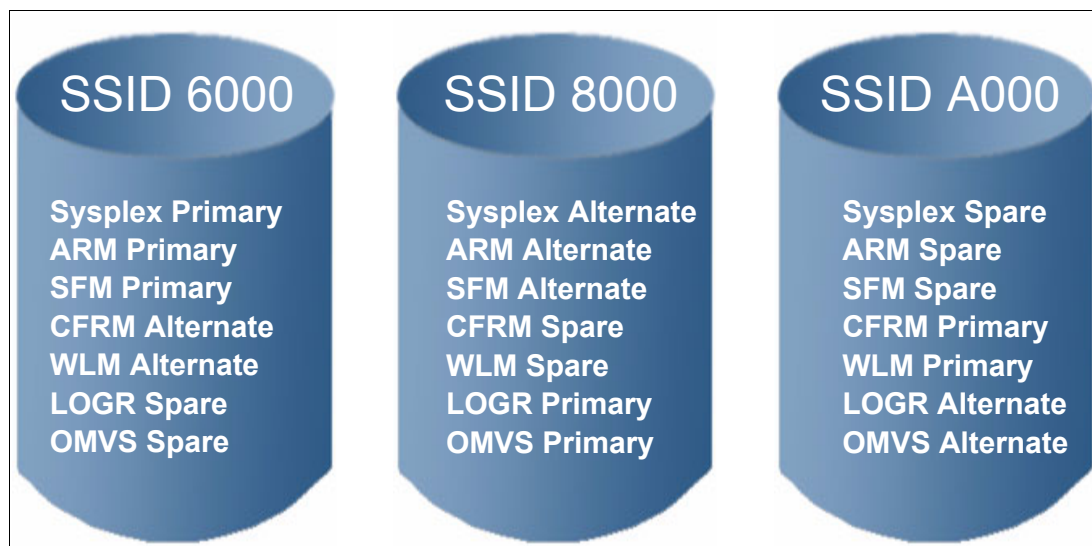


Figure 2-1 Possible CDS placement

Health check: The XCF_CDS_SPOF health check issues a warning message if it discovers a single point of failure between any of the in-use CDS data sets. However, because the spare CDSs are not defined to XCF in advance, the health check has no way of determining whether the spare CDS has any single points of failure in common with either of the other CDSs.

Use your automation or batch scheduling product to regularly submit a job that executes the IOSSPOFD program (described in 2.6, “Avoiding single points of failure” on page 76) to monitor for single points of failure related to your spare CDSs.

Multi-site sysplex considerations

If you have a multi-site sysplex, give special consideration to the placement of the CDSs across the two sites.

Important: The GDPS product documentation contains recommendations that are specific to GDPS. If you are a GDPS customer, follow the recommendations for your particular flavor of GDPS. Other disaster recovery offerings might similarly have specific recommendations about the placement of the CDSs. If you use such an offering, follow the recommendations provided by the vendor.

If you have a multi-site sysplex and are *not* using GDPS or a competitive offering, consult your DASD vendor for its position regarding mirroring the XCF CDSs. According to *z/OS MVS Setting Up a Sysplex*, SA22-7625, “Mirroring couple data sets using peer-to-peer remote copy (PPRC) or similar OEM mechanisms for synchronous data copy is not recommended for the Sysplex or CFRM couple data sets.” Based on that recommendation, you might decide to place each primary CDS in the opposite site to the alternate CDS. In that case, define a spare CDS in each site so that in the event of a site failure, you still have a spare CDS available for use. Figure 2-2 shows an example of such a configuration.

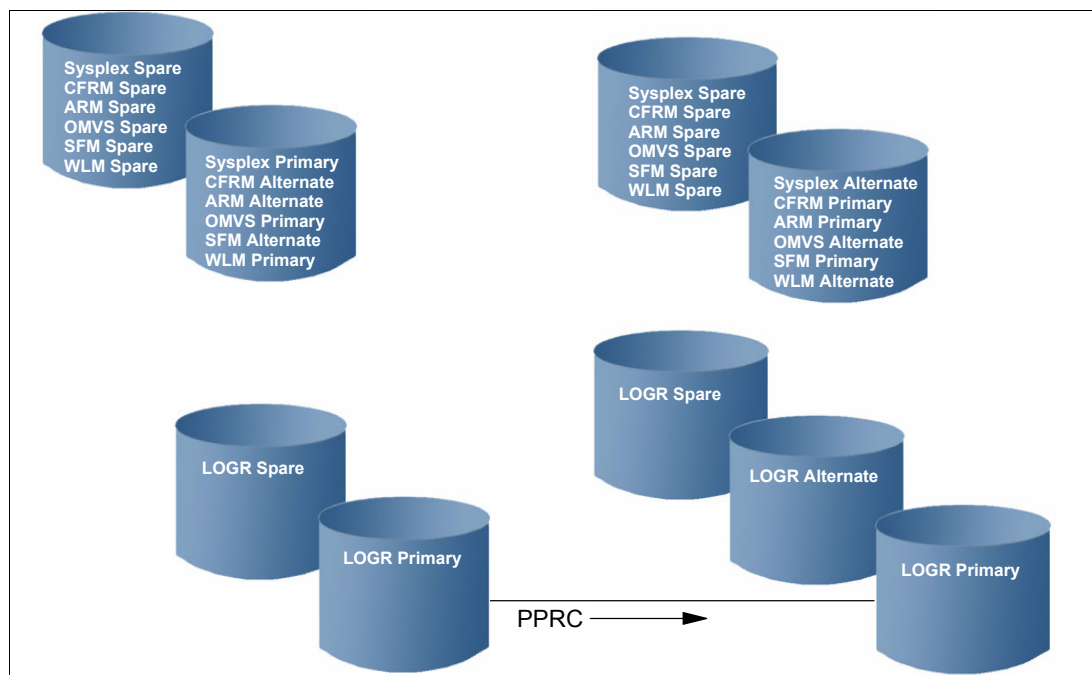


Figure 2-2 Possible CDS configuration for multi-site sysplex

For the CDSs other than the Sysplex and CFRM ones, the use of a synchronous mirroring solution is considered acceptable, although it might be easier to treat these data sets in the same way as the Sysplex and CFRM CDSs.

If you have a mechanism for freezing the secondary DASD in case of a suspected disaster and you need to start your systems using the frozen DASD, it is important that the LOGR CDS reflects the same point in time as the frozen Logger staging and offload data sets and the related subsystem data sets and databases. This means that the LOGR CDS should be remote-copied and included in the same freeze group as the other Logger data sets (staging and offload). For more discussion about mirroring CDSs, see 2.4, “Testing with copies of a sysplex” on page 68.

Separate busy CDSs from each other

While the CFRM and Sysplex CDSs are typically not very busy during normal operation, they can become very busy during recovery from a failure. As a result, placing the primary Sysplex and CFRM CDSs on the same volume can cause performance problems and increase recovery times. To ensure optimal performance, allocate these CDSs on separate volumes.

Health check: The XCF_CDS_SEPARATION health check issues a warning message if it discovers that the Sysplex and CFRM primary CDSs are on the same volume. By default, it also checks that the LOGR primary CDS is on a separate volume from either of these data sets. However, if you are unable or unwilling to do this, you can modify the check to turn off that checking.

If the Sysplex and CFRM CDSs are allocated on volumes that support HyperPAV (but not traditional dynamic PAV), then, from a performance perspective, that should provide acceptable performance. However, the XCF_CDS_SEPARATION health check is not aware of the presence of HyperPAV and still raises an exception if it finds those data sets on the same volume. The HyperPAV function is described in *IBM System Storage® DS8000®: Architecture and Implementation*, SG24-6786.

Place CDSs on volumes with no RESERVEs

It is vital for both performance and availability that all the CDSs can be accessed and updated with very good response times. This is particularly true for the Sysplex and CFRM CDSs. For this reason, it is critical that the CDSs are placed on volumes that will not have any RESERVEs issued against them.

One way to do this is to dedicate the CDS volumes to just CDS data sets. Every access to an XCF-managed CDS is done by the XCFAS address space, and XCFAS never issues a RESERVE, so dedicating the volumes to CDS usage goes a long way toward ensuring that CDS access is not impaired by RESERVEs.

If you want to take it a step further, you can add simple code to your SMS ACS routines that ensures that the only data sets that are allowed to be allocated on these volumes are CDSs. The use of a good naming convention as described in “Enable automation to add the spare couple data set” on page 12 helps keep this code simple.

You also need to consider RESERVEs that might be issued by a backup program if you back up the CDSs. If you are using DFSMSdss for the backups, you can use the ADRUENQ exit to ensure that no RESERVEs are issued by DFSMSdss against these volumes. Note, however, that IBM recommends that you do *not* back up your normal CDSs. Rather, we recommend that a separate set of CDSs are defined for use solely during a disaster. Place these CDSs on a separate set of volumes. They would never be used in normal operation, and therefore can be backed up for use in a disaster. For more information about CDSs and disaster recovery, see 2.4, “Testing with copies of a sysplex” on page 68.

Also consider using the GRS RNLs to ensure that any RESERVEs against the CDS volumes are converted to ENQs. You can achieve this by specifying QNAME(SYSVTOC) RNAME(volser) in the GRS Reserve Conversion RNL for each CDS volume.

To determine whether any RESERVEs are being issued against any of the CDS volumes, you can use the GRS ENQ/RESERVE/DEQ monitor. More information about this monitor can be found in the section titled “Using the ENQ/RESERVE/DEQ monitor tool” in *z/OS MVS Planning Global Resource Serialization*, SA22-7600.

Health check: The GRS_CONVERT_RESERVES health check verifies that GRS is set up to convert all RESERVES to global ENQs. While you might have reasons for not converting all your RESERVES to ENQs, if this check indicates that RESERVES are not being converted, at least you are aware that there is the *possibility* of someone issuing a RESERVE to a CDS volume.

Remember, however, that it is possible that the RESERVE might be issued by a system outside the sysplex, and the GRS settings for that system are obviously not available to a health check running in this sysplex.

Implement MSGBASED CFRM processing

z/OS 1.8 introduced a new mechanism called MSGBASED CFRM processing. When MSGBASED CFRM processing is enabled, the communication associated with recovery or rebuild processing for most CF structures is routed via XCF signalling rather than via the CFRM CDS. This results in much better scalability for the recovery of structures with many connectors, meaning that the recovery time following a CF failure is largely unaffected by the number of systems in the sysplex (Figure 2-3).

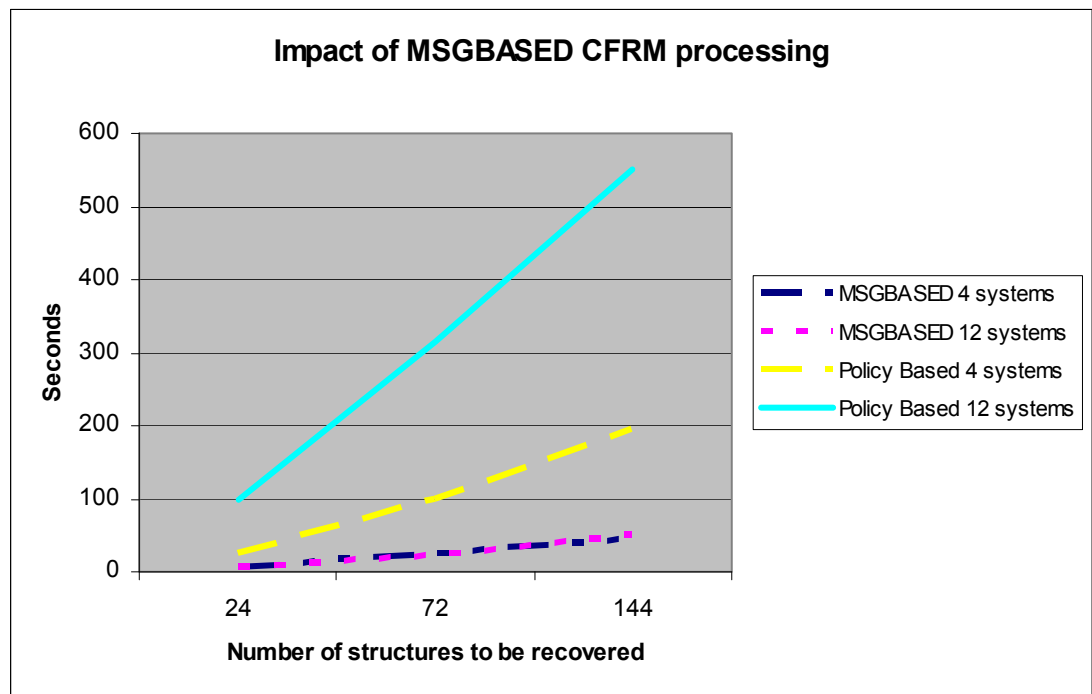


Figure 2-3 Impact of MSGBASED CFRM processing

To use MSGBASED CFRM processing, all members of the sysplex must be running z/OS 1.8 or later, and the CFRM CDSs must be formatted with ITEM NAME(MSGBASED) NUMBER(1). If all members of your sysplex are running z/OS 1.8 or later, enable MSGBASED CFRM processing.

Health check: z/OS 1.12 delivers a new check (XCF_CFRM_MSGBASED) that raises an exception if message-based CFRM processing is not being used in an environment that supports this capability.

CDS format levels

Couple data sets might be viewed as a type of inter-system control block that resides on DASD. Just as the layout of control blocks can change when new functions are introduced, couple data sets sometimes need to be reformatted in support of new functionality. However, support for new format levels usually is not rolled back to previous releases, meaning that the new format CDS cannot be used until all members of the sysplex have migrated to the new release. As a result, installations might forget to move to the new format CDSs, and miss out on valuable new functions.

To help you identify whether your CDSs are on the latest format level, Table 2-1 lists the most current format levels at the time of writing (z/OS 1.11).

Table 2-1 CDS format levels

CDS type	Latest format level
ARM	VERSION 1, HBB7707 SYMBOL TABLE SUPPORT
BPXMCDS	VERSION(2)
CFRM	SMREBLD(1) SMDUPLEX(1) MSGBASED(1)
LOGR	LOGR COUPLE DATA SET FORMAT LEVEL: HBB7705
SFM	N/A
Sysplex	SYSTEM STATUS DETECTION PROTOCOL IS SUPPORTED
WLM	Format 3 ^a

a. For more information about WLM CDS versions, functionality levels, and format levels, see the section titled “Service Definition Functionality Levels, CDS Format Levels, and WLM Application Levels” in *z/OS MVS Planning: Workload Management*, SA22-7602.

To determine the format level of most of your CDSs, issue a **D XCF,C,TYPE=cds_type** command for each CDS (Example 2-2). Information about how to format CDSs and the keywords to use to enable the latest functions is available in *z/OS MVS Setting Up a Sysplex*, SA22-7625.

Example 2-2 Displaying CDS format information

```
D XCF,C,TYPE=SYSPLEX
IXC358I 00.05.37 DISPLAY XCF 897
SYSPLEX COUPLE DATA SETS
PRIMARY DSN: SYS1.TESTPLX.SYSPLEX.CDS01
VOLSER: BH8CD1 DEVN: D170
FORMAT TOD MAXSYSTEM MAXGROUP(PEAK) MAXMEMBER(PEAK)
09/02/2009 13:43:28 4 200 (46) 203 (7)
ADDITIONAL INFORMATION:
ALL TYPES OF COUPLE DATA SETS ARE SUPPORTED
GRS STAR MODE IS SUPPORTED
SYSTEM STATUS DETECTION PROTOCOL IS SUPPORTED
```

The one exception is the WLM CDS. Information about the formatting of the WLM CDS is not provided by the D XCF,C command. To get this information for the WLM CDS, issue the D WLM command. This results in the information shown in Example 2-3.

Example 2-3 Displaying WLM CDS information

```

D WLM
IWM025I 23.31.28 WLM DISPLAY 609
  ACTIVE WORKLOAD MANAGEMENT SERVICE POLICY NAME: WLMPOL
  ACTIVATED: 2009/07/15 AT: 10:49:26 BY: VIG2 FROM: @$3
  DESCRIPTION: MTTR WLM policy
  RELATED SERVICE DEFINITION NAME: WLMMTTR
  INSTALLED: 2009/07/15 AT: 10:49:17 BY: VIG2 FROM: @$3
  WLM VERSION LEVEL: LEVEL023
  WLM FUNCTIONALITY LEVEL: LEVEL013
  WLM CDS FORMAT LEVEL: FORMAT 3
  STRUCTURE SYSZWLM_WORKUNIT STATUS: CONNECTED
  STRUCTURE SYSZWLM_DE502097 STATUS: CONNECTED

```

CDS sizes

Periodically, particularly before and after adding a new workload or after significant changes in workload volume, check that your CDSs are large enough to contain the information required to control your sysplex.

However, you also need to be careful not to oversize the CDSs. Having a Sysplex CDS that is far larger than necessary can result in an IPL taking longer than necessary. This is because XCF processes the entire data set, even if it only contains a small amount of information. Migrating to a larger CDS can be done non-disruptively. You simply define the new CDSs and roll them into production using the SETXCF COUPLE commands. However, moving to *smaller* CDSs requires a sysplex IPL. For this reason, it is a good idea to grow your CDS in small increments, rather than max'ing out the data set to avoid having to worry about how full it is.

Getting size and usage information for the Sysplex CDS

To get information about how the Sysplex CDS is formatted and its actual usage, issue a D XCF,C command (Example 2-4). Check the MAXGROUP(PEAK) and MAXMEMBER(PEAK) values against the maximum number of groups and members that the CDS can support to ensure that the maximum values are comfortably larger than the peak values.

Example 2-4 Displaying Sysplex CDS space usage

```

D XCF,COUPLE
IXC357I 08.52.28 DISPLAY XCF 840
SYSTEM SYS#3 DATA
....
SYSPLEX COUPLE DATA SETS
PRIMARY DSN: SYS1.FKPLX2.SYSPLEX.CDS01
VOLSER: @$#X1 DEVN: D20F
FORMAT TOD MAXSYSTEM MAXGROUP(PEAK) MAXMEMBER(PEAK)
05/30/2008 17:51:57 12 500 (51) 259 (111)
ALTERNATE DSN: SYS1.FKPLX2.SYSPLEX.CDS02
VOLSER: @$#X2 DEVN: D30F
FORMAT TOD MAXSYSTEM MAXGROUP MAXMEMBER
05/30/2008 17:51:59 12 500 259

```

Unfortunately, it is not as easy to get information about the formatting of the other CDSs or the peak usage of those CDSs. That information must be obtained using methods unique to each CDS type.

Also, ensure that you enable the XCF_SYSPLEX_CDS_CAPACITY health check to monitor the space usage of the Sysplex CDS (the most important of the XCF-managed CDSs).

Health check: The XCF_SYSPLEX_CDS_CAPACITY health check issues a warning message if it discovers that the Sysplex CDS is not large enough to handle an increase in the number of systems, XCF groups, or XCF members.

Note, however, that while other CDSs are also formatted to support a certain number of objects, this check does not check the space usage of those CDSs.

Getting size and usage information for the ARM CDS

To determine the size and usage of the ARM CDS, run an IXCMIAPU job, specifying DATA TYPE(ARM) (Example 2-5).

Example 2-5 Sample JCL to get LOGR CDS usage information

```
//LOGRLIST JOB (0,0),'LIST LOGR POL',CLASS=A,
//          MSGCLASS=X,NOTIFY=&SYSUID
//STEP1    EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSIN    DD *
          DATA TYPE(LOGR) REPORT(YES)
/*
```

This results in a report similar to that shown in Example 2-6.

Example 2-6 IXCMIAPU report for ARM CDS

```
/* ARM Policy Report */

/* XCF Format Utility Control Cards: */
DATA TYPE(ARM)
  ITEM NAME(POLICY)  NUMBER(20)
  ITEM NAME(MAXELEM) NUMBER(200)
  ITEM NAME(TOTELEM) NUMBER(200)

/* 2 Administrative Policies Defined */
```

You can see that the report shows the number of policies and elements that the primary CDS is formatted to support. It also tells you how many policies there are in the CDS. Unfortunately, the number of defined elements is not reported. You need to scan the remainder of the report and manually count the number of elements.

MAXELEM determines the maximum number of ELEMENT statements that you can specify in an ARM policy. While ARM does not provide a count of the number of ELEMENT statements in your policy, to easily determine the number of ELEMENTs, you can edit the policy in ISPF and enter FIND ELEMENT(ALL. This results in just the lines containing the ELEMENT(keyword being displayed. If you attempt to define a policy with more ELEMENT statements than MAXELEM allows, IXCMIAPU gives you a failing return code.

The other keyword used to format the ARM CDS, TOTELEM, determines the maximum number of elements that can be concurrently *registered* with ARM.

To determine the number of elements that are *currently* registered with ARM, issue the D XCF,ARMS command (Example 2-7). In this example, you can see that the ARM CDS is formatted to support up to 200 registered elements, and that at the time the command was issued, 12 elements had registered. Issue this command at a time when you expect the largest number of ARM-supporting components to be active because the number reported is the *current* number, not the *peak* number.

Example 2-7 Displaying ARM CDS usage information

```
D XCF,ARMS
IXC392I 00.10.22 DISPLAY XCF 736
ARM RESTARTS ARE ENABLED
----- ELEMENT STATE SUMMARY ----- -TOTAL- -MAX-
STARTING AVAILABLE FAILED RESTARTING RECOVERING
              3           9           0           0           0           12           200
```

Note: The information in the output from the IXCMIAPU program reflects the primary CDS. It is possible that the alternate CDS is formatted for larger values. However (except for the Sysplex CDS), there is no way to display this information. This is why it is so important to have a process whereby all three data sets (primary, alternate, and spare) are allocated when you allocate a new CDS.

Getting size and usage information for the BPXMCDS CDS

To determine how the primary BPXMCDS CDS is sized *and* the actual current usage of that CDS, issue the F BPXOINIT,FILESYS=DISPLAY,GLOBAL command. This results in the information shown in Example 2-8.

Example 2-8 Displaying BPXMCDS usage information

```
-f bpxoinit,filesys=display,global
BPXM027I COMMAND ACCEPTED.
BPXF040I MODIFY BPXOINIT,FILESYS PROCESSING IS COMPLETE.
BPXF242I 2010/02/02 09.25.47 MODIFY BPXOINIT,FILESYS=DISPLAY,GLOBAL
SYSTEM  LFS VERSION ---STATUS----- RECOMMENDED ACTION
FK08    1. 10. 0 VERIFIED                NONE
FKFS    1. 10. 0 VERIFIED                NONE
FK07    1. 10. 0 VERIFIED                NONE
FKFT    1. 10. 0 VERIFIED                NONE
FK04    1. 10. 0 VERIFIED                NONE
FK03    1. 10. 0 VERIFIED                NONE
FK10    1. 10. 0 VERIFIED                NONE
FK05    1. 10. 0 VERIFIED                NONE
FK06    1. 10. 0 VERIFIED                NONE
FK09    1. 10. 0 VERIFIED                NONE
FK12    1. 10. 0 VERIFIED                NONE
CDS VERSION= 2      MIN LFS VERSION= 1. 10. 0
DEVICE NUMBER OF LAST MOUNT= 638
MAXIMUM MOUNT ENTRIES= 10000 MOUNT ENTRIES IN USE= 234
MAXIMUM AMTRULES= 500 AMTRULES IN USE= 2
MAXSYSTEM= 16
```

Note that the IN USE information reflects the current usage, not the peak. Therefore, issue this command at a time when you expect usage of the OMVS file systems to be at a peak.

Getting size and usage information for the CFRM CDS

To determine the allocation attributes and the current usage of the CFRM CDS, you must run an IXCMIAPU job with the DATA TYPE(CFRM) REPORT(YES) option.

Example 2-9 shows sample output from this program. The top part of the report shows how the primary CDS is formatted. The next part of the report provides information about each defined policy, with the number of structures and Coupling Facilities. If you have multiple policies defined, scroll down through the report, searching for DEFINE POLICY NAME, and checking the information for each policy. Note that the output also tells you about formatting options that are supported by this release of z/OS, but that you are *not* currently using (MSGBASED SUPPORT RECORD IS NOT PRESENT in this example).

Example 2-9 Displaying CFRM CDS usage information

```

/* XCF Format Utility Control Cards: */
DATA TYPE(CFRM)
  ITEM NAME(POLICY) NUMBER(8)
  ITEM NAME(STR) NUMBER(150)
  ITEM NAME(CF) NUMBER(16)
  ITEM NAME(CONNECT) NUMBER(64)
  ITEM NAME(SMREBLD) NUMBER(1)
  ITEM NAME(SMDUPLEX) NUMBER(1)
/* MSGBASED SUPPORT RECORD IS NOT PRESENT. */

DEFINE POLICY NAME(POLICY1 )
  /* Defined: 01/15/2010 09:35:15.514695 User: SYDB */
  /* Version: 0 */
  /* 110 Structures defined in this policy */
  /* 5 Coupling Facilities defined in this policy */

```

Getting size and usage information for the LOGR CDS

To determine the allocation and current usage information for the primary LOGR CDS, you need to run an IXCMIAPU job, specifying DATA TYPE(LOGR) REPORT(YES). The resulting report shows how the CDS was formatted (in the Formatted column) and the actual usage of each item. Example 2-10 shows a sample report.

Example 2-10 Displaying LOGR CDS usage information

```

LOGR COUPLE DATA SET FORMAT LEVEL: HBB7705
/* Functional Items: */
/* SMDUPLEX(1) */

```

Type	Formatted	In-use
-----	-----	-----
LSR (Log Stream)	80	48
LSTRR (Structure)	16	15
DSEXTENT (Data Set Extent)	20	12

In this example, you can see that there is plenty of scope for defining more log streams and for another eight log streams to grow to more than 168 extents. However, if more than one new Logger structure is added, a new set of CDSs with larger LSTRR values must be created.

Getting size and usage information for the SFM CDS

To determine how the SFM CDS is formatted and the current usage of the CDS, you must run an IXCMIAPU job, specifying DATA TYPE(SFM) REPORT(YES). This results in a report similar to that shown in Example 2-11.

Example 2-11 Displaying SFM CDS usage information

```
/* SFM Policy Report */

/* XCF Format Utility Control Cards: */
DATA TYPE(SFM)
  ITEM NAME(POLICY) NUMBER(50)
  ITEM NAME(SYSTEM) NUMBER(32)
  ITEM NAME(RECONFIG) NUMBER(50)

/* Active Policy: NAME(SFMPOL1) CONNFALL(YES)
   Defined:    05/07/2006 00:59:42.071857 User:FPK
   Activated:  05/07/2006 01:04:04.383316

SYSTEM NAME(*)
  ISOLATETIME(0)
  MEMSTALLTIME(900)
  SSUMLIMIT(900)

2 System Definitions in this Policy

SYSTEM NAME(FKC1)
  WEIGHT(10)

SYSTEM NAME(FKC2)
  WEIGHT(10)

0 Reconfig Definitions in this Policy

End of Active Policy */
```

The top part of the report shows how the primary CDS is formatted and the second part shows the actual current usage. In this example, the CDS is formatted to support up to 50 policies, but only one policy has been defined. It is also formatted to allow you to specify individual overrides for up to 32 systems, but only two systems actually have an override specified (as denoted by 2 System Definitions in this policy).

Getting size and usage information for the WLM CDS

Unlike the other CDSs, which support the use of XCF utilities to display their usage information, you must use the WLM ISPF application to get this information for the WLM CDS.

The first thing to do is to go into the WLM application in ISPF. There are two separate steps for obtaining the allocation and usage information. To determine how the WLM CDS is formatted, select the **Utilities** pull-down menu, and then **Allocate new CDS using existing CDS values**. This presents you with a window similar to that shown in Figure 2-4. You can see that the values for the various allocation attributes are filled in based on how the existing WLM CDS is allocated. Make a note of these values.

Session C - gdps mop whitescreen.ws - [43 x 80]

File Edit View Communication Actions Window Help

File Utilities Notes Options Help

Command ==> Allocate couple data set for WLM using CDS values

Sysplex name _____ (Required)

Data set name _____ (Required)

Size parameters (optional):

Service policies . . . 10	(1-99)
Workloads . . . 35	(1-999)
Service classes . . . 100	(1-999)
Application scheduling . . . 50	(1-999)
environments . . . 50	(1-999)
VDEF extensions . . . 5	(0-80092)
VDEF extensions . . . 5	(0-80092)
VDEF extensions . . . 5	(0-80092)
VDEF extensions . . . 5	(0-80092)

Storage parameters:

Storage class . . . _____

Management class . . . _____

or

Volume 'data set?' . . . _____ (Y or N)

Catalog 'data set?' . . . _____ (Y or N)

Size parameters are initialized from service definition in the WLM couple data set. (IWMAM861)

MA C

Connected to remote server /host 9.12.6.50 using lu/pool SC38TC87 and HP DeskJet 890C on LPT1:

Figure 2-4 Determining WLM CDS allocation values

To determine the actual usage of each of these objects, go back to the primary WLM menu and select the **File** pull-down menu, and then select **Print**. The usage information is printed to the selected destination. The resulting report will be similar to that shown in Figure 2-5.

```

Session C - gdps mop whitescreen.ws - [43 x 80]
File Edit View Communication Actions Window Help
Menu Utilities Compilers Help
BROWSE KYNEF.##$2.SPF3.LIST Line 00000000 Col 001 080
Command ==>
***** Top of Data ***** Scroll ==> CSR
* Service Definition WLM - MTTR WLM Service Definition
7 workloads, with 25 service classes
4 resource groups
1 service policy
1 classification groups
1 subsystem types
3 report classes
10 application environments
4 scheduling environments
6 resources

CPU coefficient of 1.0
I/O coefficient of 0.1
MSD coefficient of 0.1000
SRB coefficient of 1.0

I/O priority management Yes
Dynamic alias tuning management Yes

+-----+
|Notepad information|
+-----+

MA c
Connected to remote server/host 9.12.6.50 using lu/pool SC38TC87 and HP DeskJet 890C on LPT1:

```

Figure 2-5 WLM CDS usage report

Note: z/OS Migration, GA22-7499, for z/OS 1.10 states that a new set of WLM CDSs should be allocated when you move to z/OS 1.10. This is because the size of certain entries in the CDS have increased in z/OS 1.10. Note, however, that z/OS 1.10 creates a CDS with the same format level (3) as earlier releases.

Specify consistent MAXSYSTEM values

One of the keywords that must be specified when formatting any of the XCF-controlled CDSs is MAXSYSTEM. While it is possible to have different MAXSYSTEM values in each of the different CDSs, it is best to use the same value for every CDS. More specifically, you want to ensure that the MAXSYSTEM value in the function CDSs is at least as large as that specified for the Sysplex CDS. If one of the function CDSs has a smaller MAXSYSTEM value, you might find that a system is allowed to join the sysplex (because the new number of systems in the sysplex is still less than the MAXSYSTEM value in the Sysplex CDS). However, it is not allowed to use certain function CDS because the number of systems in the sysplex now exceeds the MAXSYSTEM value in the corresponding function CDS.

Health check: The XCF_CDS_MAXSYSTEM health check (delivered in z/OS 1.12) issues a warning message if it discovers that the MAXSYSTEM value for any of the function CDSs is smaller than the MAXSYSTEM value in the Sysplex CDS.

2.2.2 XCF signalling paths

One of the other significant functions of XCF is to assist in communication between programs in a sysplex. The communicating programs might be on the same system, or they might be on different systems. One of the advantages of using XCF is that the exploiters do not need to be aware of the location of their peer. They only need to know the XCF member name of the peer.

XCF signalling is used by many IBM and non-IBM software products. Table 2-2 contains a list of IBM software products and functions that exploit XCF. This list is not comprehensive. However, it gives you an idea of the importance of the products that depend on XCF.

Table 2-2 IBM exploiters of XCF signalling

Exploiter	Exploiter	Exploiter	Exploiter	Exploiter
APPC, ASCH	CICS MRO	CICSplex® SM	CICS VR	Console
DAE	DB2	DFSMS	ENF	GRS
DFSMSHsm	IMS	GDPS	Tivoli System Automation	IOS
IRLM	JESXCF	JES2	JES3	MQ Series
OAM	OMVS	RACF®	RMF	RRS
Any lock structure	Tape sharing	TCP/IP	TWS	Trace
TSO Broadcast	VLF	VSAM/RLS	VTAM®	WLM
XES	zFS	XCF		

To deliver the expected levels of service, it is vital that XCF signalling is both robust and capable of delivering good response times. You also need to keep the transport class infrastructure in mind when looking at the type and number of links that you will provide.

Signalling path types

XCF signalling paths can use either dedicated CTC devices or signalling structures in the CF. For the majority of customers, having just signalling structures in the CFs should provide the best performance *and* has the advantage that defining the paths is far easier than defining CTCs. For these reasons, we advocate using CF structures for XCF communication.

If you *do* want to define CTCs for use by XCF (as a backup to the CF structures), use FICON® CTCs rather than ESCON® CTCs, define two CTC control units on each channel, and have a PATHOUT device and a PATHIN device on both control units. Also, do not be surprised to find that XCF sends far more requests via the CF structures than via the CTC devices. XCF always prefers the path that is delivering the shortest transfer times. You can get the number of messages being sent over each path, and the transfer time for each path, from the RMF XCF Path Statistics report. Figure 2-6 shows the signalling rates that were achieved for various types of XCF signalling paths in a recent performance test in IBM.

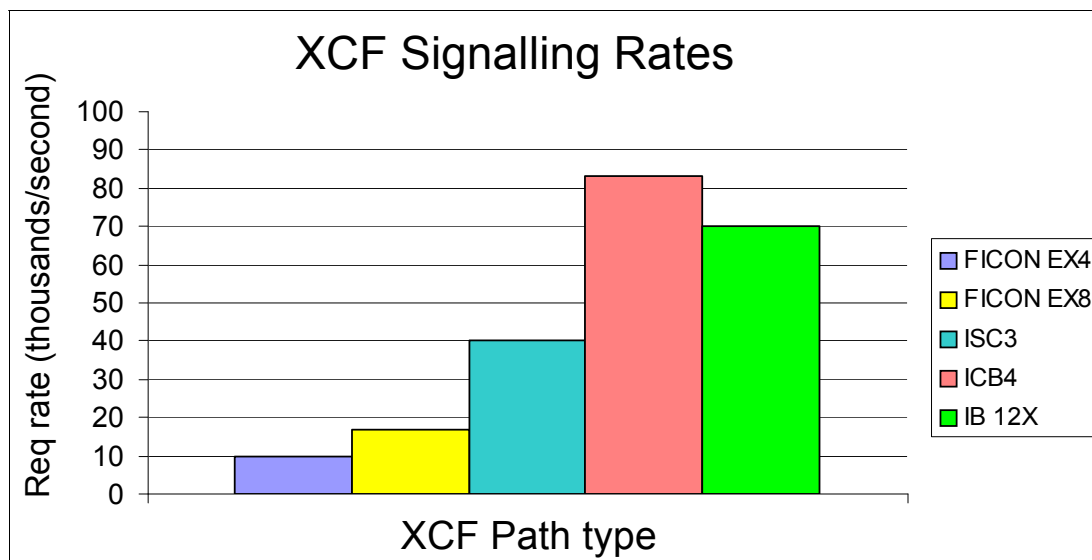


Figure 2-6 XCF signalling rates for various path types

If you have a multi-site sysplex, you *might* consider using CTCs, in addition to signalling structures, for XCF signalling. The only reason for this is that you want to do all you can to ensure that no system or set of systems loses all XCF connectivity to the other members of the sysplex. If the sysplex connectivity runs through a public area, there is an increased risk of damage to one of the paths. The most important thing you should do to protect against this risk is to ensure that there are two *completely* failure-isolated paths between the two sites. However, having both CTC and signalling structure connectivity might add a little more resilience in case you have an unintended single point of failure in the cross-site CF connectivity.

Single points of failure

Every transport class should have at least two paths between every pair of systems in the sysplex, and those paths should have no single points of failure in common between them. If a path breaks or becomes unavailable, having a second path means that XCF can still deliver messages in that transport class. If a transport class has no paths between a pair of systems, messages can still be sent over paths belonging to another transport class. However, that involves additional overhead and can result in longer transfer times.

One of the health checks related to XCF signalling paths is XCF_SIG_PATH_SEPARATION. However, be aware that this check only checks whether there are two failure-isolated paths between each *pair of systems*. It does not check that every *transport class* has two failure-isolated paths. Definitely enable the check, but the check on its own does not ensure that you will not have single points of failure at the transport class level.

Health check: The XCF_SIG_PATH_SEPARATION health check raises an exception if it discovers a single point of failure in the XCF signalling paths between a pair of systems in the sysplex. This check should be enabled, but because it checks at the system level rather than the transport class level, it should not be viewed as being comprehensive.

If you use signalling structures for XCF communication, have one structure per transport class per CF, and the structures for a given transport class should each be placed in separate CFs. Furthermore, specify an EXCLLIST for each XCF structure, naming the other structures in that transport class. Example 2-12 shows an example.

Example 2-12 Sample CFRM definition for XCF signalling structures

```
STRUCTURE NAME(IXC_DEFAULT_1)
  INITSIZE(20480)
  SIZE(31744)
  PREFLIST(FACIL03,FACIL04)
  EXCLLIST(IXC_DEFAULT_2)
  ALLOWAUTOALT(YES)

STRUCTURE NAME(IXC_DEFAULT_2)
  INITSIZE(20480)
  SIZE(31744)
  PREFLIST(FACIL04,FACIL03)
  EXCLLIST(IXC_DEFAULT_1)
  ALLOWAUTOALT(YES)
```

You can see in this example that the two structures used by the DEFAULT transport class (IXC_DEFAULT_1 and IXC_DEFAULT_2) are placed in separate CFs, and that each specifies the other structure on its EXCLLIST statement. This tells XES that, if possible, these two structures must not reside in the same CF.

Full signalling connectivity

In addition to every transport class having at least two failure-isolated paths to the systems that it is connected to, it is also important that there be full signalling connectivity in the sysplex. That is, every system should have XCF connectivity to every other system in the sysplex.

Health check: There are two health checks related to full XCF signalling connectivity:

- ▶ If a system loses all XCF connectivity to the other members of the sysplex, the default action if SFM is *not* active is to prompt the operator with a WTOR (IXC426A). The preferred option is that SFM *is* active and the CONNFAIL keyword is either allowed to default to YES, or YES is specified. In this case, SFM automatically determines which systems to partition from the sysplex to restore full XCF signalling connectivity. The XCF_SFM_CONNFAIL health check verifies that the CONNFAIL keyword in your SFM policy is set to YES. GDPS users should see the GDPS documentation for CONNFAIL recommendations specific to GDPS and update the PARM statement for the check as appropriate.
- ▶ The XCF_SIG_CONNECTIVITY health check raises an exception if it discovers that there are not at least two working paths from this system to every other system in the sysplex. This check should be enabled, and any exceptions should be addressed as a matter of urgency. Additionally, if you have more than two paths between each pair of systems, adjust the PARM setting for this check in the HZSPRMxx member to specify the actual number that you have. Doing this means that the health check raises an exception if it finds less than the expected number of available paths.

XCF signalling path buffers

XCF uses buffers at both ends of its signalling paths to handle spikes in XCF message rates. The *maximum* amount of storage that can be obtained for these buffers is controlled by the MAXMSG parameter in the COUPLExx member. If all the buffers fill, this situation is reported in the RMF XCF Usage by System report in the REQ REJECT column.

XCF running out of buffers is a serious condition, particularly if it happens on the PATHOUT end of the signalling path. This condition should be avoided, and addressed if it occurs. There are a number of possible reasons for this happening:

- ▶ The buffer is too small.
- ▶ An address space that is the target for XCF messages is not collecting its messages, resulting in those messages tying up the buffers used to hold those messages.
- ▶ The target system, or the target address space, does not have enough CPU capacity to process the volume of XCF messages that are being sent to it.

The RMF XCF Usage by System report shows the number of times that XCF tried to send a message and discovered that all buffers were already in use (known as a request reject). Figure 2-7 shows a sample report. In the sample, you can see that the req reject values in both columns are 0. This is good.

X C F A C T I V I T Y															PAGE
z/OS V1R9			SYSTEM ID FK03				DATE 01/04/2010				INTERVAL 10.00.000				
			CONVERTED TO z/OS V1R10 RMF				TIME 22.40.00				CYCLE 1.000 SECONDS				
															XCF USAGE BY SYSTEM

REMOTE SYSTEMS													LOCAL		

OUTBOUND FROM FK03													INBOUND TO SS03		FK03

		----- BUFFER -----				ALL									
TO	TRANSPORT	BUFFER	REQ	%	%	%	%	PATHS	REQ	FROM	REQ	REQ	TRANSPORT	REQ	
SYSTEM	CLASS	LENGTH	OUT	SML	FIT	BIG	OVW	UNAVAIL	REJECT	SYSTEM	IN	REJECT	CLASS	REJECT	
0FK01	DEFAULT	956	69,547	0	100	0	0	0	0	FK01	97,503	0			
	DEF04K	4,028	666	0	100	0	0	0	0						
	DEF08K	8,124	250	0	100	0	0	0	0						
	DEF16K	16,316	7	86	14	0	0	0	0						
	DEF32K	32,700	68	71	29	0	0	0	0						
	DEF64K	62,464	3	100	0	0	0	0	0						
FK02	DEFAULT	956	9,264	0	100	0	0	0	0	FK02	72,886	0			
	DEF04K	4,028	624	0	100	0	0	0	0						
	DEF08K	8,124	103	0	100	0	0	0	0						
	DEF16K	16,316	7,091	0	100	0	0	0	0						
	DEF32K	32,700	131	68	32	0	0	0	0						
	DEF64K	62,464	380	53	47	0	0	0	0						

Figure 2-7 RMF XCF Activity report

It is important to remember that the XCF buffer pool is shared between all address spaces whose messages are sent in the associated transport class. If an address space with a low (or discretionary) WLM importance in a CPU-constrained system receives a very large number of messages, it is possible for the XCF buffers to fill with those messages, impacting completely unrelated address spaces that happen to be using the same transport class (and therefore, the same XCF buffers).

For this reason, track the heaviest users of XCF services and ensure that those address spaces have sufficiently high WLM importance to ensure that they are provided with enough CPU to process their XCF messages in a timely manner.

If you encounter any non-zero request reject counts, and lack of CPU does not appear to be the cause, ensure that the MAXMSG values are at least as large as the default value (which, at the time of writing, is 2000 KB). On large or very busy systems, it is possible that a larger value might be appropriate.

Health check: There are two health checks related to the size of the XCF path buffers:

- The XCF_MAXMSG_NUMBUF_RATIO health check verifies that the current MAXMSG size is large enough to hold at least 30 messages, based on the CLASSLEN for that transport class. Remember that MAXMSG specifies an amount of storage, *not* a number of messages. So, for example, a MAXMSG value of 2000 holds many more 1 KB messages than 50 KB messages.
- The other health check is XCF_DEFAULT_MAXMSG. This check verifies that every XCF path has a MAXMSG value that is at least as large as the default value (which, at the time of writing, is 2000 KB).

If either of these checks raise an exception, adjust the appropriate MAXMSG value. The value can be adjusted dynamically using the SETXCF command, and on a permanent basis by updating the COUPLExx member.

For more information about XCF performance, see the presentation titled “WP100743 Parallel Sysplex Performance: XCF Performance Considerations V3.1,” available on the Techdocs website at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100743>

XCF signalling structure sizing

The XCF signalling structures in the CF are a temporary holding place as messages are sent from one system to another. Generally, messages only reside in the structure for a short time until they are retrieved by the target system.

For this reason, the size of the signalling structures does not need to be finely tuned. However, it is important that the structure is not undersized. There are two easy ways to determine the appropriate size of the structures:

- Use the CFSizer, available on the web at:

<http://www.ibm.com/systems/support/z/cfsizer/>

- Enable and monitor the XCF_SIG_STR_SIZE health check. This check uses the same logic as the CFSizer tool. However, it has the advantage that it runs on your system and automatically uses the correct values for your configuration.

It is common to see that the structure used for small messages (less than 1 KB) is appropriately sized, but that the structures used for larger messages are undersized. To ensure good performance, size all XCF structures appropriately for the CLASSLEN of the associated transport class.

Health check: The XCF_SIG_STR_SIZE health check should be enabled on your system, and any exceptions that it raises should be addressed.

Monitor XCF message transfer time trends

Just before XCF sends a message on a signalling path, it places a timestamp in the message. When the message is received by XCF on the target system, that timestamp is used to calculate the transfer time, and the rolling average transfer time for that path is updated. This transfer time information (in milliseconds) is provided by RMF in the RMF XCF path statistics report, in the column titled TRANSFER TIME (Figure 2-8).

XCF ACTIVITY														PAGE 31
z/OS V1R9		SYSTEM ID FK03		DATE 01/04/2010				INTERVAL 10.00.000						
		CONVERTED TO z/OS V1R10 RMF		TIME 22.40.00				CYCLE 1.000 SECONDS						
TOTAL SAMPLES = 600		XCF PATH STATISTICS												

0														
OUTBOUND FROM P003										INBOUND TO P003				

T FROM/TO								T FROM/TO						
TO	Y DEVICE, OR	TRANSPORT	REQ	AVG Q				FROM	Y DEVICE, OR	REQ	BUFFERS	TRANSFER	TIME	
SYSTEM	P STRUCTURE	CLASS	OUT	LNTH	AVAIL	BUSY	RETRY	SYSTEM	P STRUCTURE	IN	UNAVAIL	TIME		
FK04	S IXC_DEF01K1	DEFAULT	18	0.00	18	0	0	PROD	S IXC3_DEF01K1	13,375	0	0.226		
	S IXC_DEF01K2	DEFAULT	1,893	0.00	1,893	0	0		S IXC3_DEF01K2	5,381	0	0.233		
	S IXC_DEF01K3	DEFAULT	1	0.00	1	0	0		S IXC3_DEF01K3	11,611	0	0.235		
	S IXC_DEF04K1	DEF04K	332	0.00	328	4	0		S IXC3_DEF04K	112	0	0.258		
	S IXC_DEF04K2	DEF08K	7	0.00	7	0	0		S IXC3_DEF08K	65	0	0.284		
	S IXC_DEF16K1	DEF16K	1	0.00	1	0	0		S IXC3_DEF16K	418	0	0.414		
	S IXC_DEF16K2	DEF32K	27	0.00	27	0	0		S IXC3_DEF32K	12	0	0.593		
	S IXC_DEF64K1	DEF64K	13	0.00	13	0	0		S IXC3_DEF64K	17	0	0.958		

Figure 2-8 RMF XCF path statistics report

The transfer time naturally fluctuates up or down by hundreds of microseconds from one interval to the next. It also varies from one sysplex to another. However, for a given pair of systems, the transfer time is expected to stay within a certain range.

Monitor the transfer time on a long-term basis. If you find that the trend is steadily increasing, or that there is a significant increase (thousands of microseconds) for a number of intervals, investigate the cause. The most likely cause is a delay affecting the target system that delays XCF's ability to process the incoming message. A common example is a system with a very low PR/SM™ weight running on a CPC that is 100% utilized. The RMF Spreadsheet Reporter provides an easy way to view the transfer times for the entire sysplex (Figure 2-9). As you can see, the chart makes it very easy to identify intervals during which the transfer time varies significantly.

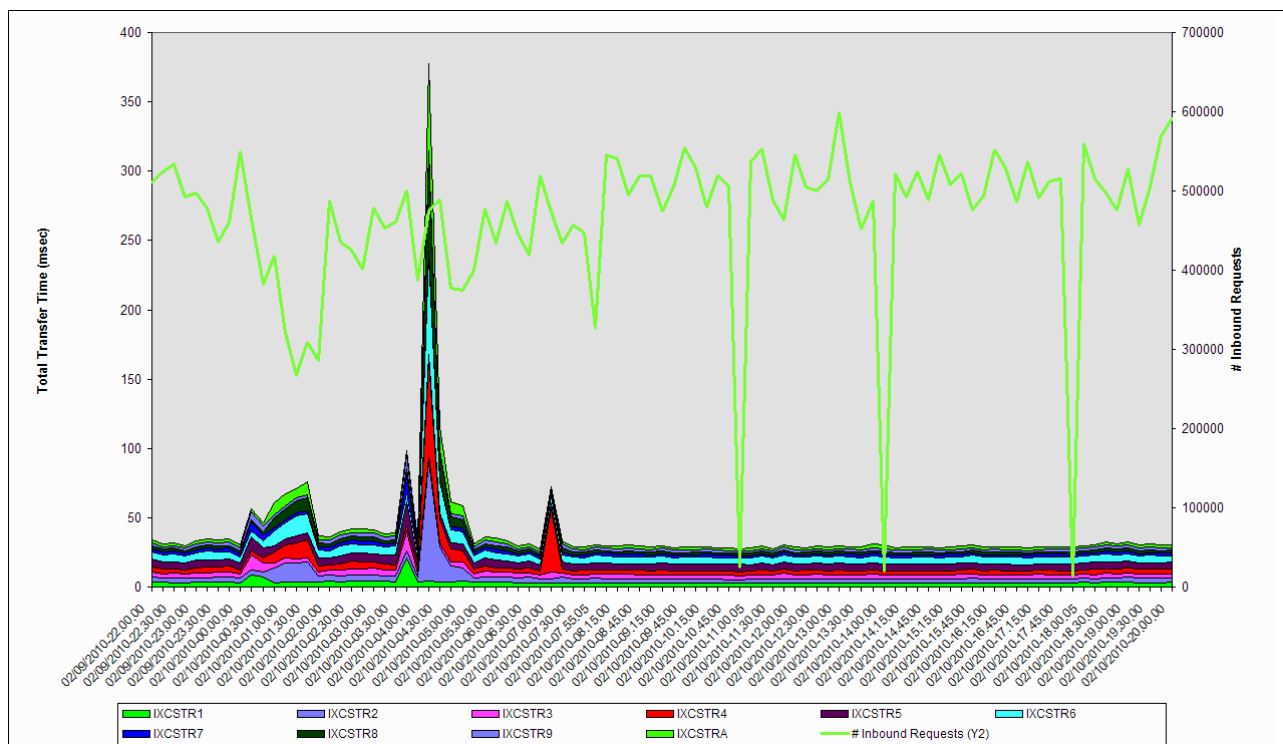


Figure 2-9 Using RMF to detect message transfer time exceptions

Suggestion: The RMF Spreadsheet Reporter (available from the RMF home page at <http://www.ibm.com/systems/z/os/zos/features/rmf/> or by downloading from the SYS1.SERBPWS data set on your z/OS system) greatly simplifies the task of checking many of the metrics and thresholds mentioned in this section. We encourage you to download it and get familiar with its use.

Monitor top XCF users

Although the pattern of XCF usage varies from one installation to another (based on their software and workload mix), within a given sysplex, the largest users of XCF, and the relationship between those users, tends to remain fairly stable. Therefore, monitoring the usage of XCF can provide a valuable early warning of software bugs or changing workload profiles.

Additionally, remember that all users of XCF services share a set of resources (path and their associated buffers). If one user is not collecting its messages in a timely manner, this can result in buffer shortages. This can particularly be the case if the user in question sends a large number of messages over XCF *and* it has a low WLM importance in a highly utilized system. For this reason, we suggest that you monitor the largest users of XCF services and

ensure that they are assigned to a WLM service class that allows them to get enough CPU resource to retrieve their messages from XCF in a timely manner.

More information about the problem of XCF users holding large numbers of XCF message buffers is contained in “MEMSTALLTIME” on page 37.

2.2.3 Transport classes

One of XCF's main roles is to provide a mechanism to transport messages between peer address spaces. These messages are sent between systems using XCF signalling paths. Transport classes are a way of grouping XCF signalling resources and assigning XCF messages to a given set of resources.

When an application sends a message to XCF for delivery to another member of the same XCF group, XCF assigns the message to the most appropriate transport class based on the message size, the XCF group that issued the message, or both. In most cases, the size of the message should be used to determine the best transport class because this results in the most efficient use of the available buffers¹.

Number of transport classes

For the majority of customers, three or four transport classes is sufficient. In most sysplexes, over 90% of XCF messages tend to be less than 1 KB, so you should always have a transport class with a CLASSLEN of 956 (XCF takes 68 bytes for his own use).

The distribution of messages larger than 956 bytes varies from one customer to another, and depends on your mix of software products and the application design and behavior. Many customers define a transport class large enough to carry the largest possible XCF message (CLASSLEN of 62464).

For the remaining transport classes, a typical CLASSLEN value is 8124, 12220, or 20412. To find the distribution of message sizes for your configuration, issue a **D XCF,CD,CLASS=transport_class_name** command for each transport class on each system in your sysplex during a busy period. You will see a response similar to that shown in Example 2-13. Note that the SEND CNT values for each buffer length wrap when they reach 99,999,999, and that each counter wraps independently of the others. Therefore, to get an accurate picture of the number of messages of each size, issue the same command 15 minutes later and calculate the delta between the two send counts for each BUFFLEN.

Example 2-13 Output from D XCF,CD command

```
D XCF,CD,CLASS=BIG
IXC344I 18.48.38 DISPLAY XCF 608
      TRANSPORT      CLASS      DEFAULT      ASSIGNED
      CLASS          LENGTH      MAXMSG      GROUPS
      BIG            12220        6144      UNDESIG

BIG TRANSPORT CLASS USAGE FOR SYSTEM #0$A
SUM MAXMSG:      18432      IN USE:      196      NOBUFF:      0
  SEND CNT:      16928      BUFFLEN (SML): 4028
  SEND CNT:      211173     BUFFLEN (SML): 8124
  SEND CNT:        4995     BUFFLEN (FIT): 12220
  SEND CNT:        384     BUFFLEN (BIG): 16316
  SEND CNT:         18     BUFFLEN (BIG): 20412
```

¹ One exception is GDPS customers, because IBM recommends that two transport classes be defined solely for GDPS use.

SEND CNT:	12	BUFFLEN (BIG): 24508
SEND CNT:	3	BUFFLEN (BIG): 28604
SEND CNT:	5	BUFFLEN (BIG): 32700
SEND CNT:	4	BUFFLEN (BIG): 36796
SEND CNT:	13	BUFFLEN (BIG): 40892
SEND CNT:	26	BUFFLEN (BIG): 44988
SEND CNT:	3	BUFFLEN (BIG): 61372
SEND CNT:	951	BUFFLEN (BIG): 62464

In Example 2-13 on page 32, the BIG transport class is defined with a CLASSLEN of 12220. If you find that the largest number of messages would have fit in a 8124 byte buffer, then you might want to change the CLASSLEN value for the BIG transport class. As a general rule, it is only worthwhile setting up a new transport class if it will carry about 5% of the total number of XCF messages.

Health check: The XCF_TCLASS_CLASSLEN health check verifies that at least a certain number of transport classes (with different CLASSLEN values) are defined. The number of classes that it checks for can be overridden in the HZSPRMxx member. The default value is 2, but we suggest a higher value, such as 3 or 4. This check should be enabled and any exceptions addressed.

There is a second health check that is partially related to the number of transport classes that you define. This check is called XCF_TCLASS_HAS_UNDESIG. This check verifies that every XCF group can potentially use any transport classes. This is normally a good idea. However, GDPS customers should see the GDPS product documentation for recommendations specific to GDPS.

Not more than one transport class per CLASSLEN

There is no benefit in having multiple transport classes handling the same set of XCF messages. Remember that for every message that is passed to XCF, it has to look at all the defined transport classes to find the best fit. If you define identical transport classes, you are increasing the number of transport classes that XCF has to check the message against (thereby driving up the XCF CPU cost), but not getting any benefit in return. If you feel that you need more signalling bandwidth, simply add another signalling path to the existing transport class.

Sometimes we see a sysplex that has two transport classes for small messages. One transport class might be called something like DEFSMALL, and the other DEFAULT. Often in this situation, the customer has defined a transport class called DEFSMALL, and did not explicitly define DEFAULT. However, it is important to remember that the system *always* defines a transport class called DEFAULT, and, unless you override it, that transport class has a CLASSLEN of 956. So, if you define your own transport class for small messages, and then do not assign any paths to the DEFAULT transport class, it is possible that XCF sometimes selects the DEFAULT transport class to send a message, and only discovers at that point that there are no paths defined for that transport class. It then has the overhead of having to find another transport class to send that message in.

Transport class connectivity

To have full XCF signalling connectivity within the sysplex, every system in the sysplex must have XCF connectivity to every other system. Because XCF signals are routed based on transport classes, you therefore need to ensure that all the transport classes that you define have paths to every other system in the sysplex.

If you use XCF signalling structures in the CF for all your XCF connectivity, then ensuring full connectivity is generally not an issue. However, if you use CTCs for XCF connectivity (which are point-to-point), then there is the possibility that you might forget to define a set of paths between one of more members of the sysplex. To help you identify and address such situations, enable and monitor the XCF_TCLASS_CONNECTIVITY health check.

Health check: The XCF_TCLASS_CONNECTIVITY health check verifies that every transport class has at least *x* paths to every other member of the sysplex, with *x* being specified on the PARM keyword in the HZSPRMxx member.

The default value for *x* is 1, but we suggest increasing this to the number of paths that you have provided for each transport class between each pair of systems. For example, if each transport class has two CF signalling structures and two PATHOUTs over CTCs, then specify a value of 4. You want the check to warn you if any of the paths that you configured becomes unavailable for whatever reason.

If this check generates an exception, address it as soon as possible.

2.2.4 Failure handling

While z/OS is legendary for its resilience and availability, like any software, it is possible that it can be impacted by problems. Hopefully, you are exploiting data sharing and dynamic workload balancing, so that even if one system has a problem, your critical applications can remain available on other members of the sysplex. Nevertheless, one of the best ways to stop a problem from spreading is to address that problem as quickly as possible. In z/OS, there are mechanisms to automatically address failures or problems within the sysplex. However, in some cases, it is the installation's responsibility to enable these functions.

System Status Detection Partitioning Protocol

z/OS 1.11 introduced a new capability called System Status Detection Partitioning Protocol whereby a system in the sysplex can use the z/OS BCPii interface to query the status of a sysplex member that has stopped updating its heartbeat in the Sysplex CDS. If it determines that the failing member is in a state that it will not be able to resume processing (for example, if it is in a non-restartable wait state, or if the LPAR has been system reset, or if it has been IPLed), the detecting member automatically (and immediately) partitions the failed system from the sysplex.

This is an important new capability. Prior to this, the only way that sysplex members had of determining the status of each other was by checking their heartbeat in the Sysplex CDS. Because there are valid situations in which a system might not update its heartbeat for a relatively long time, it was not reasonable to take action as soon as the missing heartbeat was detected. However, the ability to check the status of an LPAR means that partitioning actions can be initiated much sooner when it is determined that the LPAR in question is not able to resume processing.

The capability requires that:

- ▶ At least two members of the sysplex are running z/OS 1.11.
- ▶ The BCPii component has been set up and customized.
- ▶ The Sysplex CDS has been formatted with the ITEM NAME(SYSSTATDET) keyword.
- ▶ The system is running on a z10™ BC, or a z10 EC at the GA2 or later level, or on a later CPC.

z/OS 1.11 includes a health check called XCF_SYSSTATDET_PARTITIONING. This check understands the prerequisite hardware and software requirements and raises an exception if it determines that the environment supports System Status Detection Partitioning Protocol, but that the function is not enabled.

Health check: The XCF_SYSSTATDET_PARTITIONING health check lets you know if the system is executing in an environment that supports this new protocol, but that it has not been enabled.

To get more information about System Status Detection Partitioning Protocol, see the section titled “Using the System Status Detection Partitioning Protocol and BCPii for Availability and Recovery” in the z/OS 1.11 or later level of *z/OS MVS Setting Up a Sysplex*, SA22-7625.

Changed default partitioning action in z/OS 1.11

Another enhancement in z/OS 1.11 is a fundamental change to how z/OS reacts when it detects that a member of the sysplex might have failed. Prior to this capability, the default action (if SFM was not active) was to issue a WTOR to the operator. It was then the operator’s responsibility to take the appropriate action.

In z/OS 1.11, if there is no SFM policy, the default action when a system exceeds its failure detection interval is to partition that system from the sysplex. This is similar to how a system behaves if an SFM policy is active with ISOLATETIME specified. Nevertheless, it is still a good idea to have an active SFM policy with ISOLATETIME specified, regardless of your release of z/OS.

Failure detection interval

As discussed in “System Status Detection Partitioning Protocol” on page 34, there are situations in which a system can run for a relatively long time without updating its heartbeat in the Sysplex CDS. In such cases, the installation must provide a threshold that tells z/OS at what point to consider that the system has failed. This value, known as the failure detection interval (FDI), is specified on the INTERVAL keyword in the COUPLExx member, or via the SETXCF command.

The correct value depends on whether the system is using shared or dedicated CPs, and what spin loop recovery actions are being used. Prior to z/OS, if no INTERVAL value was provided by the installation, the system defaulted to 85 seconds for a system using shared CPs, and 25 seconds for one with dedicated CPs. The spin loop recovery actions that were in use were not used by the system in determining what FDI values to use.

In z/OS 1.11, the algorithm to determine the FDI was enhanced to take the spin loop recovery actions and the spin time value into account. In addition, the default FDI values (assuming that the default SPINTIME and SPINRCVY values are used) were changed to 165 seconds when using shared CPs, or 45 seconds when using dedicated CPs.

Best practice: z/OS 1.11 increased the default FDI values. These increased values increase the likelihood that recovery from spin loop situations will complete successfully before any sysplex partitioning actions are initiated. If you are running a release of z/OS prior to 1.11 and experience spin loop recovery situations (reported by message IEE178I), consider increasing your FDI to the z/OS 1.11 times of 165 seconds (shared CPs) or 45 seconds (dedicated CPs) by updating the INTERVAL value in your COUPLExx member.

Health check: z/OS 1.4 introduced a health check called XCF_FDI. This check compares the current FDI value to a computed value. As provided by IBM, the check uses a PARM value of MULT(2),INC(5). We advise enabling this check, but overriding the IBM-provided values so that the MULT value represents the number of spin recovery actions (other than PROMPT or ACR) specified in the EXSPATxx member, plus two. For example, if the EXSPAT member contains SPINRCVY=(ABEND,TERM,ACR), then we suggest setting MULT to 4.

Cleanup interval

When you issue a **V XCF,sysname,OFFLINE** command, a signal is sent to the target system telling it to wait-state itself after the number of seconds specified on the **CLEANUP** keyword in the COUPLExx member. When the target system receives this signal, it notifies all programs on that system that are connected to XCF groups that the system is about to shut down. **CLEANUP** seconds later, the system wait states itself.

In most cases, when you issue a **V XCF,sysname,OFFLINE** command, it is as part of a planned shutdown of the target system, and all work running on that system has already been quiesced before the **V XCF** command is issued. Therefore, there is little to be gained from having a too-large **CLEANUP** value. In fact, having a large **CLEANUP** value has little effect other than elongating the amount of time that it takes to shut down the system.

The recommended **CLEANUP** value is 15 seconds. Early releases of MVS and OS/390® had larger **CLEANUP** values. However, faster processors mean that those larger values are no longer necessary. The **XCF_CLEANUP_VALUE** health check verifies that you are using the currently recommended value of 15 seconds. We suggest that this check be enabled and any exceptions it raises are addressed.

To determine the current **CLEANUP** value, issue a **D XCF,C** command (Figure 2-10). If you want to change the **CLEANUP** value dynamically, you can issue a **SETXCF COUPLE,CLEANUP=xx** command, but remember to update your COUPLExx member to make the change permanent.

```
D XCF,C
IXC357I 15.39.14 DISPLAY XCF 131
SYSTEM #@$3 DATA
```

INTERVAL	OPNOTIFY	MAXMSG	CLEANUP	RETRY	CLASSLEN
165	165	2000	15	10	956

SSUM ACTION	SSUM INTERVAL	SSUM LIMIT	WEIGHT	MEMSTALLTIME
ISOLATE	0	720	70	NO

Figure 2-10 Displaying current **CLEANUP** value

Health check: The XCF_CLEANUP_VALUE health check should be enabled and any exceptions it raises should be addressed by updating the COUPLExx member or issuing a SETXCF command to change this value dynamically, or both.

Sysplex failure management

z/OS includes a function called sysplex failure management. Even though the use of this component is optional, it provides the ability for the system to take automatic action in response to certain sysplex-related problems, so its use is highly recommended.

Health check: The XCF_SFM_ACTIVE health check only checks that there is an active SFM policy. It does *not* check the specifics of the various policy options. While this check should be active, and any exceptions investigated and addressed, also ensure that the other SFM-related health checks are enabled and monitored.

ISOLATETIME

Prior to z/OS 1.11, the only way to have the sysplex automatically initiate partitioning actions for a failed system was to define and start an SFM policy that included the ISOLATETIME keyword.

If you are running a release of z/OS prior to 1.11, use an SFM policy with ISOLATETIME(0) (which says to start partitioning actions as soon as the FDI and OPNOTIFY intervals have expired). Note that even if you are running z/OS 1.11 or a later release, we *still* recommend having an active SFM policy with ISOLATETIME(0) specified.

Health check: The XCF_SFM_SUM_ACTION health check verifies that ISOLATETIME(0) is specified in the SFM policy. It should be enabled, and any exceptions it raises should be addressed.

MEMSTALLTIME

As discussed in “XCF signalling path buffers” on page 28, many different exploiters of XCF signalling services typically share a single set of XCF buffers. As long as all the exploiters collect their messages in a timely manner, this sharing is not an issue. However, if one of the exploiters stops collecting its messages for any reason, the number of buffers used by messages destined for that exploiter is likely to increase over time, potentially to the point that other exploiters start to suffer due to a shortage of available buffers. If the buffers on the target system fill completely, messages start to queue on the sending system, consuming buffers on that system. This situation is known as *sympathy sickness*.

To address this situation, SFM provides a function whereby stalled programs that are causing sympathy sickness can be automatically cancelled. By default, this function, known as MEMSTALLTIME, is not turned on. However, because of the potentially serious impact of running out of buffers, we suggest that you enable this function.

When you enable MEMSTALLTIME, you must specify a number of seconds as the parameter. This determines the number of seconds that elapse between when the messages indicating a sympathy sickness problem (IXC440E, IXC631I, and IXC640E) are issued and when SFM takes corrective action. The number of seconds should reflect your ability to react to the messages. If you think that you will be made aware of the messages as soon as they are issued, and want to be given an opportunity to try to resolve the problem, then specify a value large enough to allow this. Conversely, if you are unlikely to become aware of the messages in a timely manner, then it is a good idea to allow SFM to take action as soon as possible, in which case you want to specify a small value.

One other thing you might consider is to add automation to give the offending address space a better WLM service class. If the address space is not collecting its messages because of a lack of CPU, giving it a better service class might help resolve the situation. The address space that is holding the largest number of XCF buffers is identified in the IXC631I message, so that can be used to drive your automation processing.

Stalled members not causing sympathy sickness

In addition to the situation in which a stalled XCF member can result in sympathy sickness on another member of the sysplex, you can also have a situation where an XCF member stalls, but the stall does not result in an impact to other XCF members.

While this is not as serious a situation at the system level, nevertheless it is a situation that should be brought to the attention of the operators and possibly the system programmers. When XCF informs an XCF group member that there is a message waiting to be collected, it starts a timer. If the message is not collected within four minutes, messages IXC431I and IXC430E are issued (Example 2-14). You should add these messages to your automation so that they are immediately brought to the attention of the responsible staff.

Example 2-14 Stalled XCF member messages

```
10:59:09.06 IXC431I GROUP B0000002 MEMBER M1 JOB MAINASID ASID 0023
           STALLED AT 02/06/2009 10:53:57.823698 ID: 0.2
           LAST MSGX: 02/06/2009 10:58:13.112304 12 STALLED 0 PENDINGQ
           LAST GRPX: 02/06/2009 10:53:53.922204 0 STALLED 0 PENDINGQ
11:00:17.23 *IXC430E SYSTEM SC04 HAS STALLED XCF GROUP MEMBERS
```

SSUMLIMIT

If z/OS detects a system that has not updated its heartbeat in more than the FDI number of seconds, and an SFM policy with ISOLATETIME is active, the system checks to see whether the apparently failed system is still sending XCF signals. If it is *not*, SFM attempts to partition the failed system out of the sysplex. However if it *is*, SFM issues an IXC426D message asking the operator to take the appropriate action.

If the operator does not reply in a timely manner, you now have a system that is still a member of the sysplex. However, because it is unable to update the Sysplex CDS, it is unable to provide status information about its XCF members to the other systems in the sysplex, and similarly it is probably unable to obtain information about the status of XCF members on the other systems in the sysplex. This is not a desirable situation.

To address these situations, SFM provides a function known as SSUMLIMIT. SSUMLIMIT issues a message to the operator (IXC446I) when it detects a system that is not updating its heartbeat but is still sending XCF signals. The message identifies the problem system and informs the operator that the system will be automatically partitioned from the sysplex in x seconds if the system does not resume updating its heartbeat in the Sysplex CDS. The value of x is obtained from the SSUMLIMIT keyword in the SFM policy.

The value that you specify for x varies from installation to installation. If your installation has strong system management processes and skilled staff, you might choose a value of x that gives the staff enough time to identify and try to rectify the underlying problem. Conversely, if it is unlikely that you will be able to rectify the problem in a reasonably short time, then you might choose to select a smaller value for x. There is little reason for waiting longer than it takes to shut down all the work on that system.

Health check: The XCF_SFM_SSUMLIMIT health check verifies that the SSUMLIMIT function is enabled. By default, the check expects SSUMLIMIT to be set to 900. However, if you have a valid reason for using another SSUMLIMIT, you can change the check to look for your value by updating the PARM value for this check in the HZSPRMxx member. APAR OW31824 changed the default value for the check from 60 seconds to 900 seconds.

CONNFAIL

By default, SFM handles the processing in case of an XCF connectivity failure. However, this can be turned off by specifying CONNFAIL(NO). We recommend that you do not do this. Either let CONNFAIL default, or specify CONNFAIL(YES).

Health check: The XCF_SFM_CONNFAIL health check verifies that the CONNFAIL function is set to YES (the default). If the check raises an exception, review why this function is turned off, and re-enable it unless there is a good reason for it being disabled. GDPS users should see the GDPS documentation for CONNFAIL recommendations specifically for GDPS.

System weights

System weights are a way for you to communicate the relative importance to your business of the various members of your sysplex to SFM. If a failure results in SFM having to decide to partition a member from the sysplex, it uses the weights of all systems in the sysplex to determine which system should be removed from the sysplex to ensure the health of the group of systems with the highest aggregate weight.

The only way to assign a weight to a system is via the SFM policy. It is quite common to see sysplexes where every member has the same weight. But, in reality, it is likely that certain systems are more important to your business than others, either because of their size, or the applications that run on that system, or the fact that they play a critical role (a GDPS Control system, for example). For this reason, we suggest that you take the time to read about how SFM uses the weights (in the section titled “Controlling System Availability and Recovery through the SFM Policy” in *z/OS MVS Setting Up a Sysplex*, SA22-7625), and then assign appropriate weights to all the systems in your sysplex.

2.3 Coupling Facilities

A key component in any Parallel Sysplex is the Coupling Facility (CF) infrastructure and performance. This section provides best practices for the CFs themselves, and also guidance for the structures that reside in the CFs.

2.3.1 CF-level recommendations

Before getting into the best practices for individual structures, we address the CF infrastructure. If the CFs are not configured for high performance and high availability, fine tuning individual structures does not deliver significant results.

Certain items listed here are configuration-related (how many CFs to have, how many paths, and so on). Others are performance-related. Obviously, for the performance-related items, you want to select a peak interval to study. But we suggest that you select both a peak interval in the online window and also in the batch window, as the profile of requests to the CF is likely to be quite different for the two workload types.

Have multiple CFRM policies

Your CFRM policy contains the definitions for your CF structures. When you want to make a change to the policy, you can either add and activate a new policy (that is, one with a different name from the currently active policy), or you can replace the existing policy with one with the same name, and activate that policy.

We believe that the best practice is to use a different name from the current policy when you need to make a change. The simple reason for this is that it allows you to easily back out the new policy if you discover that it contains an error. If you replace the policy with an updated one with the same name, the only way to back out the change is to update the policy source again to undo your change—a process that takes longer and requires more expertise than simply activating the previous policy again.

To reduce any confusion for the operations personnel that will implement the change, consider a process such as the following:

1. Assume that you are currently using CFRM policy CFRM01.
2. You want to make a change to the policy, perhaps to add a new structure, so create a policy called CFRM02.
3. You tell the operators to start the policy called CFRM02. If the change is successful, they leave that policy started. If there is any problem with it, they revert to CFRM01.
4. When you are content that everything is OK with policy CFRM02, replace CFRM01 with the same structure definitions and have the operators start CFRM01. At this point, the two policies are identical again, and you are once again running with CFRM01.

Using a methodology such as this ensures that the operators know that every time that they are to move forward to a new policy, they should start CFRM02. If they need to fall back to the previous one, they start CFRM01. This reduces the chances of any mistakes or misunderstandings, while at the same time providing the ability to easily back out a new policy if it is found to be in error.

How many CFs to have

The high-availability characteristics of Parallel Sysplex rely on the ability to non-disruptively move the structures from one CF to another, allowing a CF to be taken offline for service without impacting the systems using that CF. Therefore, it is critical that all Parallel Sysplexes have *at least* two CFs and that those CFs are accessible to every member of the sysplex.

Health check: The XCF_CF_SYSPLEX_CONNECTIVITY health check verifies that at least two CFs are defined in the CFRM policy, and that every system in the sysplex has access to those CFs. This check should be enabled, and you should immediately address any exceptions that it raises.

Certain customers, depending on their configuration, workload, availability requirements, or number of data centers, might decide to have more than two CFs. If you are one of those customers, adjust the HZSPRMxx settings for this check to increase the MINCFS value to match the number of CFs that you are using.

Failure-isolated CFs

Certain CF structures require failure-isolation from the systems that are connected to them. An example is DB2 or IMS lock structures. Other structures, such as the GRS lock structure, do *not* require failure-isolation. Information about the failure-isolation requirements of all IBM structures is provided in Appendix C of *System-Managed CF Structure Duplexing*, available on the web at:

<ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/zsw01975usen/ZSW01975USEN.PDF>

One way to achieve failure isolation is to place the structure in a processor that does not contain any systems that are connected to that structure. This can be a processor that runs nothing but CF LPARs, or it can be a processor that runs z/VM® and zLinux LPARs. It can even be a processor that only contains members from sysplexes other than the production one. The important thing is that it should not contain any address space that is connected to the structures that require failure isolation.

If you have one CF that is failure-isolated from the systems connected to it and one that is in an ICF, place the structures that require failure isolation in the failure-isolated CF. All remaining structures can be placed in either CF, remembering that the load should be as evenly balanced across the two CFs as possible.

The other way to address the single point of failure that exists if the structure resides in the same processor as a connected address space is to use System-Managed Duplexing to duplex the structures with a failure isolation requirement to a CF in another processor.

Best practice: The best practice for any data sharing Parallel Sysplex is that there is at least one failure-isolated CF for any structures that require failure isolation.

From a purely technical perspective, the use of a failure-isolated CF is preferable to using System-Managed Duplexing because of the performance impact of this type of duplexing. You also need to consider the z/OS CPU cost of using System-Managed Duplexing. More z/OS CPU is required to process a System-Managed Duplexed request than a simplex request. If the number of duplexed requests is low, the CPU cost would probably be far less than the extra cost of an external CF (compared to the cost of an ICF). However, as the number of duplexed requests increases, there might be a point where the z/OS CPU cost of duplexing becomes larger than the cost benefit of an ICF. Identifying the precise crossover point is a complex process and includes consideration of the software stack costs on each system using System-Managed Duplexed structures.

You also need to consider the distance between the CFs. Because of the interactions between the two CFs, aim for the best possible response time for the inter-CF requests. The farther apart the two CFs are, the higher those response times will be. For this reason, we generally suggest that you avoid the use of System-Managed Duplexing over large distances.

System-Managed Duplexing

In addition to structures that require failure isolation from their connectors (like the DB2 and IMS lock structures), there are other structures that cannot transparently recover from a CF failure unless they are duplexed. These structures are identified in the *System-Managed CF Structure Duplexing* white paper referred to previously. Examples of these structures are the MQSeries® admin and queue structures. MQ provides mechanisms to recover persistent messages in these structures. However, the MQ service is disrupted if a CF containing simplex MQ structures fails. Another example is the CICS temporary storage, CF data tables, and named counter server structures.

For these types of structures, the use of System-Managed Duplexing is required to enable the structure connector to continue operating across a CF failure.

The performance impact and CPU cost of System-Managed Duplexing is related to the number of duplexed requests and the distance between the CFs. If the structure that is being considered for duplexing has a low access rate, the cost might be deemed to be acceptable. If the access rate is high, the cost increases. *Low* and *high* are relative numbers, and tend to change as the speed of the underlying technology changes. At the time of writing, low is up to a few thousand requests a second, and high is request rates in the tens of thousand per second. Each installation needs to make its own determination about whether the improved resiliency that System-Managed Duplexing provides for these structures is worth the increased response times and higher z/OS and CF CPU costs. The *System-Managed CF Structure Duplexing* white paper can help you quantify these costs.

Note: System-Managed Duplexed structures in a production sysplex should *only* be defined in CFs with dedicated engines. While the use of shared engines is supported, the resulting response times are unlikely to be acceptable for a production environment.

Enable DUPLEXCF16

z/OS 1.10, together with CF Level 16, delivers an optional enhancement that can improve the response time for System-Managed Duplexed CF requests. By default, this enhancement is turned off. If you are going to be using System-Managed Duplexing and your CFs are running CF Level 16 or later, you might benefit from enabling this function. This function can be enabled by issuing the SETXCF FUNCTIONS,ENABLE=DUPLEXCF16 command on every system in your sysplex, or by modifying the FUNCTIONS keyword in your COUPLExx member to enable this function.

Use dedicated engines for production CFs

Whenever possible, the use of dedicated engines for production CFs is highly recommended. This is particularly true if the CF contains structures are response time-critical.

Considering that most production z/OS systems use shared engines, you might wonder why IBM recommends dedicated engines for production CFs. There are a number of reasons:

- Most production z/OS systems are configured with multiple engines. So, when one shared engine is taken away by PR/SM, other engines are still available to process interrupts and new work requests.

Conversely, most CFs are configured with a single engine, so when that engine is taken away, the CF is unable to process any more work until it receives the engine back again.

- Response times on z/OS systems are generally measured in milliseconds (DASD response times) or hundreds of milliseconds (transaction response times). If a system loses a shared engine, the fastest that engine will be given back to the LPAR, on average, is probably a few milliseconds—roughly the elapsed time for three DASD I/Os.

Conversely, CF response times are measured in microseconds. Five microseconds is considered a good lock response time on current hardware. So if a CF loses an engine and gets it back in 3 milliseconds, that is the equivalent of 600 lock requests. So you can see that the relative impact of shared engines is much higher on CFs than on z/OS systems.

- When z/OS issues a synchronous request to a CF, it spins on the CP, waiting for the response. If the CF takes 5 microseconds to respond, z/OS has burned 5 microseconds of CPU time. If the CF takes 20 microseconds, then z/OS burns 20 microseconds. At some

point, if the response time continues to increase, z/OS switches over and starts issuing the requests asynchronously.

Now consider what happens when z/OS sends requests to a CF with a single shared engine. If the CF is dispatched when the first request arrives, it answers within 5 microseconds, so z/OS has burned 5 microseconds of CPU time. But what happens if the CF loses its engine before z/OS sends the next request? If we assume that it loses the engine for 3 milliseconds (3000 microseconds), z/OS burns 3000 microseconds of CPU time waiting for a response, instead of 5 microseconds.

Furthermore, in response to the long response time, z/OS might decide to send subsequent requests to the CF asynchronously. An asynchronous CF request costs more z/OS CPU than a fast synchronous request, so z/OS is burning more CPU for each request than it would have if the requests were sent synchronously to a CF that had not lost its engine.

While the use of shared engines might seem more cost-effective than providing dedicated engines for a CF, the decision process should include the impact on the connected systems, both in terms of CF response times and increased z/OS CPU time.

Health check: z/OS 1.12 delivers a new health check called XCF_CF_PROCESSORS that verifies that the CFs in use by the sysplex are using dedicated engines (in line with IBM best practices). If it is determined that dedicated engines are not required or justified for your test or development plexes, you can specify an override, listing the CFs that are intended to use shared engines.

If you are unsure of whether your CFs are using dedicated or shared engines, use the D CF command (Example 2-15). Starting with CF Level 15, the number of shared and dedicated engines is shown in the output from this command.

Example 2-15 Displaying information about CF engine types

```
D CF
IXL150I  10.09.04  DISPLAY CF 769
COUPLING FACILITY 002097.IBM.02.00000001DE50
                  PARTITION: 1E  CPCID: 00
                  CONTROL UNIT ID: FFE1

NAMED FACIL04
COUPLING FACILITY SPACE UTILIZATION
  ALLOCATED SPACE          DUMP SPACE UTILIZATION
    STRUCTURES:      1402880 K      STRUCTURE DUMP TABLES:      0 K
    DUMP SPACE:      400000 K          TABLE COUNT:      0
    FREE SPACE:      6489472 K      FREE DUMP SPACE:      400000 K
    TOTAL SPACE:      8292352 K      TOTAL DUMP SPACE:      400000 K
                                MAX REQUESTED DUMP SPACE:      0 K
                                STORAGE INCREMENT SIZE:      1024 K

    VOLATILE:          YES
    CFLEVEL:          16
    CFCC RELEASE 16.00, SERVICE LEVEL 00.34
    BUILT ON 04/01/2009 AT 14:25:00
    COUPLING FACILITY HAS 0 SHARED AND 1 DEDICATED PROCESSORS
    DYNAMIC CF DISPATCHING: OFF
```

Coupling Facility using shared engines and DCFD

A CF's *native* mode of operation is either to be doing work, or spinning, looking for new work to do. This provides excellent response times, because the CF can start processing new requests as soon as they arrive (assuming that the CF is not already busy processing a prior

request). However, this means that if the CF is using shared engines, the Coupling Facility Control Code (CFCC) does not give up its engines until their PR/SM time slice expires and the engines are taken away by PR/SM. As a result, any LPARs sharing engines with that CF tend to have to wait a long time to get the engine.

To make the use of shared engines more acceptable (especially when used by non-production CFs), IBM introduced a feature known as Dynamic CF Dispatching (DCFD). When DCFD is turned ON for a CF, the CFCC gives its engine back to PR/SM if it has no work to do. This results in more acceptable performance for the LPARs sharing engines with that CF (because they do not have to wait as long for it to give up the engine).

However, what happens if a request is sent to the CF 1 microsecond after it gives up the engine? That request now has to wait some number of milliseconds for the CF to get the engine back before it can start processing that request.

So, DCFD tends to be good for the *other* CF LPARs sharing that engine, but not so good for the performance of the CF that has DCFD turned ON. For this reason, we recommend that DCFD be turned ON for non-production CFs that are sharing engines, and OFF for production CFs using shared engines.

Hopefully, this will never happen, but if you *do* end up with more than one production CF LPAR sharing a given engine, the most important of the production CFs should have DCFD turned OFF, and the other sharing CFs should have it turned ON.

RMF CF reporting

Information about the use of the CF is gathered by RMF Monitor III. So to be able to monitor CF performance, you must start Monitor III (using the F RMF,S III command).

Furthermore, to gather as much information as possible about the CF, ensure that the ERBRMF04 member (the one that controls RMF Monitor III) specifies CFDETAIL. Prior to z/OS 1.8, the default value was NOCFDETAIL, so check to be sure that you are actually using CFDETAIL. You can check this using the F RMF,D III command.

Exploit new CF functions

Just as new releases of z/OS deliver new function, similarly, new CF functionality is usually delivered in new CF Levels. Depending on your product mix and your environment, some new functions might be important to you, while others might not. Depending on the timing of the delivery of new functions, a given processor generation might support one or more CF levels. For example, z10 EC supports both CF level 15 and level 16. For information about the new functions provided by each CF level and which generations of processor support a given CF level, see:

<http://www.ibm.com/systems/z/advantages/psa/cftable.html>

To determine your current CF level, issue a D CF command (Example 2-16).

Example 2-16 Displaying CF level information

```
D CF
IXL150I 08.52.30 DISPLAY CF 749
COUPLING FACILITY 002097.IBM.02.00000001DE50
                    PARTITION: 1E CPCID: 00
                    CONTROL UNIT ID: FFE1

NAMED FACIL04
COUPLING FACILITY SPACE UTILIZATION
  ALLOCATED SPACE          DUMP SPACE UTILIZATION
    STRUCTURES:      1402880 K      STRUCTURE DUMP TABLES:      0 K
```

DUMP SPACE:	400000 K	TABLE COUNT:	0
FREE SPACE:	6489472 K	FREE DUMP SPACE:	400000 K
TOTAL SPACE:	8292352 K	TOTAL DUMP SPACE:	400000 K
		MAX REQUESTED DUMP SPACE:	0 K
VOLATILE:	YES	STORAGE INCREMENT SIZE:	1024 K
CFLEVEL:	16		
CFCC RELEASE 16.00, SERVICE LEVEL 02.25			
BUILT ON 04/08/2010 AT 11:48:00			
COUPLING FACILITY HAS 0 SHARED AND 1 DEDICATED PROCESSORS			
DYNAMIC CF DISPATCHING: OFF			

CF REQUEST TIME ORDERING: REQUIRED AND ENABLED

You will see that the response contains the CF level for the listed CF. You will also notice on the next line in the response that the CF service level is also provided. In this case, it is 02.25. New CF service levels are generally used to deliver fixes for known problems. You can get information on ResourceLink about any CFCC service levels that have not been installed on your CPC yet. Your IBM hardware service representative can also provide you with this information. Just as IBM recommends that you remain current with your software service levels, similarly, you should, in general, try to avoid running on very old CF Service Levels.

Pay particular attention to the impact of new CF levels on structure sizes. Every structure contains space that is put aside to contain control blocks that control that structure. The new function included in a new CF level can result in an increase in the size of the control blocks, resulting in less space being available in the structure for application use. The best way to address this situation is to use the process described in “CF maintenance” on page 53 and adjust the CFRM policy accordingly.

CF CPU utilization

Over the years, there has been much spirited discussion over the maximum acceptable CF CPU utilization. Should it be 50%? 40%? 37.42%? In fact, what is actually important is what the CF CPU utilization would be were one of the CFs to fail.

Like all servers, the response time delivered by a CF varies with its utilization. At some point, the knee of the curve will be reached, and response times will start degrading at a higher rate. The position of that knee, and how sharply the response time degrades, is partly a function of the number of engines in the CF. Broadly speaking, the more engines that the CF has, the higher the utilization where the knee is reached, and the more gradual the degradation after that.

Therefore, a safe utilization to aim for in the remaining CFs should one of the CFs fail is 70 - 75% if the CF has just one engine. If the CF has more engines, utilizations up to 80% might deliver acceptable response times.

In the days before structure duplexing, determining the combined CF CPU utilization was easy, especially for customers with just two CFs. You simply find the point where the combined utilization of the two CFs was at a peak. Figure 2-11 shows an example. In Figure 2-11 (created with the RMF Spreadsheet Reporter), the peak combined utilization of the three CFs was about 66%.

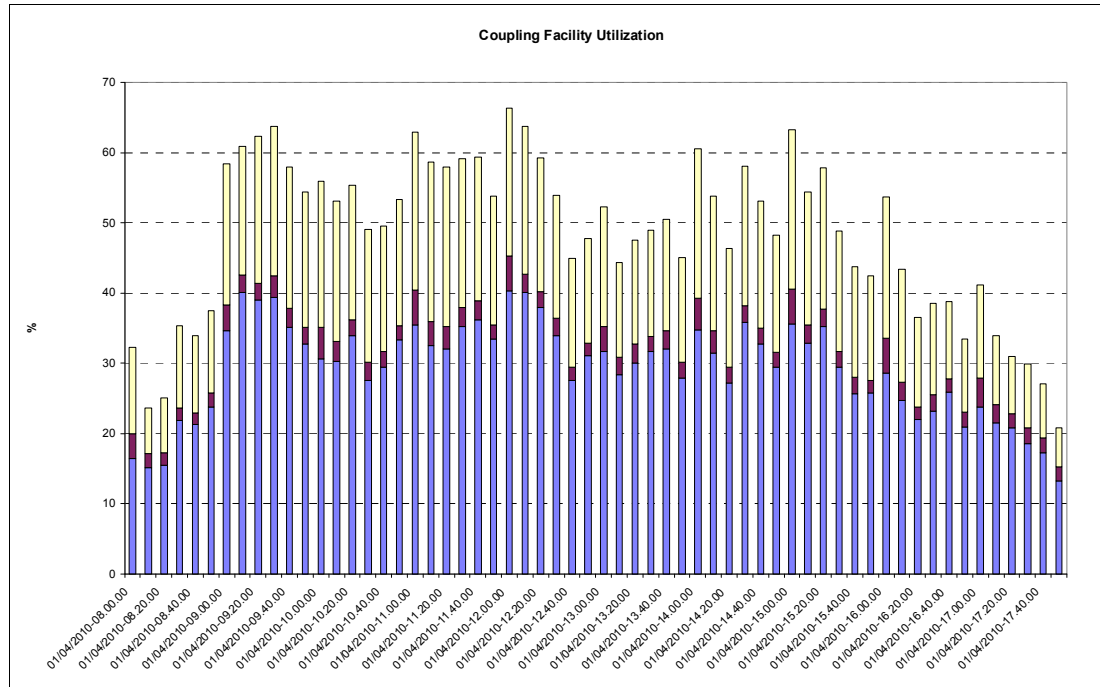


Figure 2-11 Plotting combined CF CPU utilization

However, with the increased use of structure duplexing, identifying the utilization of a surviving CF if the other CF fails becomes more complex. If you have just one CF, all the duplex structures revert to simplex. For DB2 Group Buffer Pool (GBP) structures that are using user-managed duplexing, the reduction in total CPU utilization is likely to be quite small.

However, structures that use System-Managed Duplexing are likely to see a larger decrease when they go simplex. Fortunately, the RMF Coupling Facility Usage Summary report helps you identify the amount of CF CPU being used by each structure. Figure 2-12 shows an example.

1	COUPLING FACILITY ACTIVITY											PAGE 1
z/OS V1R9			SYSPLEX PLEXFK00			START 03/02/2010-02.40.00			INTERVAL 000.10.00			
			RPT VERSION V1R9 RMF			END 03/02/2010-02.50.00			CYCLE 36.000 SECONDS			

COUPLING FACILITY NAME = CF01												
TOTAL SAMPLES (AVG) = 17 (MAX) = 17 (MIN) = 17												

COUPLING FACILITY USAGE SUMMARY												

STRUCTURE SUMMARY												

	STRUCTURE		ALLOC	% OF		% OF	% OF	AVG	LST/DIR	DATA	LOCK	DIR REC/
TYPE	NAME	STATUS CHG	SIZE	CF	£	ALL	CF	REQ/	ENTRIES	ELEMENTS	ENTRIES	DIR REC
				STOR	REQ	REQ	UTIL	SEC	TOT/CUR	TOT/CUR	TOT/CUR	XI'S
LIST	IXC_PATH1	ACTIVE	50M	0.2	53120	1.5	0.8	88.53	9773	9757	N/A	N/A
									1	17	N/A	N/A
	IXC_SMALL3	ACTIVE	11M	0.0	200039	5.5	4.3	333.40	628	611	N/A	N/A
									1	16	N/A	N/A
	IXC_SMALL5	ACTIVE	11M	0.0	103406	2.8	2.8	172.34	628	611	N/A	N/A
									1	22	N/A	N/A
LOCK	IXC_SMALL7	ACTIVE	11M	0.0	298192	8.2	5.4	496.99	628	611	N/A	N/A
									1	17	N/A	N/A
	DSNDBFK_LOCK1	ACTIVE	504M	1.5	358501	9.9	11.4	597.50	710K	0	134M	N/A
		PRIM							2170	0	0	N/A
	ISGLOCK	ACTIVE	65M	0.2	179786	4.9	0.8	299.64	0	0	8389K	N/A
									0	0	130K	N/A
	FKIM_IRLM	ACTIVE	650M	2.0	2113K	58.1	66.2	3521.9	2071K	0	34M	N/A
		SEC							196K	0	199K	N/A
...												
PROCESSOR SUMMARY												

COUPLING FACILITY			002097	MODEL E26		CFLEVEL 16		DYNDISP OFF				
AVERAGE CF UTILIZATION (% BUSY)			48.0	LOGICAL PROCESSORS:								
						DEFINED	1	EFFECTIVE	1.0			
						SHARED	0	AVG WEIGHT	0.0			

Figure 2-12 Sample RMF Coupling Facility Usage Summary report

First, sum up the CF CPU being used by System-Managed Duplexed structures. The total utilization of the CF is reported near the end of the usage summary report in the PROCESSOR SUMMARY section. In Figure 2-12, the utilization is 48.0%. Of that, the FKIM_IRLM structure accounts for 66.2% (so it is using 31.8% of the CF CPU), and the DSNDBFK_LOCK1 structure accounts for 11.4% (so it is using 5.5% of the total CF CPU). The amount of CPU used by the duplexed structures should be roughly the same in both CFs. To estimate the CPU usage if the structures were to go simplex, sum the utilization of the two CFs, then eliminate the usage associated with the secondary instance of the structure, and reduce the CPU time for the other instance by 50%.

CF storage utilization

Because CFCC does not use virtual storage, calculating the storage requirements for your CF structures is easy.

1. Ensure that CF storage utilization does not exceed 90%. When CF storage utilization reaches 90%, z/OS starts trimming storage from structures that are enabled for ALLOWAUTOALTER to ensure that storage is available if a new structure needs to be allocated.
2. Ensure that there is enough storage in each CF so that if one CF fails, all the structures in that CF can be successfully moved to another CF and the storage utilization of that CF is still less than 90%.
3. Adjust for any duplexed structures. If you have only two CFs, any duplexed structures go simplex, and the storage used by the secondary copies in normal operations can be discounted. If you have more than two CFs and have your PREFLISTs set up so that the structures re-duplex if one of the CFs goes away, then you include the storage for the secondary copies.

An easy methodology is to sum the storage usage of the CFs (this can be easily obtained using the D CF command (Figure 2-13)), then adjust that value to allow for the storage used by duplex structures (you can get this using the RMF structure summary report).

```
D CF
IXL150I 14.09.39 DISPLAY CF 362
COUPLING FACILITY 002097.IBM.02.00000001DE50
PARTITION: 1E CPCID: 00
CONTROL UNIT ID: FFE1

NAMED FACIL04
COUPLING FACILITY SPACE UTILIZATION
ALLOCATED SPACE          DUMP SPACE UTILIZATION
STRUCTURES:      1402880 K    STRUCTURE DUMP TABLES:      0 K
DUMP SPACE:      400000 K      TABLE COUNT:      0
FREE SPACE:      6489472 K    FREE DUMP SPACE:      400000 K
TOTAL SPACE:     8292352 K    TOTAL DUMP SPACE:     400000 K
MAX REQUESTED DUMP SPACE:      0 K
VOLATILE:         YES        STORAGE INCREMENT SIZE:    1024 K
CFLEVEL:         16
CFCC RELEASE 16.00, SERVICE LEVEL 02.21
BUILT ON 12/02/2009 AT 14:07:00
COUPLING FACILITY HAS 0 SHARED AND 1 DEDICATED PROCESSORS
DYNAMIC CF DISPATCHING: OFF
```

Figure 2-13 Displaying CF storage usage

Health check: z/OS 1.12 delivers a new health check called XCF_CF_MEMORY_UTILIZATION that verifies that the amount of used storage in your CFs does not exceed a certain percent. The default threshold is 60%, but you can override this value. Note that the check does not make any adjustment for duplexed structures.

Given that the CF starts reclaiming storage from structures when total storage utilization reaches 90%, you might expect the threshold for the check to be 45%. The reason that the threshold is greater than 45% is to allow for duplexed structures. If you do not duplex any structures, decrease the threshold for the check to 45% or even a little less. If the bulk of your CF storage is used by duplexed structures, you might be able to increase the threshold.

CF dump space

To make the process of capturing a dump of a CF structure as efficient as possible, the control blocks of the structures being dumped are written to an area within the CF, and then asynchronously moved to z/OS to be included in the dump.

The amount of space reserved for holding such dumps is specified on the DUMPSPACE parameter when you define the CF in the CFRM policy. The recommended value for DUMPSPACE is 5% of the total storage in the CF. So, if the CF contains 10 GB of storage, the DUMPSPACE value (which is specified in KB) should be 500000 (500 MB).

CF links

CF links are the CF's equivalent of the channels used by z/OS to communicate with the outside world. And just as the number and type of channels that you use has an impact on DASD response times and availability, the number and type of CF links used to connect z/OS to your CFs has a major impact on the response times and availability of the CF.

Number of links

Back in the early days of Parallel Sysplex, the guidance for the number of CF links to use was easy: one link per system was plenty for delivering good performance, but it was best to add a second link for availability. But the volume of requests that is being handled by CFs now is vastly more than was the case back in those early days. There are now customers processing more than 500,000 requests per second in each CF, and benchmarks in IBM have driven nearly 1,500,000 requests a second to a single CF. As a result, the number of links that are required can vary widely from one customer to another.

How do you determine the number of links required for *your* configuration? The first thing is that every connected processor should have at least two *failure-isolated* links to each connected CF.

Health check: The XCF_CF_CONNECTIVITY health check verifies that the system has two online, failure-isolated, links to each defined CF. This check should be enabled, and any exceptions should be investigated as a matter of urgency.

Note that this check is not aware of the relationship between CF link CHPIDs and the InfiniBand infrastructure. If you define two CHPIDs going to the same CF on the same InfiniBand card, the check is not aware of that. However, the CHPID Mapping Tool flags such a configuration as a potential single point of failure.

A possible indicator of the need for more CF link capacity is the number of path busy events. These are reported in the RMF CF Subchannel Activity report. The guideline is that the number of path busy events should not be more than 10% of the number of requests sent to

the CF. To be more specific, the summed path busy count for all the systems sharing a given set of CF links should not exceed 10% of the total number of requests to that CF from those systems. Note, however, that path busy events and XES subchannel tuning can only occur for links that are shared between z/OS LPARs, so we need another metric to know whether the link utilization is too high for CF links that are not shared.

The next guideline is that the subchannel utilization for CF links should not exceed 30%. To calculate the subchannel utilization, use the following formula:

$$\frac{(\text{Sync Resp Time} * \# \text{ Synch Requests}) + (\text{Asynch Resp Time} * \# \text{ Asynch Requests})}{\# \text{ Subchannels} * \text{Number of seconds in Interval} * 1,000,000}$$

All this information can be obtained from an RMF CF Subchannel Activity report (Figure 2-14). You can also find the subchannel utilization for each system in the RepCFSys sheet in the RMF Spreadsheet Reporter.

COUPLING FACILITY ACTIVITY															PAGE 88
z/OS VIR10			SYSPLEX FKLEX			DATE 01/04/2010			INTERVAL 010.00.000						
			CONVERTED TO z/OS VIR10 RMF			TIME 22.40.00			CYCLE 01.000 SECONDS						

COUPLING FACILITY NAME = CF03															

SUBCHANNEL ACTIVITY															

SYSTEM NAME	# REQ TOTAL AVG/SEC	-- CF TYPE	LINKS GEN	-- USE	PTH BUSY	REQUESTS			DELAYED REQUESTS			# REQ	% OF REQ	AVG TIME (MIC)	
						# REQ	-SERVICE TIME (MIC)- AVG	STD_DEV	# REQ	/DEL	STD_DEV			/ALL	
FK01	6183K	CFP	7	7	2466	SYNC	35557	59.2	48.8	LIST/CACHE	320	0.0	1637	737.5	0.3
	10306	SUBCH	49	49		ASYNCHANGED	6160K	89.1	137.7	LOCK	27	0.0	92.0	104.8	0.0
						UNSUCC	37	INCLUDED IN ASYNCHANGED		TOTAL	347	0.0			
							0	0.0	0.0						
FK02	251819	CFP	6	6	0	SYNC	20898	55.0	9.3	LIST/CACHE	0	0.0	0.0	0.0	0.0
	419.7	SUBCH	42	42		ASYNCHANGED	222334	411.3	445.6	LOCK	0	0.0	0.0	0.0	0.0
						UNSUCC	0	INCLUDED IN ASYNCHANGED		TOTAL	0	0.0			
							0	0.0	0.0						
FK03	3506K	CFP	5	4	130K	SYNC	81045	50.3	33.9	LIST/CACHE	959	0.1	843.6	1090	0.5
	5842.5	SUBCH	42	28		ASYNCHANGED	3407K	108.5	175.3	LOCK	323	0.0	7.7	14.5	0.0
						UNSUCC	937	INCLUDED IN ASYNCHANGED		TOTAL	1282	0.0			

Figure 2-14 RMF CF Subchannel Activity report

One final thing to watch for is whether the number of in-use subchannels (as reported in the RMF Subchannel Activity report) changes from one interval to another. When z/OS determines that the number of path busy events is unacceptably high, it might react by taking subchannels offline. This process is known as XES subchannel tuning. No message is produced on the console when this happens. However, you can spot this activity in the RMF reports. If you see this happening, that is an indication that more CF link capacity is required between that system and the CF. For more information about this situation, see the Technote titled *XES Coupling Facility Subchannel Tuning*, available on the web at:

<http://www.redbooks.ibm.com/abstracts/tips0176.html?Open>

Link types

Depending on the type of CF links that you use and the distance between the z/OS system and the CF, your CF requests might take more time traveling up and down the CF link than they spend getting processed in the CF.

Figure 2-15 on page 51 shows the results of a measurement that we ran comparing the performance of ISC and 12X InfiniBand links. You will see that the synchronous response time for lock requests was twice as long when using the ISC links. The synchronous response time for GBP requests with the ISC links was nearly four times longer than when using InfiniBand links. Furthermore, the percentage of GBP requests that were sent synchronously was *far* higher with the InfiniBand links. In this particular measurement, not only did the

InfiniBand links result in better response times, they also resulted in a reduction in the z/OS CPU utilization used to drive this particular workload from about 117% of a z10 engine down to about 107%. Note, however, that this workload was particularly CF-intensive, so the benefit that you are likely to see if you replace ISC links with InfiniBand links is likely to be less than this.

Important: This measurement is *not* a formal benchmark. It only reflects the results that we observed in our environment. It does not include any distance between the z/OS processor and the CF processor. Your results will differ depending on your workload type, how CF-intensive the workload is, CPU types, the types of links that you are using, distances, and other variables. Formal IBM benchmarks are conducted by a team of performance specialists using dedicated hardware.

<u>ISC</u>		<u>PSIFB</u>	
LOCK1 Total Activity Rate	15130.11	LOCK1 Total Activity Rate	16055.27
LOCK1 Sync Activity Rate	2082.71	LOCK1 Sync Activity Rate	15875.84
LOCK1 Sync ST	26.23	LOCK1 Sync ST	13.47
LOCK1 Async Activity Rate	13047.40	LOCK1 Async Activity Rate	179.43
LOCK1 Async ST	119.82	LOCK1 Async ST	84.84
GBP3 Total Activity Rate	11507.44	GBP3 Total Activity Rate	13419.46
GBP3 Sync Activity Rate	60.12	GBP3 Sync Activity Rate	10024.09
GBP3 Sync ST	63.44	GBP3 Sync ST	16.93
GBP3 Async Activity Rate	11447.32	GBP3 Async Activity Rate	3395.38
GBP3 Async ST	176.05	GBP3 Async ST	142.01

Figure 2-15 Comparison of InfiniBand 12X and ISC CF links

Apart from the performance of a particular link type, you also need to consider:

- ▶ The maximum distance supported by a given link type
- ▶ The types of processors that support a given link type
- ▶ IBM statements for future support of different link types

Table 2-3 shows the supported distances by link type. You can see that 1X InfiniBand links support similar distances to ISC links. However, in informal measurements that we conducted, they displayed significantly better performance characteristics.

Table 2-3 Supported distances by CF link type

Distance	ICP	ICB4	ISC	1x PSIFB	12x PSIFB
Within server	Yes	Yes	Yes	Yes	Yes
Up to 10 meters		Yes	Yes	Yes	Yes
Up to 150 meters			Yes	Yes	Yes
Less than 100 km			Yes	Yes	

The other thing to consider is which CF link types are supported by various processor generations. Table 2-4 on page 52 summarizes which processor generations support which

link types. Note that the z196 does not support ICB4 links. Any customers with ICB4 links should be formulating plans to migrate to InfiniBand link before, or as part of, the move to the next generation of processors.

Table 2-4 Supported link types by processor generation

Processor	ICP	ICB4	ISC	1X PSIFB	12x PSIFB
z890/z990	Yes	Yes	Yes	No	No
z9EC/z9BC	Yes	Yes	Yes	No	Yes ^a
z10EC/z10BC	Yes	Yes	Yes	Yes	Yes
z196	Yes	No	Yes	Yes	Yes

a. InfiniBand links are not supported for connecting two z9s to each other. InfiniBand support on z9® is for connection to z10 or later processors.

Recommendation: Given the performance advantage of InfiniBand links compared with ISC links, and the fact that z10 was the last generation to support ICB4 links, all Parallel Sysplex customers should start planning for migration to InfiniBand links.

There are also a number of recommendations relating to how your CF links are configured:

- ▶ Due to their high bandwidth and excellent performance, it should never be necessary to have more than four ICP links between a CF and a connected z/OS.
- ▶ The number of CHPIDs for ICP links should not exceed the number of CHPIDs associated with the links to an external CF. For example, if you have six CF links to an external CF, you should not have more than six ICP CHPIDs (three ICP links).
- ▶ All current CF links are designed to be used in peer mode (that is, they are capable of acting as both sender and receiver links). For example, if you have three ICP links connecting an ICF to a z/OS in the same CPC, you define six CHPIDs (two for each ICP link). The access list for each CHPID should contain both the CF LPAR and the z/OS LPARs that will connect to it.

CF volatility

It is possible to order a System z processor with an Internal Battery Feature. The battery backup maintains power to the processor for a short time so that the structure contents are not lost in case of a power failure. If the processor is configured with a battery backup, the CFCC senses this and automatically sets the status of the CF to NONVOLATILE.

However, if you are relying solely on a UPS for power during a power failure, the CF is not able to sense this and automatically sets its status to be VOLATILE. The setting of this status can affect the XES algorithms that decide where to allocate a structure.

Note: IBM 9672s had a feature called POWERSAVE, whereby the CF memory is preserved, but power to the rest of the processor is removed. Even though it is possible to order a System z processor with a battery backup, the POWERSAVE feature is no longer available.

If possible, use *both* the Internal Battery Feature and UPS for your CFs. This is because it is possible for either type of power source to fail, so by providing both battery backup *and* UPS, you improve your likelihood of your CF contents surviving a power failure.

Regardless of whether you use a battery backup, or a UPS, set up automation to check the CF volatility status after every POR. This can easily be done using the D CF command. If the status is incorrect, you can set it to the correct state using the MODE command on the CF console on the HMC.

Health check: The XCF_CF_STR_NONVOLATILE check verifies that any structure that has requested a non-volatile CF is actually residing in such a CF. If you are unable to provide a non-volatile CF, this check can be disabled. But if you *do* have a non-volatile CF and this check issues an exception, immediately investigate the cause of the exception.

You can find more information about the IBF feature in the *IBM System z10 Enterprise Class Technical Guide*, SG24-7516.

CF maintenance

There might be situations in which it is necessary to take a CF down for maintenance or an upgrade. In these cases, move all the structures out of that CF and into the other CFs. Different releases of MVS/ESA, OS/390, and z/OS have provided various ways of achieving this activity. The process that we suggest is as follows. We recommend it because we believe that it is the simplest way to achieve the desired result, allows the least opportunity for human error, and results in all structures being located in the desired CF at the end of the process.

1. Issue a D XCF,CF,CFNM=* command to show which structures are in each CF. It might be helpful to be able to come back later and see where all the structures were located before the maintenance activity started.
2. Issue a SETXCF START,MAINTMODE,CFNM=cf_name command for the CF that you want to empty. The only thing that this command does is set the status of the named CF in the CFRM policy to indicate that no new structures should be allocated in this CF. It has a similar effect to removing that CF from all the PREFLISTs in the CFRM policy. Note that this command does *not* move any structures, nor does it take the CF offline.
3. Having logically removed the CF from all the PREFLISTs, the next step is to issue a SETXCF START,REALLOCATE command. This command causes XCF to look at the PREFLIST for every allocated structure. If a structure is in a CF that is not in MAINTMODE, it will be left where it is. If it is in the CF that *is* in MAINTMODE, it will be rebuilt to the next CF in its PREFLIST. If the structure is currently duplexed, and either the primary or secondary instance of the structure is in the CF that is to be taken down, the structure is taken back to simplex, with the surviving instance being in the other CF in the PREFLIST.

At the completion of the SETXCF START,REALLOCATE command, the CF that is being stopped should be empty.

4. Log on to the HMC. Select the CF that is to be stopped and the *operating system messages* action. Note that this is different from how many installations stop a CF (by selecting the deactivate action). Using the CF console, issue the SHUTDOWN command. If there are no structures in the CF, the CF moves to a status of not operating. However, if you accidentally select the wrong CF (the one that now contains all your structures), the command responds that the shutdown has been cancelled because there are still structures in the CF. Using the SHUTDOWN command rather than the deactivate action prevents you from accidentally stopping the wrong CF.

If the shutdown is for the purpose of loading a new CFCC service level, it is not necessary to deactivate the CF LPAR². You simply need to select an activate action against the CF LPAR. If the hardware engineer indicates that it is necessary to deactivate the CF LPAR, you can select the deactivate action against the CF LPAR. However, because the CF

² In fact, in most cases, it should be possible to load a new CF service level without emptying or restarting the CF.

LPAR is already in a not operating state, there is less chance of accidentally deactivating the wrong CF.

At this point, the CF is down and the engineer can take whatever actions are required. When the engineer is finished, the CF can be brought back into use by basically reversing the process that we used to stop it.

1. To initialize the CFCC code, activate the CF LPAR on the HMC.
2. When the CF initialization completes, the CF reconnects to all connected z/OS systems. This kicks off a process whereby XCF attempts to re-duplex any structures that were in the simplex state, but defined as DUPLEX(ENABLED) in the CFRM policy. The benefit of this is that you do not need to remember to manually re-duplex all those structures. Conversely, re-duplexing structures can be disruptive, as all accesses to that structure are quiesced while the structure contents are copied.

However, because we placed the CF in MAINTMODE as part of the shutdown, this re-duplexing does *not* happen at this time.

Instead, at the time that you are ready to re-duplex (or to bring the CF back into use in general), issue a `SETXCF STOP,MAINTMODE,CFNM=cf_name` command. This makes the CF available for use again, but it does *not* initiate automatic re-duplexing of the structures, nor does it cause any structures to be moved into that CF.

3. Issue another `SETXCF START,REALLOCATE` command. This results in any simplex structures that have that CF as the first CF in their PREFLISTs moving back into that CF. Any duplex structures that reside in that CF are re-duplexed, with the primary instance of the structure being moved to that CF if that is what your PREFLIST indicates.
4. Issue another `D XCF,CF,CFNM=*` command to display the new location of every structure. It should match the locations at the start of the process.

This procedure requires that at least one system in the sysplex is running z/OS 1.9 or later (because the MAINTMODE capability was added in z/OS 1.9). Beyond that requirement, however, no other changes are required.

We advocate that you use this process to shut down and repopulate your CFs, as it is simple, efficient, and brings the structures back to the configuration that you designed (via the PREFLISTs).

Health check: The `XCF_CF_ALLOCATION_PERMITTED` health check verifies that MAINTMODE is turned OFF for all CFs. Enable this check, just in case you forget to turn MAINTMODE OFF after some maintenance activity. Any exceptions that the check raises during the period when you expect the CF to be down can be ignored.

2.3.2 Structure-level recommendations

If you follow the recommendations in 2.3.1, “CF-level recommendations” on page 39, you will have a CF infrastructure that is capable of delivering excellent response times and high availability. Now we look at guidelines relating to the structures that reside in the CFs.

Top CF users

For most sysplexes, the pattern of which CF structures are the busiest tends to be consistent, at least for a given shift. (The pattern of activity is more erratic and a less predictable during the batch window than the online day.) Take the time to become familiar with the normal pattern of activity. A sudden change in the balance of activity across the structures can be an indication of a configuration problem or a bug, as can a sudden change in CF CPU utilization.

In most installations, the heaviest users of the CF tend to be lock and cache structures for the database managers, IMS and MQ shared queue and their associated structures, GRS, and XCF. Become familiar with the heaviest users of your CFs and monitor for changes from the normal pattern.

Structure request types

Requests can be sent to a CF either synchronously or asynchronously. To understand CF response times, it is important to understand the difference between the two types.

For a synchronous request, XES sends the request to the CF and then *spins*, waiting for the CF request to complete. Thus, as soon as the CF completes the operation, z/OS sees that and completes the request.

For an asynchronous request, XES sends the request to the CF. But then instead of spinning, it transfers control to the system dispatcher, so that another work item can be processed while the request is being handled out in the CF. When the CF returns the response, the HW completion is denoted by posting a bit in HSA. At some point after this, z/OS tests the bit to discover the completion. This test happens during a trip through the z/OS dispatcher.

Thus, the time to complete an asynchronous request is greatly affected by the dispatch rate in the system. Furthermore, for the z/OS dispatcher to run, a logical engine from the LPAR must be dispatched by PR/SM to a physical processor. If there is little activity in the LPAR, or the LPAR has a low weight and is losing in competition to other LPARs with higher weights, an additional delay can occur while waiting for PR/SM to dispatch a logical engine to a physical engine. Thus, there are several points at which an asynchronous operation can be delayed:

- ▶ Waiting for the next z/OS dispatch
- ▶ Waiting for PR/SM to dispatch a logical engine

This makes it impossible to predict for a given configuration what asynchronous times to expect. It is dependent on the z/OS dispatcher rate and the effects of the total LPAR configuration on the PR/SM dispatcher.

Monitor average structure response times

What is a *good* CF response time? It depends on the structure type, the CF type, the link type, the type of requests being sent to the structures in the CF, and the distance between z/OS and the CF.

Determine whether your existing response times are within the expected range for your configuration. Table 2-5 contains ranges of expected response times for various types of sync CF requests, assuming that both the z/OS processor and the CF processor are z10s.

Table 2-5 Expected ranges of synchronous structure response times for z10

	Lock	List or cache
IC	3-8	6-10
ICB4	8-12	10-16
12x PSIFB	11-15	15-20
1x PSIFB	14-18	18-25
ISC	20-30	25-40

Because synchronous response times are affected by link type, request type, amount of data, host technology, distance between the CF and z/OS processors, CF technology, and CF

utilization, it is difficult to set a range that will cover all situations. The numbers in Table 2-5 on page 55 should cover the majority of configurations, however variances in the any of these factors could result in response times falling outside these ranges. Additionally, the measurements are based on zero distance between z/OS and the connected CF. Every 1 kilometer between the z/OS processor and the CF increases the response time by 10 mics.

For asynchronous requests from a typical busy high-weighted LPAR, response times for simplex operations typically fall in the 50 - 250 microsecond range. For other configurations, asynchronous response times can grow much higher due to the various dispatcher (both z/OS and PR/SM) delays.

After you establish the expected response times for your most important structures, monitor those structures to ensure that response times do not fall outside that range.

Monitor ratio of synchronous to asynchronous requests

The XES heuristic algorithm introduced in z/OS 1.2 determines whether a given CF request will be sent to the CF synchronously or asynchronously. The algorithm selects the most efficient way to send the request (from the perspective of minimizing the z/OS CPU time to drive the request) based on the expected response time for the request.

If the balance between synchronous and asynchronous requests changes over time, that is generally an indication that the response times are degrading, resulting in a higher percentage of requests being sent synchronously. If the ratio between synchronous and asynchronous requests is changing slowly, that is probably a result of workload increases. However, if there is a sudden change in the ratio, that might indicate a configuration problem (maybe a broken CF link) and should be investigated as a matter of urgency and the reason identified and addressed.

The RMF CF subchannel activity report provides counts of the total number of synchronous and asynchronous requests sent to each CF from each system. Extract these numbers and monitor the trends, both at the system level and specifically for your most important structures.

Spread structures and load across the available CFs

To optimize performance and availability, allocate your structures across the available CFs in a manner that results in similar CPU utilization in both CFs. Under normal circumstances, CFs should not be viewed as an active and a backup. You obtain the best performance by actively using both CFs.

In particular, spread the signalling structures for each XCF transport class over multiple CFs. If a CF fails, XCF communication is necessary to coordinate recovery. However, if you have all your XCF structures in the same CF, and that CF is the one that fails, recovery from the failure is severely impeded because XCF is no longer able to communicate via its signalling structures.

The RepCFTrd sheet in the RMF Spreadsheet Reporter provides a very easy way to check that utilization is balanced across the available CFs. Also, note that different types of CF requests use different amounts of CF CPU. For example, a lock request typically uses less CF CPU than a request to a list structure. For this reason, attempt to have a mixture of structure types in each CF.

Structures with no activity

Certain CF exploiters set up their structures so that the structures remain allocated even if the exploiter is stopped. An example of such a structure is a JES2 checkpoint structure. Therefore, in addition to monitoring for the busiest CF users, also watch for structures with

little or no activity. This can indicate a function that was accidentally or temporarily turned off and not re-enabled.

Structure sizes

There are a number of things to consider related to the size of your CF structures. The size of a given structure depends on:

- ▶ The structure type (cache, list, or lock)
- ▶ How the structure is being used
- ▶ The CF level of the Coupling Facility
- ▶ The maximum size of the structure as defined in the CFRM policy

The CFSizer tool (available on the web at <http://www-947.ibm.com/systems/support/z/cfsizer/>) can be used to estimate the size for new structures and can also be used to validate the size of your existing structures, particularly if you are planning to change the CF level of a CF or are preparing for a significant increase in workload.

Defining structure sizes

The initial and maximum sizes of a structure are defined in the CFRM policy (using the INITSIZE and SIZE keywords, respectively). By specifying a maximum that is larger than the initial size, the structure size can be quickly increased using the **SETXCF START,ALTER,STRNM=structure_name,SIZE=size** command. If the maximum size is *not* larger than the current structure size, the CFRM policy must be updated and the structure rebuilt (which is more disruptive than doing an ALTER) to increase its size.

The correct ratio between INITSIZE and SIZE is always difficult to determine, because there is no one number that is correct for all structures or all situations. A good guideline is that the SIZE value must be large enough to support twice the workload that the INITSIZE supports. This is the rule that the CFSizer uses when calculating the SIZE and INITSIZE values that it provides. You can afford to specify SIZE to be double the INITSIZE value for small structures, but for larger structures (large DB2 GBPs, for example), a SIZE value that is 25 - 50% larger than INITSIZE might be more appropriate. We do not recommend specifying a SIZE value that is more than twice as large as the INITSIZE value.

Health check: z/OS 1.12 delivers a new health check called XCF_CF_STR_POLICYSIZE that checks the ratio between the SIZE and INITSIZE values for all structures and generates an exception for any structure whose SIZE is more than twice the INITSIZE.

Structure full monitoring

In an attempt to help customers recognize when a structure is starting to fill up, XCF includes a function known as structure full monitoring. This function is automatically enabled, and wakes up on a timed basis to check the usage of any allocated structure with a FULLTHRESHOLD value (in the CFRM policy) greater than zero.

If structure full monitoring detects a structure where either the percent of entries in use, the percent of elements in use, or the total structure usage exceeds the FULLTHRESHOLD value, it warns the operator with message IXC585E (Example 2-17). Add this message to your automation so that an email is sent to the person responsible for the CFRM policy, warning him that the structure usage appears to be increasing and that perhaps the structure sizes should be increased.

Example 2-17 Structure full monitoring warning message

```
IXC585E STRUCTURE MWG1APPLIST1 IN COUPLING FACILITY CFPROD2,
PHYSICAL STRUCTURE VERSION C5ED8A4C 0B0A11C6,
IS AT OR ABOVE STRUCTURE FULL MONITORING THRESHOLD OF 85%.
ENTRIES:  IN-USE: 601          TOTAL:      7406,  8%  FULL
ELEMENTS: IN-USE: 41047        TOTAL:     44068, 93%  FULL
EMCS:     IN-USE: 395          TOTAL:     3941, 10%  FULL
```

There are certain structures that have special considerations in relation to structure full monitoring. One is the JES2 checkpoint structure. When it first accesses the checkpoint structure, JES2 pre-formats it, just as it does with the checkpoint data set on DASD. As a result, the apparent usage of the checkpoint structure never changes until you make a change to JES2 that results in an increase in the checkpoint size. However, if the amount of formatted space exceeds the FULLTHRESHOLD value for that structure, you get messages every few minutes, warning you that the structure threshold has been exceeded. To avoid these messages for this structure, you can alter the structure definition to specify FULLTHRESHOLD(0).

The other type of structures that can require special attention is the Logger structures. These structures are designed to offload to DASD when a Logger-specified threshold is reached. A common threshold is 80%, which is the same as the default FULLTHRESHOLD value. This can result in frequent messages warning you that a Logger structure has exceeded its threshold, shortly followed by another message telling you that the structure usage is now below the threshold. For these structures, specify a FULLTHRESHOLD value that is at least 5% higher than the HIGHOFFLOAD value specified for the log streams that reside in that structure.

It is important to understand that the structure full monitoring process is timer-driven. It is not invoked the instant that a structure exceeds the FULLTHRESHOLD value. Rather, to minimize overhead, it wakes up every so many seconds. As a result, if the use of the storage in a structure is increasing rapidly, it is possible that the structure will fill before structure full monitoring gets a chance to issue a message. Nevertheless, this is a valuable tool to help you detect a trend of increasing storage usage for CF structures.

ALLOWAUTOALTER

As described above, most structures support the ability to change the structure size using the SETXCF START,ALTER command. Of those structures, certain connectors provide a function to adjust the structure size, or the ratio of entries to elements, based on the structure usage. Other connectors, such as System Logger, react to structures filling up by moving the structure contents to a data set on DASD. For the remaining structures, you have the option of allowing XCF to automatically adjust the structure size when the structure exceeds the FULLTHRESHOLD value. This is achieved by specifying ALLOWAUTOALTER(YES) for that structure in the CFRM policy (note that the default ALLOWAUTOALTER setting is NO).

The structures that auto alter is valuable for are:

- ▶ XCF signaling
- ▶ IMS IRLM lock (only affects the record data portion of the structure)
- ▶ DB2 IRLM lock (only affects the record data portion of the structure)
- ▶ IMS OSAM and VSAM
- ▶ DB2 SCA
- ▶ DB2 GBP (but set the FULLTHRESHOLD value above castout threshold)
- ▶ DFSMS enhanced catalog sharing
- ▶ DFSMS VSAM RLS lock
- ▶ HSM common recall queue
- ▶ WLM LPAR Cluster (IRD) structure
- ▶ MQ shared queues admin and application structures

However, the MQ Development group prefers that MINSIZE be set equal to INITSIZE, to ensure that no storage is stolen from the structure if the CF storage utilization exceeds 90%.

For structures used by another vendor's products, consult the vendor to see whether they support and recommend the use of auto alter.

If XCF identifies a structure that requires some sort of alteration (either the ratio between entries and elements or changes to the structure size), it issues a series of messages (IXC588I, IXC589I, and IXC590I). Define these messages to your automation product to send an email to the people responsible for your CFRM policy, informing them of the fact that XCF encountered a structure that required a change. If this change is unexpected, investigate what is happening that caused the usage of the structure to change. If the change is simply a result of workload growth, then change the structure sizes (both INITSIZE and SIZE) in the CFRM policy.

Minimum structure sizes

Each new CF level tends to increase the storage requirement for at least certain structures. Depending on the structure definitions, it is possible that a structure that is defined with very small INITSIZE and SIZE values might fail allocation after a CF level upgrade.

To avoid the possibility of this situation affecting you, specify an INITSIZE value of *at least* 10 MB for every structure. (This is based on CF level 16. If you are using a higher CF level, increase that size accordingly.)

Check that current structure size equals INITSIZE

As discussed in “ALLOWAUTOALTER” on page 58, structure sizes might get adjusted, either by auto alter or by the program using that structure. Such changes are an indication that the usage of the structure is changing, prompting the adjustment to the structure size. However, when the structure is deleted and reallocated, it will be allocated at the INITSIZE again.

If a structure size has changed, someone should be made aware of this so that they investigate the reason for the change. It might be that the usage of the structure is just increasing, in which case, adjust the INITSIZE and SIZE values in the CFRM policy accordingly.

To help you detect changes in structure size, a simple piece of automation can be created that compares the current structure size to the INITSIZE value (both of these values are

contained in the output to a `D XCF,STR,STRNM=structure_name` command) and raise an alert if the two values are not the same.

PREFLIST should have at least two CFs

To ensure that all your structures can be rebuilt or reallocated in a CF if the CF in which they normally reside is unavailable, every structure should have at least two CFs in the PREFLIST, and those CFs should reside in separate physical processors.

Health check: The `XCF_CF_STR_AVAILABILITY` health check verifies that every structure has at least two CFs in its PREFLIST and that those CFs are not in the same processor. This check should be enabled and any exceptions investigated.

In addition, the `XCF_CF_STR_PREFLIST` check verifies that each structure is allocated in the first available CF in its PREFLIST. If this check generates an exception, it can be an indicator that one of your CFs is down and that the structures that normally reside in that CF have had to be allocated in a separate CF. For this reason, immediately investigate any exceptions raised by this check and address the cause of the exception.

z/OS 1.12 included an enhancement to the XCF REALLOCATE function. Before issuing a `SETXCF START,REALLOCATE` command, you can now issue a `D XCF,REALLOCATE,TEST` command. This does not result in any actual movement of structures. However, it does report what changes would be made if the `SETXCF START,REALLOCATE` command was issued.

Use of exclusion lists

One of the attributes that you can specify when you define a structure in the CFRM policy is the name of the structures that you do *not* want to be in the same CF as this structure. For example, if you have two structures for a given XCF transport class, you want those structures to be placed in separate CFs for availability reasons. In that case, when you define each structure, include an `EXCLLIST` specifying the name of the other structure.

While this is a useful capability for special cases, such as with the XCF signalling structures, the RACF primary and backup database cache structures, IMS VSO primary and secondary structures, and possibly the JES2 normal and alternate checkpoint structures, use it sparingly. If you were to define `EXCLLIST`s for a large number of structures, XCF's decision about which is the best CF for a given structure can become very complex, resulting in confusion about why XCF is not placing a structure in the CF that you think it should be placed in.

Health check: The `XCF_CF_STR_EXCLLIST` check verifies that any structure with an `EXCLLIST` is not in the same CF as the structures named on the `EXCLLIST` parameter.

Never specify a REBUILDPERCENT greater than 1

The `REBUILDPERCENT` value that you specify for a structure, together with the weight of all connected systems (if SFM is active), is used when deciding whether a structure should be moved to another CF following a connectivity failure from a system to a CF.

For nearly every structure, the pain or inconvenience of losing access to a structure is worse than the short pause in processing that is required to move the structure. For this reason, either do not specify *any* `REBUILDPERCENT` keyword on the structure definition in the CFRM policy or, if you do want to specify this keyword, do not specify a value larger than 1.

No-subchannel conditions

When z/OS attempts to send a request to a CF, it must first find an available subchannel for the target CF. If all subchannels are busy, the request is delayed. If the request is a synchronous request, in many cases the request is converted to an asynchronous request, resulting in an increased response time for that request.

No-subchannel conditions can be a result of a larger number of concurrent CF requests than there are subchannels. Or they can result from very *bursty* traffic, such as DB2 castout processing. In the former case, relief can be obtained by adding more CF link CHPIDs (each CF link CHPID results in seven more CF subchannels in z/OS). In the case of the latter, additional CHPIDs might provide some relief but are unlikely to make the condition disappear completely.

The RMF Coupling Facility structure activity report contains the number of no-subchannel conditions (Figure 2-16). Ideally, aim for zero no-subchannel conditions.

STRUCTURE NAME = DSNFK00 GBP1			TYPE = CACHE			STATUS = ACTIVE PRIMARY			DELATED REQUESTS				
SYSTEM NAME	# REQ	REQUESTS		-SERV TIME (MIC) -			REASON	#	% OF	AVG TIME (MIC)			
	AVG/SEC	# REQ	% OF ALL	AVG	STD_DEV			REQ	REQ	/DEL	STD_DEV	/ALL	
FK01	1582K	SYNC	259K	8.2	29.7	26.2	NO SCH	3240	0.2	4150	4593	8.5	
	1758	ASYN	1320K	41.9	149.3	815.4	PR WT	0	0.0	0.0	0.0	0.0	
		CHNGD	3242	0.1	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	
							DUMP	0	0.0	0.0	0.0	0.0	
FK02	0	SYNC	0	0.0	0.0	0.0	NO SCH	0	0.0	0.0	0.0	0.0	
	0.00	ASYN	0	0.0	0.0	0.0	PR WT	0	0.0	0.0	0.0	0.0	
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	
							DUMP	0	0.0	0.0	0.0	0.0	
FK03	1571K	SYNC	1218K	38.6	41.1	30.4	NO SCH	19	0.0	13018	14874	0.2	
	1746	ASYN	353K	11.2	257.1	543.2	PR WT	0	0.0	0.0	0.0	0.0	
		CHNGD	19	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	
							DUMP	0	0.0	0.0	0.0	0.0	
FK04	0	SYNC	0	0.0	0.0	0.0	NO SCH	0	0.0	0.0	0.0	0.0	
	0.00	ASYN	0	0.0	0.0	0.0	PR WT	0	0.0	0.0	0.0	0.0	
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	
							DUMP	0	0.0	0.0	0.0	0.0	
FK05	0	SYNC	0	0.0	0.0	0.0	NO SCH	0	0.0	0.0	0.0	0.0	
	0.00	ASYN	0	0.0	0.0	0.0	PR WT	0	0.0	0.0	0.0	0.0	
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	
							DUMP	0	0.0	0.0	0.0	0.0	
FK06	0	SYNC	0	0.0	0.0	0.0	NO SCH	0	0.0	0.0	0.0	0.0	
	0.00	ASYN	0	0.0	0.0	0.0	PR WT	0	0.0	0.0	0.0	0.0	
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	
							DUMP	0	0.0	0.0	0.0	0.0	
TOTAL	3153K	SYNC	1477K	46.9	39.1	30.0	NO SCH	3259	0.1	4202	4759	4.3	-- DATA ACCESS --
	3504	ASYN	1673K	53.0	172.0	767.3	PR WT	0	0.0	0.0	0.0	0.0	READS 265951
		CHNGD	3261	0.1			PR CMP	0	0.0	0.0	0.0	0.0	WRITES 885213
							DUMP	0	0.0	0.0	0.0	0.0	CASTOUTS 800487
													XI'S 188576

Figure 2-16 Determining number of no-subchannel and changed conditions

Changed requests

In the same RMF report, you will see that CF requests are broken into one of three categories:

- ▶ SYNC
- ▶ ASYN
- ▶ CHNGD

CHNGD (changed) is the number of synchronous CF requests that were converted to be asynchronous because all subchannels were busy at the time that the request was issued.

Prior to z/OS 1.2, CHNGD was a good indication of high utilization on the CF links. However, the heuristic algorithm introduced in z/OS 1.2 can convert synchronous requests that are likely to have a long response time into asynchronous requests. If all subchannels are busy when an asynchronous request is issued, this is *not* reported as a CHNGD request (because it was already converted to asynchronous prior to discovering that all subchannels were busy). As a result, while the number of CHNGD requests should be monitored, it is not as reliable an indicator of high CF link utilization as it used to be.

Ensure that structures defined to use duplexing are actually duplexed

The CFRM policy provides two options for specifying that a structure should be duplexed:

- ▶ DUPLEX(ENABLED)
- ▶ DUPLEX(ALLOWED)

DUPLEX(ENABLED) specifies that the structure be duplexed as soon as it is allocated. DUPLEX(ALLOWED) means that the structure can be duplexed, but that duplexing must be initiated by a SETXCF START,REBUILD,DUPLEX command. There are valid reasons for using either of these mechanisms.

Regardless of how you start duplexing, if you specify either of these keywords for your structures, it is assumed that you want those structures to be duplexed. Therefore, either write automation to check that all DUPLEX(ENABLED) or DUPLEX(ALLOWED) structures are actually duplexed (ideally, such a piece of automation first checks to ensure that two CFs are available), or enable and monitor the XCF_CF_STR_DUPLEX health check that was added in z/OS 1.10.

Monitor for structure hang messages

When z/OS attempts to rebuild a structure, it first sends a message to every connected address space to coordinate the rebuild. If one of the connected address spaces does not respond, the rebuild cannot proceed. To help you identify and investigate the cause of the hang, message IXL040E is issued, naming the address space that has not responded. Add this message to your automation to ensure that it is immediately brought to the attention of the people responsible for investigating the situation.

Additionally, z/OS 1.12 adds a new SFM function called CFSTRHANGTIME that automatically cancels the address space identified on the IXL040E message after the number of seconds that you specify on the CFSTRHANGTIME keyword in the SFM policy. There is also a new health check called XCF_SFM_CFSTRHANGTIME that raises an exception if this SFM function is not enabled.

Health check: The XCF_SFM_CFSTRHANGTIME health check was added in z/OS 1.12. It raises an exception if the CFSTRHANGTIME function is not enabled in SFM. By default, the check expects to find a CFSTRHANGTIME value of 300 seconds, but you can override this if you feel another value is more appropriate for your environment.

Monitoring lock structures

In a data-sharing environment, perhaps the most important structures in the Parallel Sysplex are the lock structures used by the database managers. To minimize the overhead of data sharing, the response time for lock requests should be as short as possible. Similarly, the availability of the data sharing environment is dependent on the ability to nondisruptively recover the lock structure in case of a CF failure.

Lock structure response times

Requests to a CF lock structure typically take less time than requests to a cache or list structure. As a result, when looking at the response times being delivered for various structure types in the CF, you expect to see the lock structures being at the lower end of that range. The lower the response times are, the better.

Depending on your configuration, you might find that certain systems experience different response times from the same structure than others. For example, one system might be connected to the CF containing the lock structure by InfiniBand links, whereas another might be using ISC links. If it is reasonable to do so, try to place each lock structure in whichever CF delivers the best response time to the largest user of that structure.

Ratio of synchronous to asynchronous requests

Related to the response time being delivered by a structure is the question of whether the requests are being sent to the CF synchronously or asynchronously. For certain structures, the additional response time associated with asynchronous requests is unimportant. However, because you want lock structures to deliver the best response time possible, you want as high a percentage of lock requests as possible to be synchronous requests.

Monitor the response times being delivered for your busy structures and the percentage of requests that are being sent synchronously. If APAR OA28603 is applied to your system, you can also find the threshold above which requests are sent asynchronously for each structure. To do this, issue the D XCF,C command and check the four lines that display the threshold (Example 2-18). Note that the threshold value can be different for each member of the sysplex.

Example 2-18 Displaying XES heuristic algorithm thresholds

```
D XCF,C
IXC357I 01.51.30 DISPLAY XCF 060
SYSTEM #0$2 DATA
      INTERVAL  OPNOTIFY  MAXMSG  CLEANUP  RETRY  CLASSLEN
           85      85      2000      15      10      956

      SSUM ACTION  SSUM INTERVAL  SSUM LIMIT  WEIGHT  MEMSTALLTIME
           ISOLATE           300      720      50           NO

PARMLIB USER INTERVAL:      85
DERIVED SPIN INTERVAL:      45
PARMLIB USER OPNOTIFY:      85

MAX SUPPORTED CFLEVEL: 16

MAX SUPPORTED SYSTEM-MANAGED PROCESS LEVEL: 16

SIMPLEX SYNC/ASync THRESHOLD:      26
DUPLEX SYNC/ASync THRESHOLD:      26
SIMPLEX LOCK SYNC/ASync THRESHOLD:  26
DUPLEX LOCK SYNC/ASync THRESHOLD:  28
```

Lock structure contention

Lock contention is a complex subject. However, at its most simple, a lock request can result in:

- ▶ No contention: There are either no other users interested in the same lock entry in the lock structure, or the type of access that they requested is compatible with this request.
- ▶ False contention: In this case, the same lock structure lock entry is being used to protect two separate resources and the lock requests appear to be incompatible (for example, both requestors want exclusive access to the lock entry). However, in reality, the requests are for separate resources, so there is not *really* contention between the two requests.
- ▶ Real contention: In this case, there are incompatible requests for a lock entry, and both requests are for the same resource.

Contention on the lock entry, whether it is real or false, results in additional work for XCF and the requestor. This means elongated response times and additional CPU usage. To minimize these unpleasant impacts, attempt to minimize contention.

False contention can be addressed, to some extent, by increasing the number of lock entries in the structure (by increasing its size)³. The normal guideline is that the number of requests that encounter false contention should not exceed 1% of the number of lock requests for that structure. If your lock structures have false contention significantly higher than this, consider increasing the lock structure size by an amount that provides for a doubling of the number of lock entries.

Real contention is more difficult to address because it is caused by two units of work trying to use the same resource in incompatible ways. The normal guideline is that the maximum real contention should not exceed 2%. If the real contention regularly exceeds this amount, work with the application owner and the batch schedulers to see whether work can be scheduled in a way to reduce contention.

Monitor the contention of your busiest lock structures and take action if the contention starts approaching these thresholds.

For a far more detailed discussion of DB2 locking and lock contention, see *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725.

Lock structure size

The size of each lock entry in a lock structure must be a power of two (either two, four, or eight bytes wide). The number of lock entries in a lock structure also must be a power of two. Therefore, to optimize the space available in a lock structure, define all lock structures to be a few MBs larger than a power of two. The reason for allowing a few additional MBs is to allow for control information that is present in every structure.

Cache structures

As customers continue to drive more exploitation of data sharing, and as the data sharing groups process more data and more transactions, the performance of the cache structures associated with database managers is also increasing.

³ The number of lock entries in a lock structure *must* be a power of 2.

There are basically three ways that a cache structure can be used by a database manager:

Directory only	This type of structure, typically used by IMS for its OSAM and VSAM structures, is only used to register interest in shared data. No actual data is stored in the cache structure. It is simply used to keep track of which data-sharing group members have an in-storage copy of a given piece of data.
Store through	This type of structure, used by VSAM/RLS and optionally by IMS OSAM, is also used to track interest in shared data. However, when a database manager updates a piece of data and saves it to the database on DASD, a copy of the updated piece of data is then saved in the cache structure, meaning that other members of the data-sharing group can retrieve that data much faster than if they had to read it from DASD.
Store in	This type of structure, used by DB2 and by IMS for its VSO databases, also tracks interest in shared data. However, in this case, when the associated database manager updates a piece of data, the updated data is not written to DASD at this time. Rather, the updated data is written to the cache structure, and later moved to DASD using a process called castout.

To contribute to the optimum performance and availability, there are some recommendations for cache structures that should be followed, which we discuss in the following section.

Cache structure performance

While you want to be able to retrieve and save data as quickly as possible, the performance of cache structures is typically not as critically important as the performance of the lock structures. Remember that in a non-data-sharing environment, all locking is done in storage, so we want sysplex locking to deliver performance as close to that as possible. Conversely, in a non-data sharing environment, any data that is not found in a local buffer needs to be retrieved from DASD, a process that is many times slower than retrieving the same data from a CF.

Also, the amount of data being moved back and forth to the CF is typically much larger for a cache structure than for a lock structure, and the more data that is being moved, the longer you expect the response time to be (this applies to both DASD and CF). Because of this profile of moving larger amounts of data, cache structures tend to benefit from large bandwidth CF links more so than lock structures. For example, if you have ICFs, placing a primary GBP structure in the same processor as the DB2 that uses that structure the most can deliver small performance benefits.

Therefore, check the response times of your busiest structures to ensure that they fall within the expected range. Also check for both daytime and batch window activity, as the two are often quite different. If you find that the response time trend is unfavorable, investigate to determine the cause and rectify it if possible.

Long-running CF requests

There are many types of CF requests. Certain ones, like lock requests, typically require very little processing in the CF, so these normally are expected to experience short response times. Other requests, especially ones that impact many objects within the CF, can require extensive processing in the CF, and therefore typically see longer response times. Examples of the latter are when you start an image copy of a database. The image copy program must let the CF know about the pages that it will be reading. This uses a process known as register name list (RNL). DB2 uses RNL any time that it drives sequential pre-fetch activity for a database or table. By their nature, these commands can run for a long time. Other examples

of long-running commands are batch unlocks, DB2 castout processing, and batch writes that can occur during a database reorganization.

Normally, this is not a problem. The CFCC times out the commands after a period of time, allowing other, shorter commands to run before the database manager resubmits the RNL request to carry on from where it left off. However, if your batch scheduling submits a huge number of these types of jobs around the same time (or if each job initiates a large number of parallel tasks), the sheer number of these requests can be such that other, shorter, requests see significant delays.

This situation can be observed if you notice a significant increase in CF CPU utilization and elongation of CF response times—hundreds of microseconds longer than normal. If this description matches your environment, the impact can be reduced by:

- ▶ Trying to schedule the batch jobs so that they are not run in parallel.
- ▶ If the structures are duplexed, try to avoid having all the primary structures in one CF and the secondaries in the other. Many of the long-running commands only run against the primary structures, so if all the primaries are in the same CF, that CF is very busy, while the other CF has very little work. This is discussed further in 4.2.11, “Tablespace to buffer pool relationship” on page 110.
- ▶ Try to place the busiest cache structures in separate CFs.
- ▶ If one cache structure is the target of most of the requests, work with the database DBAs to see whether certain tables can be moved to a separate buffer pool, or whether the image copies that use that structure can be scheduled at different times.

A change to the CFCC code in CF Level 16 resulted in a performance impact for a small number of customers that have exceptionally large batch database sequential processing. CFCC Service Level 2.25, delivered with Driver 79, MCL6, or Driver 76 MCL15, changes CFCC processing to address this impact.

Cache structure cross invalidates

Cache structures consist of data elements (that hold the data being cached) and directory entries. The directory entries are used to track which system has an in-storage copy of a given piece of data in a shared database.

Depending on how the structure is being used, and the ratio between data entries and directory entries in the structure, it is possible that the CF will need to reuse a directory entry that is currently being used to track interest in a data item. When that happens, the CF is no longer able to track interest in that piece of data, so it sends a cross-invalidate signal to the systems that have a local copy of that data, telling them that their buffers are invalid. This potentially results in a performance impact because if that data is needed again, a fresh copy must be obtained from the CF or from the database on DASD, even though the in-storage copy might still be valid.

To determine the level of this activity, obtain an RMF Coupling Facility usage summary report for a peak interval (which could be during the batch window). The right-most column contains the number of directory reclaims that caused a cross-invalidate. Discuss any non-zero values with the system programmer for the associated database manager. The solution might be to increase the structure size, or it might be to alter the ratio between data elements and directory entries.

Production DB2 GBPs should be duplexed

DB2 uses a store-in cache model for its GBPs. This means that updated database pages are written to the GBP at commit time, and then externalized to DASD some time after that. If the CF containing the GBP structure were to fail before the updates pages are saved to DASD,

the updated pages can be recovered from the DB2 logs, but this can be a time-consuming process.

However, if the GBPs are duplexed, updated pages are written to both instances of the GBP structures and kept in both instances until the pages are hardened to the database on DASD. As a result, if either CF is lost, there is no need to recover any pages from the DB2 logs because they are already in the surviving GBP instance.

The cost of duplexing the GBPs is minimal, and the potential benefits in case of a CF outage are significant. For that reason, duplex the GBP structures for *any* production DB2 data sharing group.

Health check: Associated with the use of duplexing for CF structures is the XCF_CF_STR_DUPLEX health check. This check verifies that every structure that is defined with DUPLEX(ALLOWED) or DUPLEX(ENABLED) in the CFRM policy is currently duplexed. If this check raises an exception, immediately identify the reason for the structure not being duplexed and address the root cause.

2.3.3 How many sysplexes to have

A common point of discussion is the question of how many sysplexes you should have, and specifically whether you should have a separate test sysplex. The answer is that it depends. The reason for this is that every customer situation is different and everyone has unique requirements and limitations. However, we provide discussion points that can help you come to the correct decision for your environment.

Let us start with the question of whether a test sysplex is required. In response, we ask how you normally address your testing requirements. For example:

- ▶ Do you have separate CICS regions for test and production?
- ▶ Do you have separate database subsystems for production and test data?
- ▶ Do you keep test data sets separate from production data sets?

If the answer to one or more of these is yes, then you appear to feel that it is a best practice to separate the test from production at the subsystem level. If that is so, would the same logic not extend to the sysplex level? Having separate systems for test and production does provide more separation than having separate test and production CICS regions. However, it is possible for a problem in one system in a sysplex to spread to other systems. So if you believe that it is necessary to keep test CICS transactions separate from production ones, for example, then the logical extension of that is that you should also keep all your test work in a separate sysplex from your production work.

IBM best practice: The normal IBM recommendation is to only have production work in the production sysplex. The more important your availability is to you, the stronger the case for maintaining separation between test and production work.

Other things to consider in your decision about how many sysplexes to have, and which systems should be in each sysplex, are:

- ▶ Which sets of systems share the same data sets or databases? For example, if applications A, B, and C are closely related, it makes sense to place them in the same sysplex and give them direct access to the data rather than moving the data back and forth from one application to the other.
- ▶ Which sets of systems share the same security database? While it is possible to have multiple security databases in one sysplex, great care must be exercised over which DASD and tape volumes can be accessed from each system. You do not want a volume containing production data sets being accessible from a system that is using the test security database.
- ▶ Do you have an audit or legal requirement for a very clear separation between test and production?
- ▶ What is your software management strategy? Do you need to be able to test new sysplex functions in a safe environment?
- ▶ There might be financial considerations, particularly relating to aggregation and software license charges.
- ▶ Do not forget that each sysplex requires its own sysplex resources (CFs, CDSs, and so forth), although the ability to share PSIFB links between multiple sysplexes might reduce the additional cost of a separate test sysplex compared to using ICB or ISC links.

2.4 Testing with copies of a sysplex

When you use a remote copy mechanism to create a backup or disaster recovery copy of your systems, you effectively create two identical copies of the same sysplex. *Great care* must be exercised when testing the second copy.

Consider what would happen if somehow the DR test copy of your production sysplex were to get access to any real production resources, or if somehow the network got connected to the test copy. As far as any user can tell, this is the real production environment. So users could potentially log on and start doing their work, thinking that they are on the production system. However, all the changes that they are making are only being applied to a test copy.

Or what happens if the test copy of the sysplex gains access to a CF that is being used by the real production sysplex? There are safeguards in place to make sure that *different* sysplexes cannot access the same CF. However, because this is a true copy of the production environment, there is no way for the CF to know that this is not really a production system. So access will be allowed, inevitably resulting in a sysplex outage.

If the test version of the sysplex does not exist in the same location as the production sysplex, this eliminates any possibility of the test copy of the sysplex having access to any production resources (in particular, the production CFs).

However, if you have no choice but to test the DR copy of the production sysplex in an environment where it could potentially get access to a production CF, then we *strongly* recommend that you do not mirror the CDSs (except possibly for the Logger CDS). Instead, set up a dedicated set of CDSs that are only used by the DR sysplex.

Also consider setting up your configuration in HCD so that the only LPARs that are allowed to access any CF LPAR are the ones that normally connect to that LPAR. This can be achieved using the access lists for the CF LPARs. If you need to move a system to a backup LPAR at

some time in the future, the IODF can always be updated *at that time* to add the new LPAR to the access list.

Another thing you might consider is the use of the CFRMOWNEDCFPROMPT parameter in the COUPLExx member. Specifying YES on this parameter causes the operator to be prompted during the IPL of the *first* system in a sysplex IPL before the system will use a CF. Figure 2-17 shows an example.

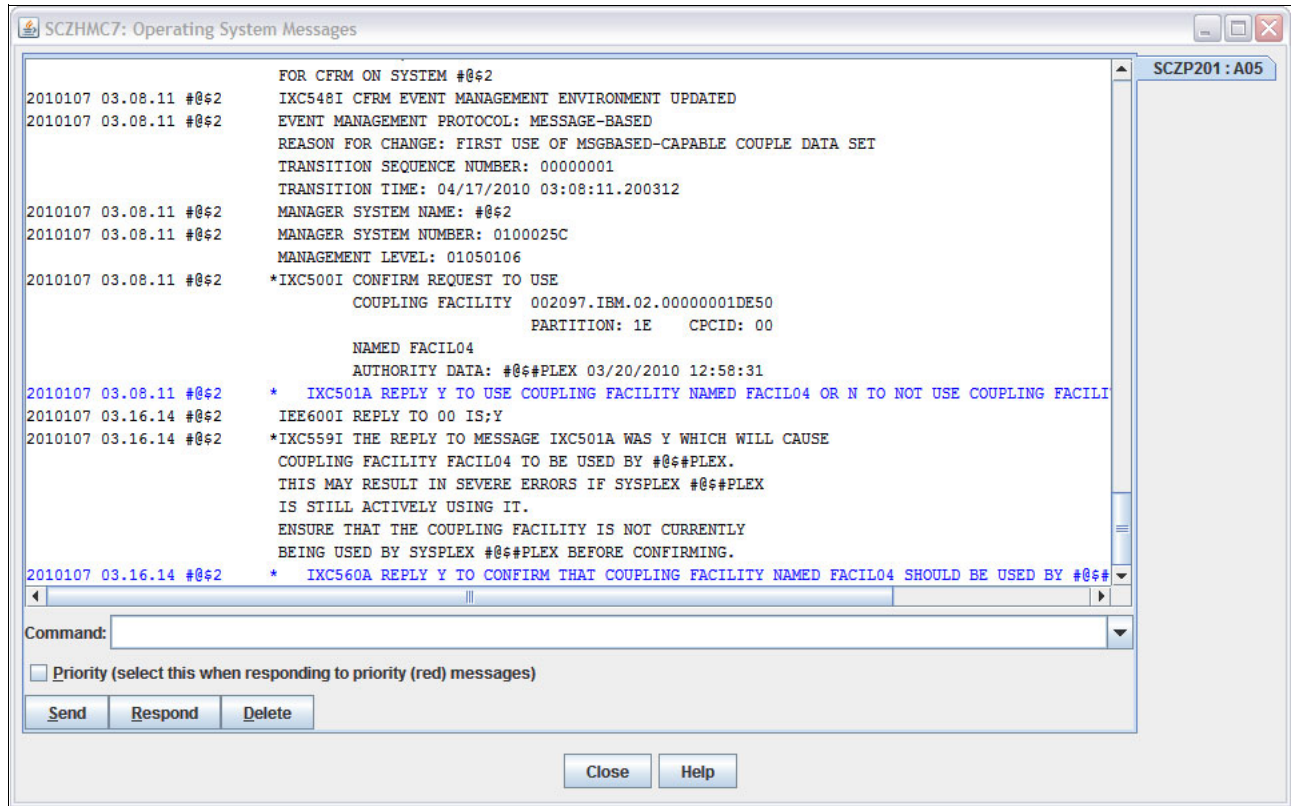


Figure 2-17 Use of CFRMOWNEDCFPROMPT parameter

Note, however, that for this to be effective, the responder *must* verify that the processor and LPAR listed in the IXC500I message are actually the ones that you are expecting. Remember that if you are doing a DR test, the system and sysplex name match the live version of that sysplex and, depending on your processes, the CF names could match those on the live sysplex also. Also, this checking is only done for the first system to be IPLed in the sysplex. So, if the first system does not access the incorrect CFs, but a subsequent system to be IPLed in that sysplex *does* access a CF being used by the live sysplex, this function does not protect you.

Regardless of which mechanism you use to provide the CDSs that the DR sysplex will use, the best protection against such situations is to exercise great care any time that you do any testing of a copy of any sysplex.

Never copy a CDS

It is not uncommon to find situations in which a CDS gets copied from one sysplex to another for various types of testing. This must *never* be done. The CDSs contain information that is specifically intended to ensure that one sysplex cannot touch a CF belonging to another sysplex. If this were to happen, a sysplex outage is nearly guaranteed. By copying the CDS, you completely negate all the mechanisms designed to ensure that this does not happen.

New CDSs must *always* be created by formatting a new CDS with the IXCL1DSU program, and then loading it with the appropriate policy.

2.5 System Logger

The MVS System Logger provides a generalized logging capability for saving and recalling log records. One example of a user of Logger is OPERLOG. The CONSOLE address space on all systems writes system messages to the OPERLOG log stream, thereby creating a single repository for all console messages in a sysplex. OPERLOG can be thought of as the merged syslogs from all systems in a sysplex. Another common user is CICS, which writes its DFHLOG and DFHSHUNT log records to Logger.

Because it is intended to provide logging services for many different exploiters, Logger provides the installation with a lot of control over how a given log stream should be managed. This control is generally exercised via the log stream definitions in the LOGR CDS.

This section provides guidelines for setting up Logger and your log streams in a manner that will provide the levels of performance and availability that you require.

2.5.1 Logger protection of log data

To protect the log records that are passed to it by its exploiters, Logger always keeps at least two copies of those records until they are hardened to the Logger offload data sets.

The location of the two copies is controlled by the installation when you define each log stream. The possible locations are:

- ▶ A CF structure
- ▶ A Logger staging data set
- ▶ A Logger data space

Each location has unique availability and performance characteristics. A CF structure is faster than a staging data set on DASD, as is a Logger data space. However, if the z/OS system fails, the data space copy of the log records will be lost. If the CF is volatile and the CF fails, the CF copy will be lost.

Logger is aware of the processor that the CF structure is located in (and its volatility status) and the processor that the system is running on, and it understands the concept of failure domains. It is able to use this information to dynamically determine where it should keep the log records for a log stream, based on the availability requirements that you specify when you define the log stream. For example, if you use DASD mirroring for disaster recovery, you probably want a copy of all log data on DASD so that it will be sent offsite. In that case, you can force Logger to use a staging data set for that log stream.

Or you might say that you want to protect data in a log stream from a processor failure, but beyond that, you want the best performance, so you define the log stream using the DUPLEXMODE(COND) keyword. This informs Logger that a staging data set should be used if the Logger structure is in the same failure domain as this system. Otherwise, the second copy of the log data should be placed in a Logger data space.

Or you might say that you just want optimal performance, but do not really care if any data is lost. The OPERLOG log stream might be an example of such a log stream. In this case, you can tell Logger (via the log stream definition) to only use the CF and a data space.

You can also define a log stream as being DASDONLY, meaning that Logger should place the log data in a staging data set and a data space, and not in a CF structure. It is important to note that log streams that are to be accessed by more than one system must be a CF log stream. DASDONLY log streams can only be accessed by exploiters on one system at a time.

2.5.2 Types of Logger exploiters

There are two basic ways in which an exploiter can manage its log data:

- ▶ It writes the log records and either never deletes them (OPERLOG, for example), or it deletes them in a large group a long time in the future (IMS Shared Message Queue, for example). These log streams are known as *funnel log streams*.
- ▶ It writes a log record, then very quickly deletes it. For example, CICS writes a DFHLOG record at the beginning of a transaction, then deletes it as soon as the transaction ends. These log streams are known as *active log streams*.

The two log stream types are managed in different ways for optimal performance. For funnel log streams, your objective should be to move the log data from the CF structure to an offload data set in as efficient a manner as possible, ensuring that the CF structure never fills up.

Conversely, because active log stream exploiters delete their log records shortly after they are created, ideally you want to ensure that these log records are never moved to an offload data set (because then you have the work of moving the log record, only to delete it a short time later).

Tip: A very easy way to see whether active log streams are being managed efficiently is to do a 3.4 in ISPF for the offload data sets associated with those log streams. The low-level qualifier of these data sets is similar to a GDG generation number. Every time that an offload data set fills, a new one with a higher sequential number is allocated. Because we do not want log records from active log streams being written to the offload data sets, the low-level qualifiers should all be very low numbers. If you spot any that have large numbers, either the CF structure or the staging data set associated with that log stream must be made larger.

You can find more information about active and funnel log streams and instructions on how best to define them in *Systems Programmer's Guide to: z/OS System Logger*, SG24-6898.

2.5.3 General Logger guidelines

There are certain guidelines for general System Logger setup that are applicable regardless of the type of log streams that you use.

GROUP attribute

There are operations in Logger that are only carried out by one task. However, there have been situations where that task can be delayed for a long time (for example, waiting for a migrated offload data set to be recalled). To assist customers that run both production and test applications in the same system to reduce the risk of activity for a test log stream impacting a production log stream, Logger introduced the ability to define a log stream as a PRODUCTION one or a TEST one, using the GROUP attribute. At the same time, a second set of tasks to handle these special activities was added. So there is now one set of tasks dedicated to handling TEST log streams, and one set for handling PRODUCTION log streams. Note, however, that there are restrictions on the number of resources that can be

consumed by log streams in the TEST group that do not apply to log streams in the PRODUCTION group.

If you do not specify the GROUP attribute, or for log streams that were defined before this capability was added, the log stream is defined as a PRODUCTION one.

If you run both production and test applications in the same system or in the same sysplex, use the GROUP attribute to assign the correct classification to your log streams.

Note that the applications connected to the log stream do not necessarily have to be test and production. They can equally be, for example, banking and insurance, although the keywords of TEST and PRODUCTION are fixed. Assigning some log streams to the TEST group and some to the PRODUCTION group allows you to take advantage of Logger's support for two tasks to do certain processing.

Offload data set CI sizes and SHAREOPTIONS

To facilitate moving data to the offload data sets in as efficient a manner as possible, it is important that the offload data sets are allocated with the correct CI Size. The default CI size, if you do not override it by using an appropriate data class, is 4 KB. However, the recommended CI SIZE to use for an offload data set is 24 KB, as this results in more efficient offloading of data⁴. If the CI size specified for the offload data set is not a multiple of 4 KB, message IXG263E is issued, and Logger does not use that data set.

The other VSAM data attribute that you need to be aware of for log stream data sets is the SHAREOPTIONS value. Log stream data sets must have a SHAREOPTIONS of (3 3). For log streams defined using a CF structure, warning message IXG267I, IXG268I, or IXG269I is issued when Logger recognizes that the appropriate attributes are not in use during its allocation of log stream data sets. The detection is done when log stream data sets are newly allocated, and also when Logger attempts to access existing log stream data sets.

Starting with z/OS 1.12, System Logger automatically corrects the SHAREOPTIONS value if it finds that an incorrect value has been specified or defaulted to when a new log stream data set is allocated. If this happens, message IXG282I is issued. Monitor for this message and adjust the log stream definition so that the correct SHAREOPTIONS value is used for future data sets.

Do not use default sizes for offload data sets

The default size for an offload data set (if you do not specify an appropriate LS_DATACLAS or LS_SIZE value) is only two tracks. This can result in a large number of new data sets being allocated every time that log data is offloaded from the CF or staging data set. For the offload to complete as efficiently as possible, you ideally want zero, or at least very few, allocations to occur during the offload. Therefore, make sure to specify an appropriate SMS data class or LS_SIZE value when you define the log stream.

Specify appropriate staging data set attributes for all log streams

Even if you do not intend to use a staging data with a particular log stream, still assign staging data set attributes that are appropriate for that log stream. The reason for this is that it is possible during certain types of recovery scenarios for Logger to assign and use a staging data set for a log stream. If you have not specified reasonable values, the staging data set values that Logger defaults to might result in less-than-optimal performance, thereby slowing down recovery.

⁴ Note that the only supported CI size for staging data sets is 4 KB.

Use DFSMS striping for busy staging data sets

While most modern DASD subsystems implement striping at the physical disk platter level, there are still performance benefits to be achieved by defining Logger staging data sets as SMS striped data sets. When Logger realizes that the staging data set for a log stream is striped, it is able to process multiple concurrent write requests for that log stream. If the data set is not defined as SMS-striped, Logger will only process one request at a time, regardless of the physical devices that the data set resides on.

Enable and monitor Logger health checks

There are three health checks provided by System Logger. The health checks raise exceptions in any of the following circumstances:

- ▶ If it finds that a staging data set has filled up
- ▶ When 90% of the entries in a Logger structure have been used
- ▶ When all the elements in a Logger structure have been used

If any of these exceptions are reported by the Health Checker, first determine the root cause. For example, in all these cases, the root cause might be a problem allocating offload data sets, or it might be that the connector is generating log records faster than Logger can move them to the offload data sets, or it might simply be that the structure or the staging data set is not large enough. After you determine the cause of the exception, make whatever adjustments are necessary.

Add Logger messages to your automation

Over recent releases of z/OS, new messages have been added to Logger to notify the operator of delays in Logger. Add these messages to your automation, both to ensure that the operator is aware of the delays and the reason for the delays and also to ensure that the system programmer responsible for the log stream is aware that there is a problem, and that the log stream definition potentially needs fine tuning.

The particular messages that we suggest monitoring for are listed in “System Logger messages” on page 174.

Format LOGR CDS with SMDUPLEX keyword

The functions that are available in Logger, and the flexibility with which log streams can be managed, are determined to an extent by the format level of the LOGR CDS. While the last update to LOGR CDS formats was delivered with z/OS 1.3, it is possible that you are still running with an old format CDS if the LOGR CDS has not been specifically formatted with the SMDUPLEX keyword.

Tip: Do not be put off by the use of the SMDUPLEX keyword. While support for System-Managed Duplexing is one of the enhancements supported by the new format that this delivers, there are also many other benefits that apply to all customers, regardless of whether you use System-Managed Duplexing.

For more information about CDS formats in general, see “CDS format levels” on page 17.

SMF Type 88 records and IXGRPT1 report tool

Logger creates SMF Type 88 records with information about the activity to its log streams. While the SMF data does not contain performance information such as response times, it does provide valuable information about the volume of data being written to each log stream and how the data is being managed once inside the log stream (for example, how much data is being moved to offload data sets).

The IXGRPT1 sample report program can be used to format these SMF records.

Offload frequency

When a log stream reaches the HIGHOFFLOAD threshold that has been specified for that log stream, Logger first physically deletes any log records that have been marked for deletion. If this brings the log stream utilization down to the LOWOFFLOAD threshold, then no further action is required. This processing typically takes very little time.

If the LOWOFFLOAD threshold has not been reached, Logger then moves log blocks to an offload data set, starting with the oldest log block, and continuing until the LOWOFFLOAD threshold is reached.

In general, it is better for Logger to have many small offloads than a smaller number of large offloads. A frequent question is how many offloads per minute is an acceptable number. The latest guidance is that the number per minute is not actually important. What is important is that the offload process does not run continually.

The SMF Type 88 record contains both the number of offloads that were initiated over the interval and the number of bytes that were moved to the offload data sets. By combining this with your knowledge of the data rate that your DASD can deliver, you can estimate the approximate total elapsed time for offload processing over the SMF interval. If you discover that offload is running nearly all the time, consider reducing the difference between the HIGHOFFLOAD and LOWOFFLOAD values, or decreasing the size of the associated Logger structure.

2.5.4 CF log stream guidelines

In addition to the general Logger guidelines, there are some that are specifically related to the use of CF log streams.

Not more than 10 log streams per structure

Do not place more than 10 log streams in each Logger CF structure.

Within the System Logger address space, there are up to 256 tasks to manage connect and disconnect requests for log streams residing in CF structures. However, a given task controls all the requests associated with its assigned structure, meaning that only one log stream connect or disconnect request within that structure can be processed at a time. It also means that for system level log stream recovery, only one log stream in this structure is recovered at a time. Therefore, having a very large number of log streams in a single structure can result in delays in processing connect or disconnect or recovery requests for certain log streams.

Also, Logger uses the LOGSNUM value (the maximum number of log streams that can be in a structure) that you specify in the Logger policy when deciding how many list headers to allocate in its CF structures. Logger then scans those list headers as part of certain structure-related activities. So, again, you can see that optimal performance is obtained when your Logger structures do not contain a large number of log streams.

Place log streams with similar attributes in the same structure

Storage within a Logger structure is divided into entries and elements. The correct ratio between entries and elements depends on the size of the log records. If the log records are very large, you need a lot of storage for the elements and not so much for the entries. If the log records are very small, you need more storage for the entries and not as much for the elements.

A given structure can only have one entry-to-element ratio. So, if you place a log stream with very large log records and one with very small log records in the same structure, you will probably end up with a entry-to-elements ratio that is not ideal for either log stream.

For this reason, try to place log streams with similar log block sizes in the same structure. For example, if you want to place CICS log streams in a CF structure, have one structure for the DFHLOG log streams (and maybe even one structure for DFHLOG for the TORs, and a different structure for the DFHLOG for the AORs) and a different structure for the DFHSHUNT log streams. This is more efficient than having one structure per CICS region, with that structure containing all the log streams used by that region.

Also attempt to place log streams with a similar level of activity in the same structure. The total storage in a Logger structure is divided evenly between the allocated log streams in that structure. If you have just two log streams in a structure, one with a write rate of 1 GB per minute and the other with a write rate of 1 KB per minute, both are given the same amount of storage in the structure. It is better to place log streams with similar write rates in the same structure, and then size the structure accordingly (a structure with a number of busy log streams has a larger INITSIZE and SIZE than one with not-busy log streams).

Try to have connectors on more than one CPU for each structure

When a connector to a log stream disconnects unexpectedly from that log stream, other connectors to log streams in the same *structure* (they do not necessarily have to be connected to the same log stream) are notified. To protect the data belonging to the connector that just went away, one of the other connectors initiates an offload to harden all the data in the log stream to an offload data set. This ensures that the data is still available even if the CF goes down before the original connector reconnects.

You can benefit from this processing (known as peer recovery) any time that multiple connectors are connected to the same structure. They can be in other systems on the same CPU or even in the same system. However, to provide the maximum resilience, spread connectors across multiple CPUs, so even if an entire CPU goes down, there is still a connector alive that can drive the offload.

Disable auto alter for Logger CF structures

Because Logger uses thresholds to initiate offload processing, we do not advocate using Auto Alter for Logger structures. Logger is designed such that, when the utilization of a log stream reaches a threshold defined by the installation, it either deletes or moves data in the log stream to bring it back to the low threshold. If you enable auto alter for a Logger structure, it is possible that auto alter processing will be invoked before Logger gets a chance to do its offload. As a result, no offload processing takes place until the structure finally reaches the SIZE value specified in the CFRM policy. Offload processing then continues as designed, but you have now lost the ability to increase the structure size without updating the CFRM policy and rebuilding the structure.

Avoid use of System-Managed Duplexing for Logger structures

The design of how Logger manages the data in its structure, particularly during offload processing, means that the combination of System-Managed Duplexing and Logger structures might result in poor performance.

If you want to protect a Logger structure from a CF failure, define the log streams in that structure to use staging data sets rather than duplexing the structure.

Ensure staging data sets are appropriate size

If you have a CF log stream that has the second copy of the log data in a staging data set, you do not want offloads from the structure being triggered by the staging data set filling up (because this effectively means that you are not getting the benefit of all the space that is defined for the structure).

You can detect this situation by examining the SMF88ETF and SMF88ETT fields in the SMF Type 88 records. If you find any cases where this is a non-zero value for a CF log stream, increase the size of the staging data set.

2.5.5 DASDONLY log stream guidelines

Sizing the staging data sets is critical. Making them too small results in the data set filling frequently, which can cause delays in the address space that is attempting to write to the log stream.

Making the staging data sets too large can result in the amount of data space storage for the log stream also being very large, consuming processor storage. Also, very large staging data sets can result in longer elapsed times for offload processing, especially if large volumes of data are going to be moved to offload data sets.

If you have DASDONLY log streams that will be expected to receive very high request rates, special care must be taken to ensure that the performance will be acceptable. The data sets should be placed on high-performance volumes that are not targeted by RESERVE requests. The HIGHOFFLOAD and LOWOFFLOAD values should be selected carefully. And you might consider using SMS data set striping to enable multiple concurrent I/O requests to the data set. Also conduct performance testing before placing the log stream into production, to ensure that the desired request rate can be achieved. And do not forget - if the production staging data sets will be mirrored, then the ones used for performance testing should also be mirrored to ensure that the performance test results are representative of the production environment.

2.6 Avoiding single points of failure

The primary objective of Parallel Sysplex is to deliver better availability through the removal of single points of failure. For example, if you have two DB2s and exploit data sharing, application availability can be maintained if one DB2 needs to be stopped. If you have two z/OS systems, applications can continue to run on the first system while the second one is IPLed to apply maintenance. Having two CFs enables you to move structures from one CF to allow nondisruptive maintenance of the CF.

Similarly, from a hardware configuration point of view, strive to have no single points of failure. Therefore, you need at least two channels to every DASD device. And if you use switches, the paths to each device should be routed over at least two switches. And if the DASD subsystem has interfaces that are failure-isolated from each other, ensure that the switches are connected to interfaces that are failure-isolated from each other.

The challenge that you face is that z/OS and its subsystems are designed to be highly resilient. If one channel to a device fails, all operations continue over the other paths. And, depending on how you are configured, such a failure might not even result in a performance impact.

This is excellent from the perspective of the business. Application availability is vital to the competitiveness of the company, and it is great that the applications remain available even if a component fails. However, from the perspective of the technical staff, such resiliency means that you might potentially lose a component and not be aware of it. Similarly, in a large, complex environment, it is possible that you might inadvertently configure the paths to a device with a single point of failure in common.

To help you detect single points of failure in the channel subsystem, IBM delivered a new system service called IOSSPOF in z/OS 1.10. This highly flexible service can help you detect various types of single points of failure in your I/O configuration.

For example, let us say that we have two data sets that back each other up (XCF-managed couple data sets are a good example). You want to ensure that no single failure can remove access to both data sets. So, the service can be given the name of two data sets and asked to check that they do not have a common single point of failure (for example, they must not both be on the same device).

Another example of a critical resource is your z/OS sysres. You want to be sure that a single component failure will not remove all access to that device. So, IOSSPOF checks to ensure that the device is accessible via at least two channels, configured on separate channel cards, and connected to at least two separate switches.

The XCF_CDS_SPOF health check uses this service to ensure that there are no single points of failure between the primary and alternate CDSs.

IOSSPOFD service

To make it easier to use this service, IBM provides a batch front end to IOSSPOF called IOSSPOFD. IOSSPOFD, together with documentation on its use, is available as a download from:

<http://www-03.ibm.com/servers/eserver/zseries/zos/unix/bpxalty2.html>

The IOSSPOFD tool must be installed on the system. Example 2-19 shows a sample of the JCL to use to have IOSSPOFD check whether there are any single points of failure between the volumes 1CDS01 and 1CDS02.

Example 2-19 Sample IOSSPOFD JCL

```
//KYNFR JOB (0,0),'RUN IOSSPOFD',CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//STEP EXEC PGM=IOSSPOFD
//STEPLIB DD DSN=SYS1.USER.LOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSIN DD *
        VOLSER1(1CDS01) VOLSER2(1CDS02)
/*
```

Note that the current release of IOSSPOF looks for single points of failure from the physical device level, all the way back into the CPC. So, if you are running on a processor with a single book, IOSSPOF flags that as a single point of failure. While this might be disconcerting, it *is* accurate, and all the other information provided by IOSSPOF is both accurate and useful.

Set up a batch job to run IOSSPOFD on a regular basis and check for single points of failure in critical resources. Examples would be:

- ▶ z/OS sysres volume
- ▶ Spare CDSs (Because XCF is not aware of the spares, they are not included in the XCF_CDS_SPOF check.)
- ▶ IODF volume
- ▶ Master catalog (if this is not on the IODF volume)
- ▶ Database manager primary and alternate log data sets
- ▶ Security database data sets
- ▶ JES spool and checkpoint data sets

There will be other critical resources, depending on your configuration and software mix. Tailor the job to match your environment.

2.7 z/OS Health Checker

The z/OS Health Checker is an invaluable tool to help you discover aspects of your system that do not conform to IBM best practices. However, it potentially does more than simply check your configuration against IBM best practices. There might be situations in which you are aware of IBM best practices, but you have a valid reason for not following them. Health Checker can help you in these cases as well. Many checks allow you to override the IBM-recommended value with one of your own choosing. So now, instead of looking for deviations from the IBM value, Health Checker looks for deviations from your value.

Another benefit of the Health Checker is that it warns you if something is changed without your knowledge. Let us say that you set the widget parameter to 17, based on a unique characteristic of your system. Then your colleague goes to an IBM class and hears that the IBM-recommended value is 12. He comes back from the class and changes your override back to 12, unaware of the reason for you using your override. Without Health Checker, such a change might remain in place forever without your becoming aware of it. However, if you have Health Checker, and requested it to monitor for your override of 17, the next time that it checks and finds that your override has been changed, it warns you.

Taking the same example a step further, let us say that IBM changes its recommendation from 12 to 18, based on our experiences with customers and the speed of the latest processors. This might mean that you should change your override to 25 instead of 17. However, because you do not know that the IBM recommendation has changed, nothing drives you to change your override. If you had asked Health Checker to monitor for your override, and IBM then ships a PTF for Health Checker to update the IBM-recommended value, Health Checker does not warn you that the IBM recommendation has changed, prompting you to re-examine the new correct value for your system.

At the time of writing, there are over 200 IBM-provided health checks, covering over 30 aspects of z/OS. Additionally, ISV vendors are also using the Health Checker infrastructure, meaning that you can now monitor for exceptions, not just for your IBM products, but potentially for all products, from a single interface.

For all these reasons, we strongly encourage you to do the following:

- ▶ Run Health Checker on all systems in your environment.
- ▶ Run it all the time. Certain customers only run it once, when they move to a new release. However, using it in this manner means that you lose the entire benefit of Health Checker monitoring your system for changes of which you might be unaware.
- ▶ Aim to have all checks run with zero exceptions. If you get an exception, attempt to bring your system into line with the IBM recommendation. If that is not possible or not appropriate for this particular case, see whether you can modify the check so that it monitors for your specific value. If that is not possible either, then as a last resort, disable that health check. However, the most important point is that you do not get into a situation in which you ignore exceptions in Health Checker because “that thing is always issuing exceptions —just ignore it.”



CICS sysplex best practices

This chapter looks at IBM best practices for maximizing CICS availability in a Parallel Sysplex environment. It discusses the following:

- ▶ Eliminating affinities
- ▶ Effective workload balancing
- ▶ Connecting CICS to MQ and DB2
- ▶ Recovery and restart considerations
- ▶ General guidelines

3.1 Introduction

Enterprises have traditionally run multiple CICS regions, with separate regions dedicated to specific responsibilities (terminal owning region, application owning region, queue owning region, and so on). As a result, extending CICS to maximize availability in a Parallel Sysplex is really the next logical step in its evolution.

Similarly, regardless of whether you are running in a sysplex, there is a need to manage multiple CICS regions. The IBM-provided tool to help with this task is the CICSplex System Manager. Just as CICSplex System Manager can manage multiple CICS regions in a single z/OS image, it can manage CICS regions across one or more Parallel Sysplexes.

Before we look at what changes might be required to fully exploit the capabilities of Parallel Sysplex, let us briefly look at what benefits Parallel Sysplex can potentially deliver in a CICS environment:

Scalability	The concept of Parallel Sysplex is that any work should be able to run on any member of the sysplex. For CICS, this means that if you require additional CICS capacity, it should be possible to quickly and easily clone another CICS region, add it to any system in the sysplex, and work should automatically start flowing to that region.
Flexibility	<p>Conversely, if you want to reduce the number of CICS regions, it should be possible to simply stop a region, and all the work that was running in that region is now simply routed to one of the other remaining regions.</p> <p>You can also move CICS regions from one system in the sysplex to another (for example, if one CPC is upgraded to add more capacity). By easily being able to move regions, you can exploit the additional capacity on the upgraded CPC and free up capacity on the CPCs from which the regions are moved.</p>
Availability	<p>If there are multiple regions that deliver every used CICS function, it should be possible to stop any of those regions and still deliver the service to the user, as long as at least one of those regions is still available.</p> <p>Furthermore, if a CICS region experiences an unplanned outage, all the work that the region would otherwise have processed should automatically get routed to one of the other regions. The work that was running at the time of the failure obviously is impacted, but new work should be unaffected.</p>
Performance	If the CICSplex spans multiple systems in the sysplex, it should be possible to optimize performance by routing CICS transactions to the CICS region that can deliver the best response time to the user.

To fully achieve these benefits, there might be changes required to the applications. In the many years since CICS was initially written, many enhancements were introduced to help it deliver better performance. A number of these involved passing data between transactions using memory, rather than DASD. If you want the ability to run any CICS transaction in any region in any system in the sysplex (which maximizes your ability to achieve the benefits listed above), you need to analyze your transactions, identify any that exploit these in-memory techniques, and replace those techniques with ones that enable the cooperating transactions to run in different CICS regions.

3.2 Affinities

Fundamental to fully realizing the benefits of Parallel Sysplex for CICS is removing as many affinities as possible. The reason this is important is that you do not want any transaction to be tied to the availability of any single CICS region. You want to be able to maintain application availability across both planned and unplanned CICS region outages. By removing all affinities, and providing multiple regions with the same capabilities, you should be able to deliver service to all CICS transactions regardless of which CICS regions are up or down.

CICS transactions use many techniques to pass data from one to another. Certain techniques require that the transactions that are exchanging data execute in the same CICS region and therefore impose restrictions on the dynamic routing of transactions. If transactions exchange data in ways that impose such restrictions, there is said to be an affinity between them.

There are two categories of affinities:

- Inter-transaction affinity

An inter-transaction affinity is an affinity between two or more CICS transactions. It is caused by the transactions using techniques to pass information between one another or to synchronize activity between one another in a way that requires the transactions to execute in the same CICS region.

Inter-transaction affinities, which impose restrictions on the dynamic routing of transactions, can occur under the following circumstances:

- One transaction terminates and leaves *state data* in a place that a second transaction can access only by running in the same CICS region as the first transaction.
- One transaction creates data that a second transaction accesses while the first transaction is still running. For this to work safely, the first transaction usually waits on an event, which the second transaction posts when it has read the data that the first transaction created. This synchronization technique requires that both transactions are running in the same CICS region.

- Transaction-system affinity

A transaction-system affinity is an affinity between a transaction and a particular CICS region. It is caused by the transaction interrogating or changing the properties of the CICS region. Transactions with an affinity to a particular CICS region, rather than to another transaction, are not eligible for dynamic transaction routing. Typically, they are transactions that use CICS SPI commands, such as EXEC CICS INQUIRE or SET, or that depend on global user exit programs.

The restrictions on dynamic routing caused by transaction affinities depend on the duration and scope of the affinities. Clearly, the ideal situation is for there to be no transaction affinity at all, which means that there is no restriction in the choice of available target regions. However, even when transaction affinities do exist, there are limits to the scope of these affinities determined by the:

- Affinity relations: The most important are global, LUsername and user ID, as these determine how the dynamic routing program can select a target region for an instance of the transaction.
- Affinity lifetime: determines when the affinity is ended. These are classified as one of:
 - System
 - Permanent
 - Pseudoconversation
 - Logon
 - Signon

As far as possible, try to remove affinities from your systems. Affinities can stop you from being able to run instances of an application on multiple systems due to several of the required resources being unavailable. In a CICSplex environment, if all affinities are removed, it is then possible to keep the application running on the other regions in the CICSplex, thus masking any outage from the users. When examining the affinities that exist, it is important to not just look at a single application, but to consider all applications, as one might place a restriction on others.

CICS has delivered a number of alternatives to help you address existing affinities, many of which can be used without changes to your applications. The remainder of this section describes those functions.

3.2.1 Use of temporary storage pools

Temporary storage is the primary CICS facility for storing data that must be available to multiple transactions.

Data items in temporary storage are kept in queues whose names are assigned by the program storing the data. A temporary storage queue containing multiple items can be thought of as a small data set whose records can be addressed either sequentially or directly by item number. If a queue contains only a single item, it can be thought of as a named scratch-pad area.

Traditionally, temporary storage queues could be defined as non-recoverable or recoverable. Non-recoverable queues typically exist within the virtual storage of a CICS region. This provides better performance than having to do an I/O to DASD. However, if the CICS region goes down, the data on the queue is lost. Recoverable temporary storage queues must reside in a VSAM data set, so access to them is slower. Additionally, because all updates to the queue must be logged, the overhead of using recoverable queues is higher.

If the temporary storage data must be passed from a task in one region to a task in another region in an MRO scenario, a dedicated CICS region (a queue owning region) can be defined and specified to each CICS AOR that wants to use the queues located in that region.

While a QOR removes the affinity between the two transactions that are sharing data in the temporary storage queue, performance is not as good as a queue held within the same AOR as the transactions. The function shipping associated with communicating between the AOR and the QOR generates additional overhead, and the QOR constitutes a single point of failure. If the QOR fails, all data in the queues that it contains are lost.

A better alternative for sharing non-recoverable temporary storage queues is to place them in a CF structure. The location of the queue is transparent to the application. The location of the queue is defined on the DFHTST macro or via the RDO TSMODEL definition by the system programmer.

Note: Although the DFHTST macro is supported by CICS Transaction Server for z/OS, use resource definition online (RDO) to define the equivalent TSMODEL definitions.

The application continues to access the queue by the same name, but CICS dynamically determines where the queue is located and how to access it.

Because the queues are held in a Coupling Facility (CF) structure, the data on the queues is protected from CICS region outages. Even if all CICS regions are stopped, the data they had placed on the queues in the temporary storage structure is accessible when they are restarted.

Additionally, because CF access times are typically measured in microseconds, access to the data in the structure is far faster than would be the case with a QOR.

However, there is one availability consideration. Because the only queues that can reside in the temporary storage structure are non-recoverable, CICS does not, on its own, provide any mechanism to restore the structure contents to the point of failure if the structure, or the CF that it resides in, fails. If the availability of the data in the structure is critical to your applications, you can use System-Managed Duplexing to duplex the structure. But remember that the only queues that can reside in the temporary storage structure are non-recoverable ones, so by definition, applications using these types of queues should be designed to be able to handle the loss of that data.

CICS transactions running in an AOR access data in the temporary storage structure through a temporary storage server address space that supports a named pool of temporary storage queues. You must set up one temporary storage server address space in each z/OS image in the sysplex for each pool that is defined in the CF. All temporary storage pool access is performed by cross-memory calls to the temporary storage server for the named pool.

The name of the temporary storage pool that the server is going to support is specified on the POOLNAME parameter in the temporary storage server address space JCL (Example 3-1).

Example 3-1 Sample temporary storage server JCL

```
//DSTTSS  PROC POOL=#@$TOR1,MAXQ=1000,BUF=500
//DFHSEVR EXEC PGM=DFHXQMN,REGION=0M,TIME=NOLIMIT,
//  PARM=('POOLNAME=&POOL','MAXQUEUES=&MAXQ','BUFFERS=&BUF')
//STEPLIB DD DSN=CICST32C.CICS.SDFHAUTH,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN   DD DUMMY
```

You also need to specify the numbers of buffers to allocate for the server address space. To avoid the risk of buffer waits and to reduce the number of CF accesses, you can increase the minimum number of buffers from the default of 10 buffers per CICS region that can connect to the server. Providing a reasonable number of buffers will keep the most recently used queue index entries in storage. When a READ or WRITE request is completed, the queue index information is retained in the buffer. If the current version of a queue index entry is in storage at the time a queue item is read, the request requires only one CF access instead of two.

For information about the use of temporary storage and the considerations for avoiding transaction affinities, see *CICS Application Programming Guide*, SC34-7022. Information

about defining the shared temporary storage structure is available in *CICS Resource Definition Guide*, SC34-7000.

3.2.2 Named counters

There might be cases in which your applications have a requirement for obtaining a unique number for documents (for example, for customer orders or invoices numbers), or for obtaining a block of numbers for allocating customer record numbers in a customer file. In a single CICS region, there are various methods that you can use to control the allocation of a unique number. For example, you could use the CICS common work area (CWA) to store a number that is updated by each application program that uses the number. The problem with the CWA method is that the CWA is unique to the CICS address space and cannot be shared by other regions that are running the same application.

A CICS shared data table can be used to provide such a service, but the CICS regions all have to reside in the same z/OS image.

The CICS named counter facility overcomes all the sharing difficulties presented by other methods by maintaining its named counters in the CF, and providing access through a named counter server running in each system in the sysplex. This ensures that all CICS regions throughout the Parallel Sysplex have access to the same named counters.

In fact, the named counter facility even extends beyond CICS programs. The program that provides the interface to the named counter facility has a callable interface that can also be used by batch programs. This means that you can run batch jobs anywhere in the sysplex and have a high-performance mechanism for generating unique values, without having to create a mechanism of your own to provide this functionality.

Information about setting up and managing the named counter facility is provided in the chapter titled “Setting up and running a named counter server” in *CICS System Definition Guide*, SC34-6999.

Named counter recovery

Named counters are only stored in a structure in the CF, and updates to the counters are not logged. Applications using named counters might therefore need to implement recovery logic to handle the loss of the named counter structure.

Perhaps the easiest way to protect yourself from the loss of a named counter structure is to use System-Managed Duplexing for those structures. Typically, the access rate to the structure is not very high, so the cost of duplexing the requests is usually acceptable, particularly considering the impact if the structure were to be lost. Also, if the named counter structure is lost (and it is not duplexed), all the named counter server address spaces using that structure will abend.

In relation to the named counter server address spaces, there are a number of scenarios to be considered:

- **Loss of the named counter CF structure**

As stated above, if the named counter structure is lost, all the connected named counter server address spaces will abend. We suggest using the Automatic Restart Manager (ARM) to automatically restart the server address spaces on the same system on which they were running prior to the failure. The first server address space to start detects that its structure is not allocated and allocates a new, empty instance in an alternate CF. You must have a process that restores the structure contents before any applications start using the service again.

- **Abend of a named counter server address space**

The failure of a named counter server address space impacts any applications on that system that try to use the service. However, it does not have any impact on the contents of the named counter structure. In this case, we suggest using ARM to automatically restart the failed server address space on the same system on which it was running prior to the failure. No additional recovery is required.

- **Failure of a system containing a named counter server address space**

If the system containing a named counter server address space fails, we do not suggest using ARM to restart the address space on another system. There is no benefit from starting the address space on any other system, so simply let your normal system startup process start the address space as it would after any IPL.

It is prudent to have a tested mechanism to recreate the contents of the counters. For example, if you do not duplex the structure and the CF containing the structure fails, the contents are lost. Even if you do duplex the structure, an event such as a power failure can still result in the loss of all the counters contents.

If a named counter is only being used for temporary information that does not need to be preserved across a structure failure (for example, unique identifiers for active processes that are not expected to survive such a failure), then recovery techniques can be minimal. For example, you can recreate the counter with a standard initial value.

However, if a named counter is being used for persistent information, such as an order number, recreating it might require specific application logic to reconstruct the counter value.

Information about techniques that you can use to recover the counter values following a loss of the named counter structure are provided in the section titled “Name counter recovery” in *CICS Application Programming Guide*, SC34-7022.

3.2.3 CICS data tables

A CICS file is a representation of a data set on DASD. If you specify that the file is to use data table services, CICS copies the contents of the data set into an z/OS data space when the file is opened and uses that copy whenever possible.

The concept of shared data tables is simple. It exploits the fact that it is more efficient to:

- ▶ Use z/OS cross-memory services instead of CICS function shipping to share a file of data between two or more CICS regions in the same z/OS image.
- ▶ Access data from memory instead of from DASD.
- ▶ Access a file of data from memory using services integrated within CICS file management instead of using VSAM services and a local shared resource (LSR) pool.

There are basically two types of CICS data tables from the perspective of an application program:

- ▶ CICS-maintained tables (CMTs)

A CICS-maintained data table is a data table whose records are automatically reflected in the source data set. When you update the file, CICS changes both the source data set and the data table.

- ▶ User-maintained tables (UMTs)

A user-maintained data table is a data table whose records are *not* automatically reflected in the source data set. When you update the file, CICS changes only the data table.

Prior to CICS/ESA 3.3, the data in a given data table was only accessible to transactions running in the same CICS region that owned the data table. CICS/ESA 3.3 then added the shared data table capability. This provided the ability to access a CICS data table from multiple CICS regions in a single z/OS using high-performance cross-memory services. This delivers excellent performance. However, it creates an affinity between the programs that want to use that data table and the system that the data table resides on.

To provide the performance benefits of data tables while also providing the flexibility to access the data table from anywhere in the sysplex, CICS TS 1.3 added support for placing CICS data tables in a structure in the CF. These are called Coupling Facility Data Tables (CFDTs). They are similar in many ways to a shared user-maintained data table. A CF data table is different from a UMT in one important respect: initial loading from a VSAM source data set is optional.

One of the nice things about this implementation is that no application changes are required if you want to change a user-maintained data table to make it a Coupling Facility Data Table. The mapping of the location of the data table (in an z/OS data space or in a CF structure) is controlled by CICS, and application programs should be unaware of the location of the data table.

The availability considerations for the data table contents are a combination of protecting the CF data tables structure from a failure (you use System-Managed Duplexing, just as you do for the temporary storage structure and the named counter structure), and the normal considerations for protecting data in a user-maintained data table (that is, it is the user's responsibility to harden updates to the data tables back into the DASD data sets).

While the performance delivered by a data table in a CF structure is not quite as good as that possible with a data table residing in a data space, it is far better than can be delivered if the data had to be accessed from a data set on DASD or by function shipping requests between CICS regions.

Information about the use of shared data tables in general is available in *CICS Shared Data Tables Guide*, SC34-7107. For application considerations specifically relating to CF data tables, see the section titled "Coupling facility data tables" in *CICS Application Programming Guide*, SC34-7022. For information about defining and managing CF data tables, see the section titled "Coupling facility data tables" in *CICS System Definition Guide*, SC34-6999.

3.2.4 VSAM RLS

Accessing VSAM files concurrently from separate CICS regions with full data integrity was possible by implementing a file owning region (FOR). All access to the VSAM data set was through the FOR. The CICS transactions run in application owning regions (AORs) and the VSAM file requests are function shipped to the FOR. This implementation has several problems, such as:

- ▶ The FOR is a single point of failure.
- ▶ The FOR can be a bottleneck for CPU and throughput.
- ▶ Function shipping between AOR and FOR is additional overhead.

VSAM RLS is designed to allow VSAM data to be shared, with full update integrity, among many applications running in one or more MVS images in a Parallel Sysplex, without the need for a FOR in CICS. The best sources of information are the product manuals and the following IBM Redbooks publications, which are companions to this book:

- ▶ *CICS and VSAM Record Level Sharing: Planning Guide*, SG24-4765
- ▶ *CICS and VSAM Record Level Sharing: Implementation Guide*, SG24-4766

3.2.5 CICS ENQs

For many years, CICS has provided an ENQ/DEQ capability that can be used by CICS applications to serialize access to named resources. Originally, the scope of the serialization was within a single CICS region. If a transaction wanted exclusive access to a resource, it could issue the ENQ command, make whatever changes it wanted (with the knowledge that no one else could be doing anything with that resource), and then issue the DEQ to release serialization.

As long as all transactions that have an interest in that resource run in the same CICS region, then this is fine. However, you now have an affinity between all those transactions and that single CICS region. If a transaction in another CICS region were to try to change that resource, it would not be aware of the serialization held by the transaction in the original region.

To address this affinity, CICS now provides the ability for the system programmer to specify that serialization requests for selected resources be externalized to GRS and therefore treated as sysplex-wide resources. These resources are defined using ENQMODEL definitions.

If a resource has been defined using ENQMODEL to be a sysplex-wide resource, you can now run the transactions that have an interest in that resource in many CICS regions, and all are aware of any serialization resulting from a program issuing an ENQ request. The important point is that no application changes are required, as the control of whether the resource is to be treated as a local resource or a sysplex resource is at the CICS level.

For more information about the use of global ENQs and DEQs, see the section titled “Using ENQ and DEQ commands with ENQMODEL resource definitions” in *CICS Application Programming Guide*, SC34-7022. Information about setting up the ENQMODEL definitions is available in the chapter titled “ENQMODEL Resources” in *CICS Resource Definition Guide*, SC34-7000.

For more information about the impact of affinities in a CICSplex, see *Parallel Sysplex Application Considerations*, SG24-6523. For complete information about how to identify and remove affinities, see the IBM Redbooks document *IBM CICS Interdependency Analyzer*, SG24-6458. There is also helpful information in the chapter titled “Affinity” in *CICS Application Programming Guide*, SC34-7022.

3.3 Effective CICS workload balancing

Having successfully removed all affinities from your CICS applications, you are now able to run any transaction in any of your available CICS regions. However, just because you *are able* to do something does not necessarily mean that you are actually *doing* it. What we want is for the system to automatically route transactions to the most appropriate CICS region. Specifically, what we do *not* want is for anyone to have to take manual action to move transactions from one region to another to maintain application availability over the outage of a region. If we are going to mask outages (both planned and unplanned) from the users, the distribution of work across the available regions must be happening at machine speeds. This section discusses the capabilities in CICS for efficiently distributing transactions across the available CICS infrastructure.

3.3.1 CICS topology and session balancing

To maximize CICS availability, take a top-down approach. The first thing that needs to be addressed is providing multiple regions to accept the initial session connection request. For example, for traditional SNA users, provide multiple terminal owning regions (TORs) so that users can still log on if one TOR needs to be stopped. If possible, you want to have TORs on every CPC in the CICSplex, so that if one CPC is down, users can still access CICS via the TORs on the other CPC. Ideally, you will also have more than one TOR in each system, so that if one TOR is down, you do not end up with all the CICS sessions on other members of the sysplex, and none on the system where the TOR is down.

However, having multiple session-owning regions is not enough. You also want to have a mechanism where new logon requests are automatically balanced across the available regions. For SNA, this means that you use VTAM Generic Resources. For TCP/IP connections, use Sysplex Distributor. More information about balancing logon requests across the available regions is provided in *System z Dynamic Workload Balancing*, REDP-4563.

3.3.2 Dynamic transaction routing

The traditional CICS MRO model is that one set of CICS regions own the session with the user, and transactions started by the user are then sent to another region for execution. The decision about which target region to select is controlled by the dynamic transaction routing exit. You can write your own program to fulfil this role, or you can use the exit provided by CICSplex System Manager.

CICSplex System Manager workload management

CICSplex SM workload management selects the best system in your enterprise by dynamically routing transactions and programs to the CICS region that is the most appropriate at the time, taking into account any transaction affinities that exist.

Workload balancing makes the best use of the available CICS systems, and provides opportunities for increased throughput and performance by routing transactions or programs among a group of target regions according to the availability, performance, and load of those target regions.

CICSplex SM's workload management function uses a user-replaceable dynamic routing program, EYU9XLOP, to identify which target region runs a work request originating in a requesting region. (The region where the decision is made is called a routing region. The same region can be both the requesting and the routing region.)

The target region can be selected using one of two algorithms:

- ▶ The queue algorithm routes work requests to the target region that has the shortest queue of work, is least affected by conditions such as short-on-storage, and is the least likely to cause the transaction to abend.
- ▶ The goal algorithm routes work requests to the target region that is best able to meet the transaction's response time goal, as predefined in z/OS Workload Manager. Prior to CICS TS 4.1, CICSplex SM only supported average response time goals. However, CICS TS 4.1 adds the ability to also use percentile response time goals.

Using the queue algorithm

During workload processing using the queue algorithm, CICSplex SM routes all transactions and programs initiated in the requesting region to the most appropriate target region within the designated set of target regions using the following factors:

1. How heavy is the task load of the region?

This is calculated as a percentile value by dividing the target region's MAXTASKS setting by the current task count.

2. What is the region's health status?

This is calculated by assigning arithmetic weights depending on whether the region is short on storage, taking a transaction dump, taking a system dump, running at its MAXTASKS limit, or is in a stall condition.

3. What is the speed of the connection between the router and the target?

This is calculated by assigning arithmetic weights depending on whether the routing region is linked to the target via an MRO connection, an XCF connection, an LU6.2 connection, an IPIC connection, or if the target region is the router itself.

4. Are there any outstanding CICSplex SM Realtime Analysis (RTA) events associated with the workload?

This is calculated by assigning weights depending on the severity of the outstanding events. These are only factored in when an event name is specified in the CICSplex SM WLM specification for the workload, or in any transaction group definitions associated with it.

5. Are there any transaction affinities outstanding to override the dynamic routing decision?

Regardless of any of link factors described above, if the routing request has an outstanding affinity associated with it, then that affinity determines the routing target.

Using the goal algorithm

CICSplex SM also supports the goal algorithm. The aim of the goal algorithm is to select the target region that is best able to meet the defined response time goals for all work in a workload.

CICS obtains the goal for a transaction from the service class assigned to the transaction by the z/OS Workload Manager. Service classes are assigned to a transaction based on the transaction name, LU name, or user ID. CICSplex SM (starting with CICS TS 4.1) recognizes both average response time and percentile response time goals. If transactions are given velocity or discretionary goals, they are assumed to be meeting their goals.

Attention: If your workload is mainly made of a diverse mix of transactions and response time goals, then a region goal approach might work better.

z/OS Workload Manager: region and transaction goals

z/OS Workload Manager can manage CICS work toward a region goal or a transaction response time goal. You can choose which goal is used.

When you choose to manage toward a region goal, z/OS Workload Manager uses the goal for the service class assigned to the CICS address space under the JES or STC classification rules. When you choose to manage towards a transaction goal, z/OS Workload Manager uses the goals for the service classes assigned to transactions or groups of transactions under the CICS subsystem classification rules. You can select which mode to use when you are working with the JES or STC classification rules.

When you manage towards a transaction goal, z/OS Workload Manager does not directly manage resource allocations for the transactions. Instead, it makes calculations to assign appropriate resources to the CICS regions that can run the transactions. This can work less well if regions have a diverse mix of transactions and response time goals. In this situation, managing toward a region goal might work better.

Sometimes, the processing for a single work request requires more than one transaction in the CICS region. For example, up to four transactions, with different transaction identifiers, might be needed to process an inbound SOAP request in a CICS provider region. Take this into account when deciding whether to use a transaction goal or a region goal.

If you would like to be able to use RMF to report on transaction response times, but still prefer to manage the CICS work at the region level, it is possible to do this. The process for achieving this is documented in the section titled “11.5 Reporting” in *System Programmer's Guide to: Workload Manager*, SG24-6472.

Optimizing CICS workload routing

Prior to CICS TS 4.1, CICSplex SM in each system maintained real-time information about the regions that it managed on that system. However, the information was only exchanged with the other systems in intervals, meaning that the information being used to decide where to route a transaction might be out of date.

Starting with CICS TS 4.1, workload throughput is improved through a more efficient workload management function. CICS dynamic workload management now takes advantage of the Coupling Facility to store current region status information posted directly from CICS. This allows routing decisions to be made based on more up-to-date information, regardless of whether the routing and target regions are in the same z/OS system.

To enable this capability, every region in the CICSplex must be running CICS TS 4.1, each system must have a region server address space, and the regions must be defined as running in optimized mode via the CICSplex SM CSYSDEF view (specify WLM optimization enablement as ENABLED).

Other CICSplex SM suggestions

In a CICSplex where multiple regions are capable of running each transaction, you have the possibility of routing work away from regions that are experiencing a high percentage of abending transactions. An example might be an AOR whose DB2 subsystem has abended. Any DB2 transactions routed to that AOR will abend until DB2 is available again.

In a situation like this, the CICSplex SM ABENDTHRESH and ABENDCRIT parameters can be used to have transactions routed away from a region where there is a probability that they will fail. For more information about these parameters, see *CICS TS 4.1 CICSplex SM Administration*, SC34-7005.

3.4 CICS subsystem connectivity

As applications become more functionally rich (and more complex), it is becoming increasingly common to find that CICS interacts with other subsystems. To provide a high-availability environment, therefore, it is not sufficient that CICS on its own is configured to avoid single points of failure. This section provides suggestions for setting up CICS to maximize the availability of the connections between it and the other subsystems.

3.4.1 CICS-MQ connection

Support for WebSphere MQ queue-sharing group attach is provided starting in CICS TS V4.1. Prior to this release, it was necessary to specify the name of a specific MQ subsystem.

You can now specify a WebSphere MQ queue-sharing group name for the CICS-WebSphere MQ connection, so CICS uses any eligible queue manager in the group when it reconnects to WebSphere MQ, rather than waiting for a specific queue manager. Queue-sharing groups increase reliability when you reconnect to WebSphere MQ, and help you standardize this aspect of CICS setup across CICS regions and z/OS images.

Instead of defining default settings for the CICS-WebSphere MQ connection in the DFHMQPRM operand of an INITPARM system initialization parameter, you must now use the new MQCONN resource definition. You can use the MQCONN resource definition to specify a queue-sharing group name or the name of a specific queue manager.

If you have specified a queue-sharing group name for the connection, you can select appropriate resynchronization actions for CICS using the RESYNCMEMBER attribute of the MQCONN resource definition. Resynchronization works in the same way as it does for the group-attach function for DB2. Resynchronization takes place when the connection to WebSphere MQ is lost and CICS is holding outstanding units of work for the last queue manager. You can choose whether CICS waits to reconnect to the same queue manager, or whether CICS makes one attempt to reconnect to the same queue manager, but if that attempt fails, connects to a different eligible queue manager in the group.

A queue manager is eligible for connection to a CICS region if it is currently active on the same LPAR as the CICS region.

You can upgrade to use the new EXEC CICS and CEMT commands or CICSplex SM to start and stop the CICS-WebSphere MQ connection and change all the attributes of the connection. Alternatively, you can continue to use the existing methods of operating the CICS-WebSphere MQ adapter to initiate and manage connections between CICS and WebSphere MQ.

You can use the CKQC transaction from the CICS-WebSphere MQ adapter control panels, or call it from the CICS command line or from a CICS application.

3.4.2 CICS-DB2 connections

The DB2CONN definition contains the name of the DB2 subsystem or the group attach name of the DB2 data sharing group to which CICS should connect.

We suggest always using the group attach name, rather than a specific subsystem name. This provides the flexibility to run a CICS region on any system in the sysplex and to be able to connect to any DB2 in the data sharing group.

If CICS loses its connection to the DB2 that it is currently connected to, and it has INDOUBT units of work in progress with that DB2, it attempts to reconnect to that DB2. However, even if it is not successful in its initial reconnection attempt, it remembers that it has an outstanding unit of work with that DB2. If that CICS region is subsequently restarted (or if the DB2 connection is stopped and restarted) and that DB2 is available at that time, CICS automatically reconnects to it (as long as CICS is on the same z/OS system as the DB2 subsystem), even if there are multiple DB2 subsystems from the same data sharing group on that system.

How CICS reacts to the initial loss of connection to DB2 is controlled by the RESYNCMEMBER and STANDBYMODE keywords on the DB2CONN definition. CICS always makes one attempt to reconnect. However, if that fails, CICS either waits for that DB2 to become active again, or it attempts to connect to another member of the data-sharing group, if one exists on the same z/OS system as CICS.

If you have multiple DB2 subsystems in the same data sharing group on the same z/OS system, you should consider setting the RESYNCMEMBER keyword to NO. This means that the CICS region connects to another DB2 and is able to continue processing DB2 transactions. The INDOUBT units of work remain outstanding and are not be resolved until CICS reconnects to the original DB2. However, you can drain the CICS region at a convenient time, then stop and restart the DB2 connection, and the INDOUBT units of work are resolved at that point.

To stop the flow of DB2 transactions to this region during the time after it initially loses its connection to DB2, exploit the CICSplex SM ABENDTHRESH and ABENDCRIT function as described in “Other CICSplex SM suggestions” on page 92.

3.4.3 CICS and Automatic Restart Manager

The z/OS Automatic Restart Manager (ARM) provides the ability to automatically restart address spaces, either after an address space failure or a system failure. Always use ARM to restart a CICS region following a region failure.

Whether you use ARM to restart CICS on an alternate system following a system failure depends on whether you use ARM to restart any connected subsystems (DB2, for example) on another system. Assuming that you have a number of instances of each CICS region type on every member of the sysplex, it should not be necessary to start CICS regions on another image from the perspective of providing a transaction processing capability. However, if you restart a connected subsystem on another system, it might be necessary to also restart any CICS regions that were connected to that subsystem to resolve INDOUBT units of work. If you do not do this, locks associated with these units of work are not released, potentially impacting other transactions trying to use those resources.

ARM provides the ability to specify a group of address spaces that must all be restarted on the same system in case of a system failure. So, if you use ARM to restart DB2 elsewhere, include the CICS regions associated with that DB2 in the same group. Example 3-2 shows an example of the ARM policy statements.

Example 3-2 Sample ARM policy statements for CICS and DB2

```
DATA TYPE(ARM) REPORT(YES)
  DEFINE POLICY NAME(ARMPOL1) REPLACE(YES)

  RESTART_GROUP(DEFAULT)           /* FOR THOSE NOT SPECIFIED */
    TARGET_SYSTEM(*)               /* ON ALL SYSTEMS           */
    ELEMENT(*)                     /* ALL ELEMENTS             */
```

```

RESTART_ATTEMPTS(0)                /* DO NOT RESTART */
RESTART_GROUP(DB2ACC1)             /* GROUP FOR ACCEPTDB2 */
TARGET_SYSTEM(FK01,FK02)          /* ONLY ON THESE SYSTEMS */
ELEMENT(DFK1DA03)                 /* RESTART ONLY ONCE */
    RESTART_ATTEMPTS(1)           /* WHEN SYSTEM FAILURE */
    RESTART_METHOD(SYSTEM,STC,'_DA03 STA DB2,LIGHT(YES)')
ELEMENT(DFK1DA04)                 /* RESTART ONLY ONCE */
    RESTART_ATTEMPTS(1)           /* WHEN SYSTEM FAILURE */
    RESTART_METHOD(SYSTEM,STC,'_DA04 STA DB2,LIGHT(YES)')
ELEMENT(SYSCICS_AICICA21)
    RESTART_ATTEMPTS(1)
    RESTART_METHOD(SYSTEM,STC,'S FK21CICS,START=AUTO')
ELEMENT(SYSCICS_AICICA22)
    RESTART_ATTEMPTS(1)
    RESTART_METHOD(SYSTEM,STC,'S FK22CICS,START=AUTO')

```

3.5 General suggestions

This section contains information about ways to set up or manage other aspects of CICS that optimize your exploitation of the availability and scalability benefits afforded by running in a Parallel Sysplex.

3.5.1 CICS log-stream-related suggestions

CICS regions use a number of log streams. Every CICS region requires a DFHLOG and a DFHSHUNT log stream. In addition, depending on your applications, product mix, and the functions and tools that you use, regions might use a number of additional log streams. This section contains information to help make CICS's interaction with its log streams as efficient as possible. We also provide tips to maximize the availability of the data in the CICS log streams. Before proceeding, go back and review 2.5, "System Logger" on page 70, as the recommendations there generally apply to CICS and to other z/OS System Logger users. For a general understanding of z/OS System Logger and how it works, see the IBM Redbooks document *Systems Programmer's Guide to: z/OS System Logger*, SG24-6898.

The following guidelines are specifically for the DFHLOG and DFHSHUNT log streams, which tend to be the busiest and most important of the CICS-related log streams:

- ▶ Because the CICS DFHLOG and DFHSHUNT log streams are only accessed by a single CICS regions, you have the choice of defining the log streams as CF log streams or DASDONLY log streams.
- ▶ For CF log streams:
 - Place more than one, but less than 10 log streams in the same structure.
 - Try to ensure that each structure contains log streams that are connected to by CICS regions on more than one CPC, thereby enabling peer recovery if one of the CICS regions abends.
 - Do not mix DFHLOG and DFHSHUNT log streams in the same structure.
 - Try to place log streams with a similar level of activity in the same structure. For example, place busy log streams in one structure, and less busy ones in a different structure.

- For all log streams:
 - Size your staging data sets or z/OS System Logger structures (or both) so that no log records are moved from primary storage to the offload data sets.
 - Specify a reasonable LS_SIZE value. The default value of just two tracks is far too small.
 - Use a data class for the offload data sets that will result in a CI size of 24 KB. However, the staging data sets must have a CI size of 4 KB.
 - Specify OFFLOADRECALL(NO) for the DFHLOG and DFHSHUNT log streams.
 - Use model log stream definitions so that you do not have to explicitly specify new log stream definitions every time that you start a new CICS region.
 - If you use DASD mirroring for disaster recovery purposes, and you do not want to do a INITIAL start of your CICS regions in case of a disaster, define all your CICS log streams to use staging data sets.
 - A good empirical range for CICS system logstreams DFHLOG and DFHSHUNT, HIGHOFFLOAD should be set between 80 and 85%. The LOWOFFLOAD parameter value is most suitable between 40 and 60%. Too low a value might result in physical offloading of log data from primary to secondary storage after the z/OS System Logger offload process has completed physical deletion of any unwanted log data during offload processing. Conversely, too high a value might mean that subsequent offload processing occurs more frequently, as less space is freed up from primary storage during an offload operation.
 - The guidelines for general log streams like forward recovery logs and user journals are different from those for the system log. There is no requirement here to retain logged data in the coupling facility structure. Rather, due to the typical use of such data, you might only need a small structure to offload the data rapidly to DASD. If this is the case, set HIGHOFFLOAD between 80 and 85% and LOWOFFLOAD to 0.
 - It is generally preferable to have smaller structures, and to do frequent, small offloads, rather than large structures and the resulting infrequent, but long-running, offloads.
 - Do not enable auto alter for z/OS System Logger structures.
 - Keep the log streams for test CICS systems (and other systems not in regular use) in structures separate from the structures holding the log streams of production CICS systems.
 - Keep the log streams for terminal-owning regions (TORs) in structures separate from those accommodating log streams for application-owning regions (AORs). In addition, keep log streams for file-owning regions (FORs) in structures separate from those accommodating log streams for TORs and AORs.
 - Set MAXBUFSIZE to slightly less than 64 K (for example, 64000).

Log defer interval (LGDFINT) value

The LGDFINT CICS SIT parameter defines the log defer interval. The interval is the length of time, specified in milliseconds, that the CICS log manager is to wait before invoking the z/OS System Logger for a forced write request. The value chosen can have an impact on transaction response time.

Specifying a large LGDFINT value might increase the number of log records that CICS is able to place in a log block, and also decreases the cost of using z/OS System Logger (because z/OS System Logger is called less frequently). However, given the speed of more recent processors, it is better to send the log blocks to z/OS System Logger more frequently. For this reason, the default LGDFINT value has been decreased to 5 milliseconds. While some

benefit might be achieved from using an even smaller value, we do not advise using a value larger than 5.

Activity keypoint frequency (AKPFREQ) value

The AKPFREQ keyword specifies the number of write requests to the CICS system log stream output buffer required before CICS writes an activity keypoint. One of the consequences of taking an activity checkpoint is that log records that are no longer required for backout and recovery are deleted from the log stream, thus freeing up space in interim storage. A larger AKPFREQ value means that log records are retained in interim storage for longer.

The result of this is that either log blocks have to be moved to the offload data sets (which is not desirable) or the size of interim storage needs to be increased, resulting in additional storage requirements in the CF and in the z/OS System Logger data space. Another effect is that higher AKPFREQ can increase restart times.

Reducing the AKPFREQ value caused CICS to trim the tail of the log more frequently. Trimming the tail more often makes it possible for the offload process to reach the LOWOFFLOAD point without the overhead of issuing I/O to move data to the offload data sets.

3.5.2 CICS use of XCF groups

CICS Inter Region Communication (IRC) supports the use of XCF for inter-region communication between regions in the same sysplex, making it unnecessary to use VTAM for this communication. Each CICS region is assigned to an XCF group when it logs on to IRC, even if it is not currently connected to any regions in other z/OS images. When members of a CICS XCF group that are in separate z/OS images communicate, CICS selects XCF dynamically, overriding the access method specified on the connection resource definition.

Starting with CICS TS 3.2, you can specify the name of the XCF group on the XCFGROUP system initialization parameter. If you do not specify XCFGROUP, the region becomes a member of the default CICS XCF group, DFHIR000. If you run different types of CICS regions in the same sysplex (test and production or development and production, for example), you might want to use separate XCF groups for the separate collections of regions. You might also want to use multiple groups if you are approaching the limit of 2047 members in an XCF group *and* if you can break your CICS regions into groups of regions that do not need to communicate with each other.

If you do need to have multiple CICS XCF groups, follow these guidelines:

- ▶ Put your production regions in a separate XCF group from your development and test regions.
- ▶ Do not create more XCF groups than you need.
- ▶ Try not to move regions between XCF groups.

Note that CICS regions can only use XCF to communicate with other CICS regions in the same XCF group. Members of different XCF groups cannot communicate using MRO even if they are in the same z/OS image.

3.5.3 CICS naming conventions and cloning

To fully benefit from the ability to have pools of various types of CICS regions (AORs, TORs, and so on), you want to be able to:

- ▶ Easily and quickly define new CICS regions and bring them into service with minimal effort.
- ▶ Easily manage the CICS regions, while minimizing the opportunities for human error.

The key to these objectives is to have good naming conventions. An effective naming convention allows you to look at the name of a CICS region and immediately determine what type of region it is, which system it should normally run on, and which instance it is. *System z Platform Test Report for z/OS and Linux Virtual Servers*, SA22-7997, contains a section called “About our naming conventions,” which provides a summary of the naming convention used by the IBM test groups. This information might prove helpful when deciding on your own naming conventions.



DB2 sysplex best practices

This chapter provides recommendations for DB2 to deliver optimum performance, scalability, and availability in a Parallel Sysplex, data-sharing, workload-balancing, environment.

4.1 Introduction

DB2 is increasingly becoming the database manager of choice for enterprises large and small. One of the advantages of that success is that it attracts many application vendors to write products for a DB2 environment. However, for the DB2 system programmer and database administrator, this heavy usage and vital role brings challenges in delivering on the business's performance and availability requirements.

One of the cornerstones of DB2's ability to support highly available applications is its support of Parallel Sysplex and sysplex data sharing. This chapter pulls together into one place our latest thinking on the best practices for DB2 in a Parallel Sysplex environment.

In an attempt to keep the chapter short and concise, we do not spend a lot of time explaining the meaning of various parameter and settings. That information is typically available in the standard DB2 manuals, especially *DB2 V9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845. There is also invaluable information in the IBM Redbooks document *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322.

4.2 Coupling Facility-related recommendations

A Coupling Facility (CF) is a System z LPAR that runs Licensed Internal Code called Coupling Facility Control Code. Associated with the CF are z/OS-provided services that allow multiple DB2 subsystem instances running in the same or multiple z/OS systems to share the same data. The performance and availability of your DB2 configuration is dependent on optimizing all interactions between DB2 and the CF. This section contains the latest recommendations in this area.

4.2.1 Avoid single points of failure in CF configuration

Because of the importance of the CF to DB2 processing when in a data-sharing environment, it is critical that all connected systems have access to the CF at all times. To ensure this, have more than one CF so that if one needs to be stopped or experiences an unplanned outage, the affected structures can be moved to, or recovered in, the other CFs. For this reason:

- ▶ Always have at least two CFs.
- ▶ If possible, at least one, and preferably both, CFs must be failure-isolated from the systems that are connected to them.
If the CF containing the DB2 Lock and SCA structures is not failure-isolated from the connected DB2s, duplex those structures using System-Managed Duplexing.
- ▶ Every connected system should have at least two failure-isolated links to each CF.¹
- ▶ Ensure that the XCF_CF_STR_AVAILABILITY, XCF_CF_CONNECTIVITY, and XCF_CF_STR_PREFLIST health checks are enabled *and* that any exceptions that they raise are immediately investigated and addressed.

¹ Note that the IOSSPOF service does not support checking for single points of failure related to CF links.

4.2.2 Failure-isolated lock and SCA structures

DB2 uses three types of CF structures:

- ▶ Shared Communication Area (SCA)
- ▶ Lock
- ▶ Group buffer pools (GBPs)

These structures contain different types of information and therefore have different recovery considerations.

The information in the lock and SCA structures is kept in three places:

- ▶ In the structures themselves
- ▶ In the virtual storage of the connected address spaces
- ▶ In the DB2 logs (although retrieving the information from the logs takes longer than retrieving it from the virtual storage of the connected address spaces)

If a CF containing either the lock or SCA structure fails, the structure contents can be recreated using the in-storage information from all the connected address spaces. If one of the connected address spaces fails, the total sysplex-wide information is still available in the structure, so no structure recovery is required.

But what happens if a single failure (processor) impacts both a lock or SCA structure *and* a connected DB2? In this case, the in-storage information for one of the connected address spaces is now gone *and* the contents of the lock or SCA structure, or both, are also gone. This means that the remaining members of the data-sharing group no longer have access to information about what locks were held by the failed DB2. As a result, to protect the integrity of the DB2 data, all the surviving members of that DB2 data sharing group immediately abend. When the DB2 data-sharing group is restarted, if the impacted member is not included in the restart, the other members of the group read that member's logs and recreate the missing information in the associated structure.

The most likely scenario in which this situation will arise is if the lock or SCA structure (or both) is in a CF in the same processor as a connected DB2. Figure 4-1 shows a sample configuration. In that configuration, CF01 is failure-isolated from both DB2s, because it is in a separate CPC to ZOSA and ZOSB, but CF02 is in the same failure domain as ZOSB. If you have a configuration like this, with one failure-isolated CF and an ICF, place the lock and SCA structures in the failure-isolated CF (CF01 in this example). If you do not have a failure-isolated CF, protect the contents of these structures by duplexing these two structures across the two ICFs².

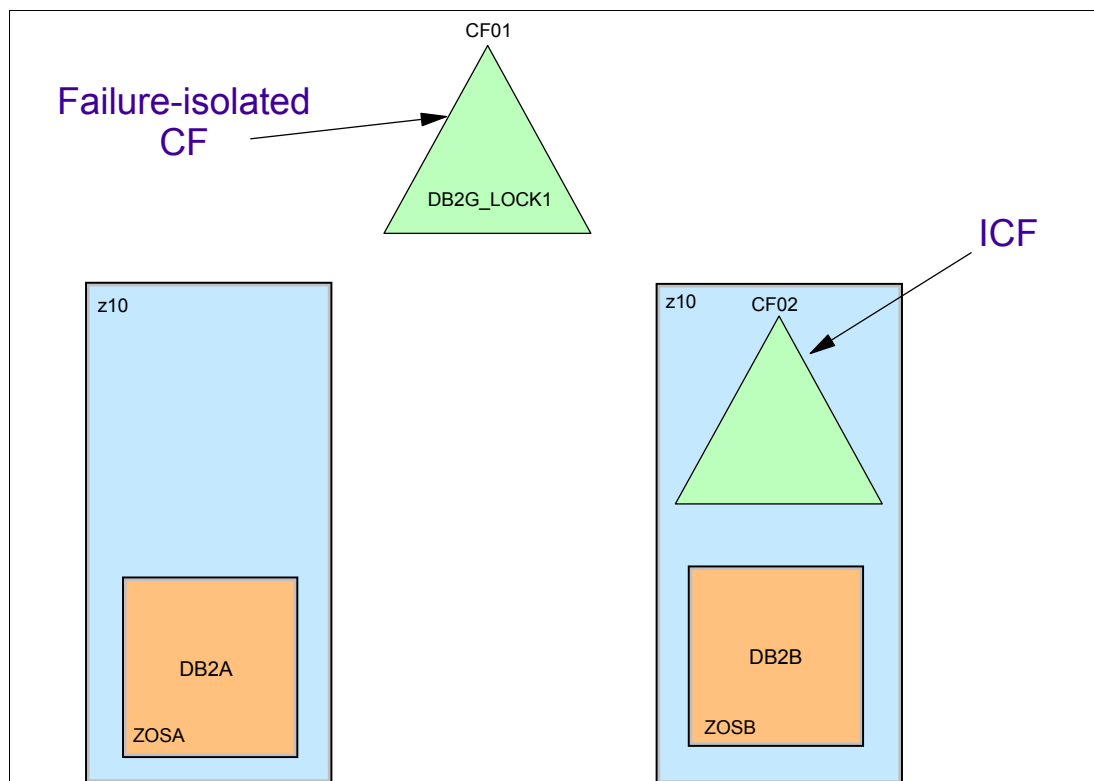


Figure 4-1 DB2 CF options

Note that this involves a tradeoff: using System-Managed Duplexing for these structures protects you from losing the structure contents in an ICF-only configuration. However, it elongates the response time for any duplexed request and increases the CPU utilization of both the CF and the connected z/OS systems. Conversely, not duplexing the lock or SCA structure (or both) might result in a lengthy recovery time if you lose the structure and one or more connected DB2s.

More information about System-Managed Duplexing can be found in a Whitepaper titled *System Managed Structure Duplexing*, available on the web at:

<ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/zsw01975usen/ZSW01975USEN.PDF>

4.2.3 Always use duplexing for DB2 GBP structures

The other DB2 CF structures that must be considered are the group buffer pools. Unlike the lock or SCA structures, the information that is used to recover a lost GBP structure does *not*

² In the case of a test or development sysplex with only two ICFs where you might not want to use duplexing, place the lock and SCA structures in the same CF to reduce the chance of a single CPC failure taking down the entire data-sharing group.

exist in the virtual storage of the connected DB2s. It exists in the log files of the those DB2s. The GBPs are also unlike the lock or SCA structure in that if you were to lose both a GBP structure *and* a connected DB2, the other members will *not* abend.

If a simplex GBP structure is lost, any updated pages belonging to tablespaces or indexspaces in the affected GBP that had not yet been castout to DASD are set to group recovery pending (GRECP) status. Each member that has pages in that GBP automatically issues internal START commands to perform GRECP recovery (assuming that you have taken the default action of AUTOREC for the GBP). The recovery is accomplished by reading the DB2 logs.³

However, the elapsed time for the recovery of the GBP contents is typically much longer than the time that is normally required to rebuild a lock or SCA structure. For one thing, the data must be read from DASD, rather than from memory. More importantly, the volume of data to be moved is typically much larger than is required to recreate the lock or SCA information, with recovery typically taking minutes (at least) compared to seconds for the lock or SCA structures. If the changed pages in a GBP need to be recovered, DB2 needs to read back through its logs until the last completed GBP checkpoint is reached. It then processes the logs forward from that point, identifying any pages that were updated but not yet written to the associated pageset.

For this reason, *every* production DB2 data-sharing group should duplex all production DB2 group buffer pool structures. The cost of turning on GBP duplexing is trivial compared to the savings this can deliver if you encounter a failure in a CF containing a simplex GBP structure. Remember that the CF storage used by the secondary structures should have been reserved as *white space* anyway, to allow for a failover of the simplex GBP structures. That is, you can have the storage sitting there empty, or you can pre-populate it with the data that is required to continue processing if the primary structure instance is lost.

GBP DUPLEX: Enabled or allowed

The CFRM policy specifies whether a GBP structure can be duplexed and, if so, whether it should be automatically duplexed as soon as it is allocated (DUPLEX(ENABLED)) or only duplexed when the operator issues a command (DUPLEX(ALLOWED)).

The advantage of using DUPLEX(ENABLED) is that the system takes responsibility to ensure that the structure is duplexed if the configuration supports it.

The advantage of using DUPLEX(ALLOWED) is that you have more control over when the structure gets duplexed, particularly after a CF outage.

On balance, we recommend that DUPLEX(ENABLED) is used, because you do not know when DB2 will allocate and deallocate its GBP structures. If all the tablespaces or indexspaces associated with a structure become non-GBP-dependent, the structure is deallocated. If you use DUPLEX(ALLOWED) and then use automation to re-duplex the structure every time that it is allocated, you might as well just specify DUPLEX(ENABLED).

Also ensure that the XCF_CF_STR_DUPLEX health check is enabled and that any exceptions that it raises are actioned in a timely manner.

³ If one or more members were impacted by the same failure that resulted in the loss of the GBP structure, and those members are not restarted, the other members of the data-sharing group can read their logs and perform the recovery on their behalf.

4.2.4 Enable auto alter for all DB2 CF structures

Tip: Always use the CFSizer tool (available on the web at <http://www.ibm.com/systems/support/z/cfsizer/>) to size all DB2 structures. This provides a good starting point that can then be fine tuned.

Auto alter is a z/OS function that automatically makes adjustments to a structure's space allocation when it detects that certain thresholds have been exceeded. The intent of auto alter is to reduce the pressure on the system programmer to identify and define exactly the right amount of storage for a given structure, and to maintain the correct value, as the use of the structure changes with workload. Still monitor the structure space usage and make adjustments as your workload changes. However, the day-to-day fine tuning can be handled by auto alter.

Difference between structure ALTER and structure REBUILD: There are two ways to change the size of a structure. One is via an ALTER of the structure, and the other is via a REBUILD. ALTER takes far less time to complete because it changes sizes, entries, and elements on the fly without rebuilding the structure. Conversely, a REBUILD quiets activity to the structure for the duration of the REBUILD. Therefore, exploiting the ALTER capability is the most effective way to adjust for changing structure storage needs.

For more information about REBUILD versus ALTER, see *z/OS MVS Sysplex Services Guide*, SA22-7617.

It is important to note that the process that triggers auto alter actions (structure full monitoring) is a timer-driven process. It is possible that a structure might fill up between iterations of the structure full monitoring routine that checks to see how full the structures are. As a result, enabling structure full monitoring and auto alter does not guarantee that you will never have a structure run out of storage. Nevertheless, enabling these functions certainly increases the chances of the system detecting a structure space problem and being able to react before this escalates into a performance problem or an outage.

We recommend that auto alter be enabled for all DB2 structures, but there are certain considerations:

- ▶ It is not possible to change the number of lock entries in a lock structure without rebuilding the structure. However, auto Alter *can* increase the number of record data entries without a rebuild.
- ▶ If the DB2 SCA structure fills up, all DB2 subsystems in the data sharing group abend. There are two ways in which this can occur:
 - The structure is too small for the configuration, and its use is slowly increasing over time, to the point that all the storage is eventually used. Auto alter can address this situation.
 - The used space in the structure increases dramatically within a very short period of time, as would happen if a DASD device containing many DB2 database data sets were to fail. Because of the speed with which this can happen, it is possible that the structure will fill before auto alter has an opportunity to increase the structure size.

To give you perspective, at the time of writing, we have never experienced a customer having a problem with the structure filling if the size is set to 128 MB.

- The impact on DB2 of a GBP structure running out of space depends on which part of the structure is affected:
 - If the structure runs out of directory entries, the CF might need to reuse directory entries that are currently being used to track DB2 interest in database pages. This situation is known as a cross-invalidation due to directory reclaims. This is not an ideal situation because it forces DB2 to invalidate a buffer that contained valid data. However, the only impact is a small performance hit.
 - If the structure runs out of elements (that is, all the elements contain pages that have not yet been cast out to DASD), DB2 might be unable to write a changed page to the GBP. You can determine whether this is happening by issuing a DB2 **-DISPLAY GROUPBUFFERPOOL(GBPn) GDETAIL(*)** command, or by looking at the Write failed-no storage field in a DB2 PE buffer pool activity report. Also monitor for DSNB319A messages from DB2, indicating that a structure is running low on storage. If the shortage is not addressed, message DSNB325A is issued, indicating that the structure has reached 90% full.

If DB2 is unable to write an updated page to the GBP, the affected DB2 member initiates a castout and retries the write several times. If the page still cannot be written to the GBP, and assuming that the page being written to the GBP is a changed page, it is added to the logical page list (LPL) requiring recovery, which is performed automatically by DB2 in most cases.

The auto alter function can potentially address both of these situations by dynamically altering the entry-to-element ratio or by increasing the current structure size.

On a longer-term basis, you might want to reduce the CLASST value to address DSNB319A and DSNB325A messages. You can also try reducing the GBPOOLT value. However, reducing the CLASST value is considered to be a more effective option.

For the latest guidance on the use of auto alter with DB2 GBPs, see an excellent presentation titled “Auto Alter for Couch Potatoes,” available on the web at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS1956>

There is one thing we want like to stress in relation to automatic functions provided by the system. Just because the system is able to make adjustments for you, it does not mean that you no longer have responsibility for actively managing your system. Specifically, if the system makes an adjustment (via auto alter, for example), be aware of the change, and reflect that change back into your CFRM policy. Do not forget that if you increase the INITSIZE value to reflect the auto alter change, you must adjust the SIZE value accordingly as well to maintain the ratio between the two values.

Automation reminder: There are certain messages that are issued by the system to inform you about activity or potential problems in the system. You must ensure that these messages are added to your automation product message table to ensure that the messages are brought to someone’s attention.

If structure full monitoring encounters a structure that is over the FULLTHRESHOLD value, it issues message IXC585E, naming the structure and displaying the current defined and used entries and elements.

If the structure is enabled for auto alter and auto alter decides that it wants to make a change to a structure, it issues message IXC588I, naming the structure and listing the current and target number of entries and elements and the total structure size.

While we are on the subject of usage of storage in the GBP structures, this is a good time to summarize the DB2 parameters that have a relationship to the use of storage in the GBPs:

GBPOOLT	This threshold determines the percent of storage in the group buffer pool structure that can contain changed pages before a castout is initiated.
CLASST	The space in a GBP structure is broken into a number of castout class queues (determined by DB2). CLASST specifies the threshold at which castout for that class will be triggered. For example, if CLASST is set to 5%, a castout is initiated for a particular pageset or partition when the changed pages for the associated class account is 5% of the total GBP pages.
DWQT	The deferred write threshold. This is the percentage of the local buffer pool that can be occupied by unavailable pages, including both updated pages and in-use pages.
VDWQT	The vertical deferred write threshold. This threshold is similar to the deferred write threshold, but it applies to the number of updated pages for a single pageset in the buffer pool.

The point at which auto alter takes action is controlled by the FULLTHRESHOLD value in the CFRM policy. The default is 80%, but you are able to change it to be a higher or lower value. Do not set it higher than 80% unless the GBP is large (greater than 1 GB). Setting FULLTHRESHOLD too high increases the likelihood of the structure filling up before auto alter has a chance to take any action.

You also need to consider the relationship between the GBPOOLT and FULLTHRESHOLD values. You do not want the two values to be so close that auto alter kicks in before GBPOOLT causes a castout (which reduces the number of changed pages in the GBP). Use the default FULLTHRESHOLD value (80%) and specify GBPOOLT to be not larger than the default of 30%. Specifically, GBPOOLT must be low enough that a sudden spurt in pages and temporary lag in castout processing does not result in the structure reaching the FULLTHRESHOLD value.

Set the buffer pool deferred write thresholds (DWQT and VDWQT) and the GBPOOLT threshold to ensure a trickle of I/O requests rather than short bursts of intensive I/O activity. This minimizes the impact on the I/O subsystem and helps reduce latency in asynchronous replication. I/Os for asynchronous database writes can contain up to 128 KB of data, so attempt to set your thresholds so that any castouts result in at least that much data being moved.

In most cases, set the CLASST value somewhere between 2 and 5%. If there is little or no reuse of data in the GBP, you might even use a value as low as 1% (keeping in mind that you want to move at least 128 KB on each castout). If there is a lot of data reuse from the GBP, aim for closer to the 5% end of the scale.

Specify structure SIZE to be 50 to 100% larger than INITSIZE

Note that for auto alter to be able to increase the structure size, the SIZE value for the structure (in the CFRM policy) must be larger than the structure's current size. It is difficult to provide a ratio between SIZE and INITSIZE values that applies to all cases. Use CFSizer to get an appropriate INITSIZE for the structure. (When asked by CFSizer, use the heavy data sharing option.) Then specify a SIZE value that allows for a reasonable amount of growth. By default, the SIZE value provided by CFSizer is based on a workload that is double the values that you provided to CFSizer to size the structure. In general, expect to see SIZE being 50 to 100% larger than INITSIZE.

Do not specify SIZE to be larger than twice INITSIZE. When a structure is allocated, control block space large enough to manage a structure that is the maximum size (as defined by the SIZE value) is reserved. The larger the ratio between INITSIZE and SIZE, the more space is reserved, reducing the amount of storage available for entries and elements. Also, assuming that you monitor changes to the structure's sizes, it is unlikely that a structure's storage requirement will more than double in less time than it takes you to update the CFRM policy, start the new policy, and rebuild the structure.

Set MINSIZE=INITSIZE for critical GBPs

In addition to altering the ratio between entries and elements and increasing the size of a structure, auto alter also has the ability to take storage away from structures that it determines have more than they need. This processing is only kicked off if the overall storage utilization in the CF exceeds 90%.

If you have especially critical GBP structures that should not have storage stolen from them, you can protect those structures by specifying a MINSIZE value that is equal to the INITSIZE value in the CFRM policy. If you enable auto alter for a structure and do not specify a MINSIZE, the default MINSIZE is 75% of the INITSIZE value.

4.2.5 Ensure that storage for the SCA structure is generous

As stated in 4.2.4, "Enable auto alter for all DB2 CF structures" on page 104, all members of the DB2 sharing groupabend if the SCA structure fills up. For this reason, always err on the generous side when sizing your SCA structure.

Because the SCA tends to contain very little information most of the time, but needs to be able to hold a large amount of information if there is a failure, it is difficult for CFSizer to provide an accurate recommendation about structure sizes. For this reason, define the SCA structure with an INITSIZE of 64000 and a SIZE of 128000. We are not aware of any customers having a problem with the SCA filling up when using these values.

4.2.6 Aim for zero cross-invalidations due to directory reclaims

As mentioned in 4.2.4, "Enable auto alter for all DB2 CF structures" on page 104, a lack of entries in a GBP structure can potentially result in the CF having to reuse directory entries. If there are no unused directory entries, the CF attempts to use one related to a page that none of the DB2s have an interest in. If no such entries exist, the CF must reuse a directory entry that is being used to track interest in a piece of data that is in the local buffer pool of at least one member of the data-sharing group. In the latter case, it has no choice but to do a cross-invalidate to tell all the interested DB2s to invalidate their local copy of that page.

The result of the directory reclaim is that if a DB2 needs to use that piece of data again, it must re-read it from the database on DASD. This results in poorer performance for the user of that data because it requires an additional CF access and a synchronous read to the database on DASD.

There are two ways to address this situation:

- ▶ Increase the size of the structure (via the INITSIZE parameter in the CFRM policy) so that it has more entries and elements.
- ▶ Have the DB2 DBAs change the ratio between entries and elements. You can achieve this by adjusting the entry-to-element ratio for the structure or by enabling auto alter for the structure. Note that it is *not* possible to adjust the entry-to-element ratio using the SETXCF START,ALTER MVS command. You can alter the entry-to-element ratio by using the DB2

-ALTER DBPOOL(....) RATIO(xx) command. However, this requires a subsequent rebuild or reallocation of the structure to effect the change.

The way to determine whether directory reclaims are causing valid pages to be cross-invalidated is to review an RMF structure summary report (Figure 4-2). The right-most column contains the count of the number of directory reclaims and the number of reclaims that caused a cross-invalidate.

1

COUPLING FACILITY ACTIVITY													PAGE 1
z/OS V1R8			SYSPLEX FPKPLEX			DATE 06/03/2009			INTERVAL 010.00.000				
			CONVERTED TO z/OS V1R9 RMF			TIME 10.50.00			CYCLE 01.000 SECONDS				

COUPLING FACILITY NAME = CFFK00													
TOTAL SAMPLES (AVG) = 599 (MAX) = 599 (MIN) = 598													

COUPLING FACILITY USAGE SUMMARY													

STRUCTURE SUMMARY													

TYPE	STRUCTURE NAME	STATUS CHG	ALLOC SIZE	% OF CF STOR	# REQ	% OF ALL REQ	% OF CF UTIL	AVG REQ/ SEC	LST/DIR ENTRIES TOT/CUR	DATA ELEMENTS TOT/CUR	LOCK ENTRIES TOT/CUR	DIR REC/ DIR REC XI'S	
CACHE	DSND8QU_GBP0	ACTIVE SEC	74M	0.2	342	0.0	0.0	0.57	63K 1947	13K 1388	N/A N/A	0 0	
	DSND8QU_GBP1	ACTIVE SEC	499M	1.5	32335	0.2	0.9	53.89	436K 112K	87K 4663	N/A N/A	0 0	
	DSND8QU_GBP11	ACTIVE SEC	250M	0.8	129	0.0	0.0	0.21	218K 0	44K 0	N/A N/A	0 0	
	DSND8QU_GBP12	ACTIVE SEC	250M	0.8	171	0.0	0.0	0.28	218K 32	44K 8	N/A N/A	0 0	
	DSND8QU_GBP13	ACTIVE SEC	245M	0.7	1811	0.0	0.0	3.02	57K 57K	57K 163	N/A N/A	463 463	
	DSND8QU_GBP14	ACTIVE SEC	245M	0.7	196	0.0	0.0	0.33	214K 545	43K 153	N/A N/A	0 0	
	DSND8QU_GBP16K0	ACTIVE SEC	49M	0.1	138	0.0	0.0	0.23	13K 0	10K 0	N/A N/A	0 0	
	DSND8QU_GBP2	ACTIVE PRIM	494M	1.5	1899	0.0	0.0	3.16	432K 9862	86K 1182	N/A N/A	0 0	
	DSND8QU_GBP32K	ACTIVE PRIM	74M	0.2	136	0.0	0.0	0.23	10K 1	17K 0	N/A N/A	0 0	
	DSNPSTG_GBP0	ACTIVE SEC	74M	0.2	1251	0.0	0.0	2.08	63K 5991	13K 4926	N/A N/A	0 0	
	DSNPSTG_GBP1	ACTIVE SEC	733M	2.2	1807K	9.5	21.0	3011.3	642K 596K	128K 127K	N/A N/A	796 0	
	DSNPSTG_GBP11	ACTIVE SEC	250M	0.8	17535	0.1	0.2	29.22	218K 1831	44K 1691	N/A N/A	0 0	
	DSNPSTG_GBP12	ACTIVE PRIM	250M	0.8	55366	0.3	0.6	92.28	218K 54K	44K 16K	N/A N/A	0 0	
	DSNPSTG_GBP13	ACTIVE PRIM	250M	0.8	121336	0.6	2.6	202.23	218K 214K	44K 16K	N/A N/A	54K 61K	

Figure 4-2 RMF structure summary report

4.2.7 Recommendations on GBP checkpoint frequency

If a GBP is damaged, all changed data in that structure that belongs to GBP-dependent pagesets must be recovered to the pagesets from the DB2 logs. The number of log records that need to be processed is determined by the frequency of the GBP checkpoint.

Part of the GBP checkpoint process includes the writing of all changed pages in the GBP to the pageset. The purpose of the checkpoint is to reduce the amount of time needed to recover data in a GBP. If you specify too small a value, you might end up moving pages that are about to be updated out to DASD, resulting in the same page being written to DASD multiple times, and if you specify too large a value, the time to recover after a GBP failure might be increased.

Use the default GBP checkpoint value of 4 minutes. While the GBP checkpoint process will move all updated pages to DASD, it is preferable if the majority of castouts are triggered by the GBPOOLT or CLASST thresholds (described in 4.2.8, “Recommendations on PCLOSET and PCLOSEN” on page 109) rather than by a GBP checkpoint. The reason for this is that there is less performance impact on the GBP structure if there is a constant trickle of pages being moved to the pagesets, rather than a large burst of activity every few minutes. Note, however, that the only way to determine what triggered the castout of a given page is to enable and analyze the IFCID 261, 262, and 263 trace records.

Note that each GBP has its own checkpoint value, and that the only way to change the default is via an ALTER GROUPBUFFERPOOL command. The new value specified on the ALTER command is saved in the BSDS, so even if the GBP is deallocated, the changed value takes effect again the next time that the structure is allocated.

It is also important to mention at this point that GBP recovery should not be as large a concern if you duplex the GBP structures. With duplexed structures, no recovery is required if either of the structure instances are lost. Note, however, that if you were to lose *both* instances (following a power failure, for example), the structure contents still need to be recovered from the DB2 logs, so still specify a reasonable GBP checkpoint value.

4.2.8 Recommendations on PCLOSET and PCLOSEN

Another set of parameters that are related to pagesets going in and out of GBP-dependency are PCLOSET and PCLOSEN. PCLOSET specifies the number of minutes since a pageset or partition was last updated, after which DB2 converts the pageset or partition from read-write to read-only. PCLOSEN specifies the number of consecutive DB2 checkpoints since a pageset or partition was last updated, after which DB2 converts it from read-write to read-only.

If DB2 has a pageset open for read-only processing, it has to send less information to the CF than if the pageset is open for read-write processing. Therefore, transitioning a pageset from read-write to read-only can reduce the cost for a job that is reading the pageset. Conversely, if you set the PCLOSET and PCLOSEN values too low, you can incur additional overhead as the pageset constantly transitions back and forth between read-write and read-only access.

Our recommendation is to look at the “Data sets converted from R/W → R/O” field in the DB2 PE statistics long report and use the default until you see more than 10 to 20 pagesets per minute converted to read-only.

Note that converting a pageset from read/write to read-only does not necessarily make the pageset not GBP-dependent. If another DB2 is using the pageset read/write, and this DB2 converts it to read-only, both DB2s must still propagate lock information to the CF, and the DB2 that is using the pageset read/write must still send all updated pages to the GBP. However, if the pageset is not touched by this DB2 for another PCLOSEN or PCLOSET interval, DB2 then physically closes the pageset, removing all interest in that pageset from this DB2. If there were only two DB2s using the pageset, and one physically closes the pageset, that results in the pageset becoming not GBP-dependent. The previous sentence is true if the tablespace is specified as CLOSE YES (on the CREATE/ALTER TABLESPACE statement). However, if the tablespace is specified as CLOSE NO, the data set remains GBP-dependent, even if there is no activity on the other member.

DB2 -ACCESS command

One of the options for the -ACCESS command that DB2 V9 added is the ability to remove GBP dependency from a tablespace or set of tablespaces. If you have large update-intensive batch jobs, or large utility jobs, you might investigate adding a first step to the job that will

issue the -ACCESS command to remove the GBP-dependency from the pagesets that the job will use. The -ACCESS command performs the following steps:

1. It attempts to drain all readers and writers on all members other than that on which the command is entered.
2. If the drain is successful, then the writers on the system on which the command was entered will be drained, assuming that the object is not already pseudo closed (in read-only status), and the member on which the command was issued converts the object to non-GBP dependent, including the castout of any changed pages.

If the drain was not successful (for example, if the tablespace is actively being used by a job on another system), the net effect is the same as though the -ACCESS command had never been issued.

Removing the GBP dependency should improve the elapsed times of batch jobs on that system that are using those tablespaces.

If there are DB2 batch jobs that you want to run on a specific system, investigate the use of WLM Scheduling Environments, perhaps in combination with the -ACCESS command. Scheduling Environments provide a way to very easy to set up and manage the control of which systems can run a given job. Also, a relatively simple piece of automation can be created that controls which systems a given Scheduling Environment is made available on, based on criteria such as whether DB2 is up or down, the location of a particular CF structure, which system is running on the fastest processor, or any number of other conditions.

4.2.9 Synchronized SYNCVAL value

The SYNCVAL parameter in DSNZPARM controls whether DB2 statistics recording is to be synchronized with a part of the hour. Every member of the data sharing group should have the same SYNCVAL value. Additionally, to make it easier to use a combination of RMF and your DB2 reporting tool to analyze DB2 activity, synchronize the RMF interval (specified on the SYNC keyword in the ERBRMFxx member) with the DB2 SYNCVAL value.

A related DSNZPARM parameter is STATIME. STATIME controls the interval (in minutes) between writing statistics trace records. Set STATIME to 1 if you are investigating a problem or want to undertake a tuning exercise. The cost is almost zero and the ability to see what the system is doing each minute can be invaluable. Even if you set this value as low as 1, there are still only 1440 SMF Type 100 records created per day (1 per minute).

4.2.10 Best practices for GBPCACHE attribute

There are a number of options that control what information is placed in the GBPs. The section titled “How the GBPCACHE option affects write operations” in *DB2 V9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845 provides information about the various options and which ones are the most appropriate for particular situations. But, in general, the recommendation is to use CHANGED for most GBPs, and CHANGED or SYSTEM for LOB tables.

4.2.11 Tablespace to buffer pool relationship

When defining a DB2 table, the DBA decides which local buffer pool to use to cache pages from that table. This decision also controls which GBP structure is used to cache pages from that table, as there is a one-to-one correspondence between local buffer pools and GBP structures.

Traditionally, the decision of which was the best buffer pool to use was driven primarily by considerations about the most effective way to use the local buffer pool. However, as customers implement ever-larger DB2 data sharing groups, it is prudent to also keep in mind effective use of the GBP in the Coupling Facility.

There are many things that come into play when deciding on the most appropriate buffer pool for a given table, so it is difficult to give advice that always applies. However, there are certain considerations to keep in mind that might help improve the effectiveness of your decisions.

The traditional approach has been to separate your data across your local buffer pools using one of the following methods:

- ▶ Based on access pattern and object type. That is, do not mix TS and IX tables. Objects that are primarily processed sequentially are placed in one pool, and randomly accessed objects (TS/IX) in another pool (or set of pools). There is a dedicated pool (BP0) for the DB2 catalog and directory, and a dedicated pool for DB2 RDS sort activity. This approach is common.
- ▶ Based on application, or set of applications. Using this method, all tables belonging to application A are placed in one buffer pool, those belonging to application B go in another buffer pool, and so on.
- ▶ A combination of the above.

There are valid reasons for having a small number of large pools⁴:

- ▶ They are easier to manage.
- ▶ They provide more efficient use of local buffer pool resources. If you have a dedicated pool with very low or intermittent usage, you can waste a lot of storage both in z/OS and in the CF.
- ▶ There is a better chance of caching more pages if the object is very active (the DB2 algorithm keeps more of your pages in the buffer pool if it is big and heavily used).

Conversely, large local buffer pools in a data sharing environment can have a negative performance impact due to the long scans when going into and out of GBP dependency to register and de-register pages in the GBP⁵.

However, the move to extremely large DB2 data sharing groups (either in terms of the number of members in the group or the volume of work being processed) means that there are certain cases in which you might want to do more separation based on the characteristics of the CF and how GBP structures work:

- ▶ There are certain situations in which very large GBPs do not deliver optimum performance:
 - If there are massive updates against an object in a very short time and the object is GBP dependent, this can flood the GBP with updated pages (resulting in a GBP-full condition and potentially pages being marked as GRECP). Because castout operates at millisecond speed (gated by the performance of the DASD containing the pagesets) and the writes to the CF are at microsecond speeds, there is no way that castout can keep up with the influx of pages.

It might make sense to place these objects in their own GBP so that they do not affect other applications, and set low GBPOOLT and CLASST thresholds for this GBP.
 - This phenomenon can also occur with utilities where objects are GBP dependent.

⁴ In the past, there were inefficiencies in DB2 related to managing large local buffer pools. However, those have been fixed.

⁵ One of the enhancements in DB2 V10 is a reduction in the time required to perform these scans.

Sometimes it helps to partition the object more (create more partitions or partition the index). This means that the updates are spread across more objects, giving the castout process a better chance to keep up.

- ▶ Putting everything in a small number of GBP structures can result in long chains of requests in the CF, which can elongate processing of the CF requests.
- ▶ Just as z/OS has locks and latches, similarly, certain processes within a CF structure are also protected by latches. If you have a large number of concurrent similar requests to a single structure, the requests might be delayed due to contention on one or more latches. If you were able to spread the requests over more structures, there would be a reduced likelihood of latch contention, resulting in a reduced elapsed time for the requests.

The CF code is constantly evolving and being enhanced to better handle ever-larger workloads. Nevertheless, it still makes sense to bear the characteristics of CF behavior in mind when deciding on how to use buffer pools.

4.3 Restart considerations

In a non-data sharing DB2 environment, when a DB2 is stopped (or fails), all the data under DB2's control is unavailable until the DB2 comes back. Conversely, one of the design points of a DB2 data sharing group is that the DB2 data remain available as long as at least one member of the data-sharing group is available.

If one of the members of the data-sharing group fails, it is almost certain that the DB2 is holding locks on DB2 objects when it fails. Until that DB2 is restarted and releases those locks, the related objects are not available to the other members of the sysplex.

4.3.1 Use ARM to restart DB2 as quickly as possible after a failure

There are two scenarios relating to a DB2 failure:

- ▶ DB2 failed.
- ▶ The system that DB2 was running on failed.

In either case, the most important thing is to get the failed DB2 back up and running as quickly as possible. The best way to achieve this is to use the MVS Automatic Restart Manager (ARM). Many automation products provide support for ARM. This means that they manage DB2 for normal startup, shutdown, monitoring, and so on. However, if DB2 fails, they understand that they must allow ARM to take responsibility for restarting DB2.

If the failure was just of DB2, and the system it was running on is still available, restart DB2 in the same system, with a normal start. DB2 automatically releases any retained locks as part of the restart.

If the failure affected the *system* that DB2 was running on, start DB2 on another member of the sysplex as quickly as possible. The reason for this is that it results in DB2 coming up and cleaning up its locks far faster than it is able to do were you to wait for z/OS to be IPLed and brought back up.

Furthermore, if DB2 is started on another system in the sysplex, you really only want it to release any locks that it was holding. More than likely, there is another member of the data-sharing group already running on that system. If you specify the LIGHT(YES) option on the START DB2 command, DB2 starts with the sole purpose of cleaning up its locks. In this mode, it only communicates with address spaces that it was connected to before the failure,

and that have indoubt units of work outstanding⁶. As soon as DB2 completes its cleanup, the address space automatically shuts itself down⁷. Hopefully, the failed system is on its way back up at this time, and the DB2 can be brought up with a normal start in its normal location.

In addition to restarting DB2 using ARM and Restart Light, also define a restart group to ARM so that it also restarts any subsystems that were connected to DB2 prior to the failure. By restarting all the connected subsystems, any indoubt units of recovery can be cleaned up.

Note that when the Restart Light capability was introduced by DB2 V7, it did not handle cleanup for any INDOUBT units of work. However, in DB2 V8 the Restart Light capability was enhanced so that it cleans up any INDOUBT units of work, assuming that the associated address space is also restarted on the same system. If you do not want to have DB2 resolve the INDOUBT units of work, or if you do not plan to restart the connected address spaces on the other system, start DB2 with the NOINDOUBT option.

Recommendation: Use ARM to restart DB2 following a DB2 failure.

If only DB2 failed, ARM must do a normal restart for DB2 on the same z/OS system. Never do a RESTART LIGHT of DB2 on the same system on which it was running previously.

If the system failed, ARM must do a Restart Light for DB2 on another system in the sysplex. Also, define a restart group so that ARM can also restart any related subsystems together with the restarted DB2.

The ARM policy must specify which address spaces to be started together in case of a system failure. Example 4-1 shows an example of such a policy.

Example 4-1 Sample ARM policy

```

DATA TYPE(ARM) REPORT(YES)
  DEFINE POLICY NAME(ARMPOL1) REPLACE(YES)

  RESTART_GROUP(DEFAULT)                /* FOR THOSE NOT SPECIFIED */
    TARGET_SYSTEM(*)                    /* ON ALL SYSTEMS           */
    ELEMENT(*)                          /* ALL ELEMENTS             */
    RESTART_ATTEMPTS(0)                 /* DO NOT RESTART           */

  RESTART_GROUP(DB2ACC1)                /* GROUP FOR ACCEPTDB2      */
    TARGET_SYSTEM(FK01,FK02)            /* ONLY ON THESE SYSTEMS    */
    ELEMENT(DFK1DA03)                  /* RESTART ONLY ONCE        */
    RESTART_ATTEMPTS(1)                /* WHEN SYSTEM FAILURE      */
    RESTART_METHOD(SYSTEM,STC,'_DA03 STA DB2,LIGHT(YES)')
    ELEMENT(DFK1DA04)                  /* RESTART ONLY ONCE        */
    RESTART_ATTEMPTS(1)                /* WHEN SYSTEM FAILURE      */
    RESTART_METHOD(SYSTEM,STC,'_DA04 STA DB2,LIGHT(YES)')
    ELEMENT(SYSCICS_AICICA21)
    RESTART_ATTEMPTS(1)
    RESTART_METHOD(SYSTEM,STC,'S FK21CICS,START=AUTO')
    ELEMENT(SYSCICS_AICICA22)
    RESTART_ATTEMPTS(1)
    RESTART_METHOD(SYSTEM,STC,'S FK22CICS,START=AUTO')

```

⁶ If you have a valid reason for not wanting indoubt tread resolution to take place, you must specify LIGHT(NOINDOUBTS) on the -STA DB2 command.

⁷ DB2 V9 added the ability to tell DB2 to not wait for indoubt units of recovery to be resolved, but rather to shut itself down as soon as all inflight units of work have been completed or rolled back.

4.3.2 Planned restart

If you are shutting down DB2 and plan to restart it very quickly (for example, if DB2 is being shut down to pick up maintenance), consider using the CASTOUT(NO) option on the STOP DB2 command. This option can speed shutdown processing in a data-sharing environment because DB2 does not cast out any pages that it owns as part of the shutdown process.

If you are shutting down multiple members of a data-sharing group with CASTOUT(NO), certain changed data might reside in the group buffer pools after the members have shut down. Therefore, if you want consistent data on disk (for example, you are shutting down all members to create a copy of the database to send offsite), do not use CASTOUT(NO).

With CASTOUT(NO), a retained pageset or partition P-lock is held in IX state for each object for which the DB2 member was the last updater. Also, group buffer pool connections from that DB2 enter a failed-persistent state.

4.4 General recommendations

In addition to the CF-related best practices and DB2 restart considerations, there are other suggestions that can enhance DB2 availability or performance in a data-sharing environment.

4.4.1 Minimize lock contention

Even in a non-data sharing environment, it is important to keep contention within DB2 as low as possible. Contention delays the work that is waiting for access to the resource and creates additional work for DB2 to do as it tries to address the contention.

However, in a data-sharing environment it is even more important to minimize lock contention. Our recommended thresholds are that false contention should not exceed 1% and real contention be less than 2%.

The easiest way to decrease false contention, up to a point, is to increase the size of the lock structure. Remember that the number of lock entries must be a power of two, the lock structure INITSIZE must be a power of two, plus a few MB (for control blocks within the structure). This means that each time that you increase the lock structure size, double the structure size. Eventually you get to the point where increasing the structure size stops yielding benefits (this is the law of diminishing returns). In most installations, it should be possible to get false contention down to around 1% before you reach that point.

To determine the false contention for each member of the data-sharing group, use the RMF structure detail report. The right side of the report provides the contention information for each member of the data-sharing group (Figure 4-3). To calculate the false contention percentage, divide the -FALSE CONT value by the REQ TOTAL value. If you use the RMF Spreadsheet Reporter, the RepTrdStr sheet provides the real and false contention percentages.

1	COUPLING FACILITY ACTIVITY													PAGE 27	
z/OS V1R8			SYSPLEX FPKPLEX			DATE 02/03/2009			INTERVAL 010.00.000						
			CONVERTED TO z/OS V1R9 RMF			TIME 10.50.00			CYCLE 01.000 SECONDS						

COUPLING FACILITY NAME = CFFK00															

COUPLING FACILITY STRUCTURE ACTIVITY															

STRUCTURE NAME = DSNXD22_LOCK1 TYPE = LOCK STATUS = ACTIVE															
# REQ ----- REQUESTS ----- DELAYED REQUESTS -----															
SYSTEM	TOTAL		#	% OF	-SERV	TIME (MIC) -	REASON	#	% OF	----	AVG	TIME (MIC)	----	EXTERNAL REQUEST CONTENTIONS	
NAME	AVG/SEC		REQ	ALL	AVG	STD_DEV		REQ	REQ	/DEL	STD_DEV	/ALL			

FKM0	2507K	SYNC	2426K	40.6	24.3	9.6	NO SCH	31	0.0	12.4	12.9	0.0	REQ TOTAL	2734K	
	4178	ASYN	81K	1.4	188.6	1005.7	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED	46K	
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	-CONT	45K	
													-FALSE CONT	10K	
FKM1	1765K	SYNC	9243	0.2	266.7	19.2	NO SCH	3711	0.2	16.7	32.2	0.0	REQ TOTAL	1901K	
	2941	ASYN	1756K	29.4	315.0	253.8	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED	264K	
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	-CONT	47K	
													-FALSE CONT	11K	
FKM2	1326K	SYNC	1270K	21.3	24.7	9.4	NO SCH	6	0.0	7.2	3.3	0.0	REQ TOTAL	1389K	
	2210	ASYN	56K	0.9	143.1	651.1	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED	44K	
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	-CONT	43K	
													-FALSE CONT	9298	
FKM3	375K	SYNC	1538	0.0	270.7	24.3	NO SCH	260	0.1	45.5	263.7	0.0	REQ TOTAL	341K	
	625.3	ASYN	374K	6.3	391.8	1100.0	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED	41K	
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	-CONT	37K	
													-FALSE CONT	7587	

TOTAL	5973K	SYNC	3706K	62.1	25.2	16.2	NO SCH	4008	0.1	18.5	74.2	0.0	REQ TOTAL	6366K	
	9954	ASYN	2266K	37.9	318.9	546.1	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED	394K	
		CHNGD	0	0.0			PR CMP	0	0.0	0.0	0.0	0.0	-CONT	171K	
													-FALSE CONT	38K	

Figure 4-3 Identifying DB2 false contention

Addressing real contention is more difficult. Real contention is a result of two units of work requesting incompatible accesses to the same resource (for example, a transaction trying to update a row that is currently being read by a batch job). Addressing real contention is best handled by the group responsible for scheduling work, or (if you are unfortunate) it might be related to the application design, and therefore more difficult to tackle.

There are a number of DB2 recommendations that can help minimize DB2 lock contention. For more information about this topic, see the section titled “Tuning your use of locks” in *DB2 V9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845.

Use partitioned DB2 tablespaces to enhance concurrency

In a partitioned table space, locks are obtained at the partition level. Individual partitions are locked as they are accessed. This locking behavior enables greater concurrency. Each locked partition is a separate parent lock. As a result, DB2 and IRLM can detect when no inter-DB2 read/write interest exists on that partition, and therefore they do not need to propagate child L-locks. Anything that reduces the amount of locking in a data-sharing environment is a good thing.

Use a design default LOCKSIZE of PAGE

DB2 provides the ability to lock resources at a number of levels. Generally speaking, locking at a high level reduces the cost of locking but reduces concurrency. At the other extreme, locking at the row level provides for the maximum amount of concurrency, but it increases the cost of locking.

The default in DB2 (LOCKSIZE ANY) specifies that DB2 dynamically decides what is the best level of lock to use. Usually it selects to lock at the page level. However, the current recommendation is to explicitly specify LOCKSIZE PAGE to ensure that DB2 will always use page-level locking for that table. This provides a balance between the concurrency of row locking and the minimal CPU cost of tablespace locking.

Consider using LOCKSIZE(ROW) where appropriate

Row-level locking can be implemented in cases where there are many concurrent transactions that are attempting to access data on the same physical page. A general recommendation is to implement it only if you are encountering concurrency issues.

The downside of using LOCKSIZE(ROW) is that this results in additional P locks being generated. An alternative is to use LOCKSIZE(PAGE) and MAXROWS(1) for small table spaces. This provides similar concurrency benefits as LOCKSIZE(ROW), but with less locking overhead.

Use CURRENTDATA(NO) with ISOLATION(CS)

In DB2 versions prior to V9, the default CURRENTDATA value was YES. This had the effect of disabling lock avoidance. Particularly in a data-sharing environment, lock avoidance is a good thing, so the IBM recommendation has been to override the default and select CURRENTDATA(NO). Thankfully, DB2 V9 has changed the default to line up with the recommended value of NO, so with DB2 V9 you no longer need to override the default.

Another change in DB2 V9 was the default value for ISOLATION. Prior to DB2 V9, the default was repeatable read. This might be good for integrity, but it is not good for performance because of the amount of contention that this can potentially cause. As a result, the DB2 recommendation has long been to select cursor stability. DB2 V9 changes the default for ISOLATION to be cursor stability. Together with CURRENTDATA(NO), ISOLATION(CS) increases the likelihood that lock avoidance will be successful. Remember that your applications need to be rebound with DB2 V9 to pick up the new default values.

Note: Certain changes in DB2 require a rebind of your application to pick up the change. In general, do not do a rebind until all members of the data-sharing group are stable on the new version. At that point, do a rebind of all your applications.

If you are concerned that a rebind might change the access path used by DB2, DB2 V10 provides an option to tell DB2 not to change the access path when a rebind is done.

Uncommitted read

Specifying an ISOLATION value of uncommitted read means that the program is able to tolerate a situation in which a returned row is in the middle of being updated by another process, and that the update might even be backed out. If the program is printing customer statements, it is probably required that the data being processed has been committed. However, if the program is doing something like counting the numbers of customers in the database (a value that might be changing every few seconds), then it probably is acceptable to process uncommitted data.

In the latter case, where the use of uncommitted data is acceptable, specify ISOLATION(UR), as this avoids any locks being requested by that program. The best performing lock request is the one that is never issued. Note, however, that there is only one ISOLATION level for the entire package, so if the program can tolerate uncommitted data in one table, but requires committed data from another table, you need to specify the more restrictive option.

Frequent commits

Especially in a data-sharing configuration, it is important that programs do frequent commits. Running for a long time without a commit has a number of negative aspects:

- ▶ The unit of recovery holds more locks than if a commit is issued more frequently. This increases the likelihood that this process is holding a lock that is required by another process, and that other process is forced to wait (or time out) until the commit is issued and the lock released.
- ▶ Holding a large number of locks increases the number of lock entries in the lock structure that are used, increasing both the real and false contention that is likely to be experienced.
- ▶ In case of a DB2 failure, when it restarts, DB2 must read back through its logs until it finds the beginning of the longest running unit of recovery. The longer that a process runs without a commit, the longer it takes DB2 to restart.
- ▶ In data sharing, lock avoidance is affected by the oldest CLSN Commit Log Sequence Number (CLSN) of all pagesets with inter-DB2 R/W interest. This is also known as the global commit log sequence number (GCLSN). Each member puts its value in the SCA structure in the CF, and the oldest value is used as the GCLSN for the entire data-sharing group. Therefore, it is imperative that applications issue commits at frequent intervals to ensure that the GCLSN stays current. Long-running UOWs without commits can drastically affect lock avoidance and cause DB2 to take additional locks.

Each member periodically updates its CLSN value and stores it in the shared communication area (SCA). The new GCLSN is the oldest of the CLSN for each member of the data sharing group.

DB2 provides an optional facility to identify long-running units of recovery. There are two parameters in DSNZPARM:

URLGWITH	Specifies the number of log records that can be written by an uncommitted unit of recovery (UR) before DB2 issues a warning message to the console
URCHKTH	Specifies the number of checkpoint cycles that can complete before DB2 issues a warning message to the console and instrumentation for an uncommitted unit of recovery

By default, both of these are turned off (the default is 0). Turn them on and set them to values that you deem appropriate for your environment. The smaller the number, the better from the perspective of minimizing the number of concurrent locks held by a unit of work. Also put automation or a process in place to ensure that any exceptions are detected and actioned.

Use Locking Protocol 2

DB2 V8 introduced an improved locking protocol known as Locking Protocol 2. However, this new protocol must be specifically enabled (and this requires a one-time, data-sharing, group-wide restart). This new protocol can significantly reduce DB2 false and XES contention. You might notice an increased number of CF locks for certain applications, but overall, reduced CPU usage and reduced contention are expected⁸.

⁸ Note that a one-time, group-wide DB2 outage is required to enable Locking Protocol 2.

For more information about this new protocol, see *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079.

4.4.2 TRACKMOD recommendation

The TRACKMOD parameter on the CREATE or ALTER TABLESPACE command controls whether the database manager tracks database modifications so that the backup utility can detect which subsets of the database pages must be examined by an incremental backup and potentially included in the backup image.

Setting TRACKMOD to NO can reduce the CF cost associated with the constant updating of the space map pages of the pagesets. With TRACKMOD NO, DB2 does not keep track of changed pages in the space map page of the pageset. By choosing TRACKMOD NO and not tracking updates, you reduce the CF cost, but the cost of incremental image copies is much higher because DB2 must use a table space scan to read all pages to determine whether the page has been changed and thus needs to be copied.

Our recommendation is to specify TRACKMOD(NO) if you rarely or never use incremental image copies, or if you always use DFSMS concurrent copy with DB2 LOGONLY recovery.

4.4.3 CONDBAT recommendation

The CONDBAT DSNZPARM parameter value specifies the maximum number of inbound DDF *connections*. The MAXDBAT parameter specifies the maximum number of *concurrently active* DDF connections.

We recommend that CONDBAT be set to no more than double the MAXDBAT value. Note that any changes to the CONDBAT or MAXDBAT values should be coordinated with corresponding values in DB2 Connect™.

4.4.4 RETLWAIT system parameter

The RETLWAIT DSNZPARM parameter indicates how long a transaction waits for a lock on a resource if another DB2 in the data-sharing group has failed and is holding an incompatible lock on that resource.

While we used to recommend the use of this parameter, the amount of time that it typically takes to restart a large DB2 is such that transactions are likely to time out before the retained lock is freed up. However, if your DB2 restart process is very fast, or if the DB2 is not very large, you might still consider using this parameter to get transactions to wait a little longer for the locks to be freed. The default is that the transaction does not wait for incompatible retained locks, but instead the lock request is immediately rejected, and the transaction receives a resource unavailable SQLCODE.

4.4.5 Tune system checkpoint frequency

The DB2 system checkpoint frequency can be specified in terms of the number of log records between checkpoints or in terms of the number of minutes between checkpoints. To ensure consistent restart times after a failure, set the checkpoint frequency to be a number of log records that make sense for your environment.

DB2 V10 adds the ability to specify both a number of log records *and* an amount of time, and DB2 takes the checkpoint at whichever one occurs first.

4.4.6 Relative WLM importance of DB2 address spaces

To ensure that DB2 delivers acceptable performance, that it plays its role as an active member of the DB2 data-sharing group, and especially that it can respond to lock contention events in a timely manner, it is critical that the various DB2 address spaces are assigned to appropriate WLM service classes.

One of the challenges faced by customers is how to assign the relative priorities of not just the DB2 address spaces in relation to each other, but also in relation to other major subsystems (CICS, IMS, MQ, and so on). Most product documentation tends to talk about WLM service classes just from the perspective of that product. In an attempt to address this gap, the section titled “Workload Manager” in *System z Mean Time to Recovery Best Practices*, SG24-7816, discusses appropriate WLM service classes for the major IBM subsystems in relation to each other.

4.4.7 Ensure that the system has sufficient free main storage

Typically, with today’s systems with very large amounts of storage, there is very little paging, meaning that most pages in main storage are not backed by pages in auxiliary storage. As a result, if MVS needs to obtain a large number of pages (for example, to hold one or more dumps until they can be offloaded to the dump data sets), it must first page-out a large number of pages. Because of the large amounts of storage that DB2 normally has for its local buffer pools, DB2 is typically one of the largest users of main storage and therefore is likely to be affected as MVS retrieves pages to hold the dump.

The MAXSPACE parameter defines the maximum amount of virtual storage to use when processing dumps. The default value is 500 M. However, because of its large working set, the DB2-recommended value when running on a z10 is 5000 M.

To protect DB2 performance in these circumstances, it is important that the system has sufficient free storage to hold one, or potentially more than one, dump. Also ensure that the paging subsystem is robustly configured to provide both the capacity and the performance to handle these situations. The use of SMS striping for the dump data sets can also be very beneficial to reducing the amount of time required to offload the dump information from virtual storage to the dump data sets. For more information about efficient processing for SVC dumps, see the white paper titled *z/OS Availability: Managing SVC Dumps to Mitigate Exhausting the Paging Subsystem* available on the IBM Techdocs website.

4.4.8 Workload balancing considerations

Implementing data sharing and ensuring that all critical applications perform acceptably in a data-sharing environment is an important step in your quest for the highest possible levels of application availability.

However, to truly get the benefits of data sharing, you want all incoming work requests to be automatically sent to the DB2 instance that is most likely to be able to deliver the installation’s performance objectives. Specifically, if one or more DB2 instances are unavailable, you do *not* want to have to take any manual actions to ensure that all your DB2 work can still run. Manual actions are time consuming and expose the outage to your users.

This section discusses considerations relating to DB2 workload balancing.

Remove application affinities to specific DB2 members

To be able to exploit dynamic workload balancing, it is vital that any DB2 work is able to run on any member of the DB2 data-sharing group. Specifically, ensure that there are no affinities between a given piece of work and a particular DB2 instance. While it might not always be simple to spot such affinities programmatically, the people who are responsible for managing the online systems and the batch workload typically have very good knowledge about any affinities that might exist.

Another thing to check for is that all the DB2s are set in up a similar manner, as clones of each other if at all possible. In particular, ensure that the same local buffer pools are defined in every DB2 subsystem. The amount of storage assigned might be different from one system to another, but be sure that every GBP structure has a corresponding local buffer pool in every connected DB2.

Development configuration should mirror production configuration

If your production system uses CICS MRO and DB2 data sharing, then your development and test sysplexes should be configured in a similar way. One obvious reason is that this tests applications in a production-like environment before they are moved to production. However, from an affinities perspective, enforcing the use of CICS MRO and DB2 data-sharing group names in the test sysplex help ensure that affinities cannot creep into new applications by accident.

Use DB2 group attach name rather than specific member name

Part of the process of ensuring that any of your DB2 work can run on any member of the data sharing group is to make sure that batch jobs, CICS regions, IMS subsystems, WebSphere Application Server, and WebSphere MQ servers specify the DB2 group name rather than a specific DB2 subsystem name.

There might be cases in which you want to run DB2 work on a particular system, not because you *need* to, but because you *want* to. One example might be that you want to run all the DB2 batch jobs for a given application on one system in an attempt to avoid GBP dependency. The most flexible way to address this requirement is through the use of WLM Scheduling Environments. So your batch job specifies a particular Scheduling Environment name (DBAPPLA, for example) and points to the DB2 group name. By turning the Scheduling Environment on or off on separate systems, you can easily control which systems are eligible to run those jobs. Also combine the use of Scheduling Environments with automation routines to ensure that the Scheduling Environments are turned on or off on the correct systems.

If you are using Scheduling Environments as part of an effort to minimize the overhead of data sharing, consider using the DB2 -ACCESS command to make the associated pagesets non-GBP-dependent, if possible. See “DB2 -ACCESS command” on page 109 for more information about the -ACCESS command.

Multiple DB2s in the same data-sharing group in the same z/OS

A number of customers run more than one DB2 subsystem from a given data-sharing group in a single z/OS LPAR. This is typically done either to get around constraints on the amount of above-the-line private storage, or to segregate various types of work from each other (OLTP versus data warehouse, for example). Prior to DB2 V9, if a job specified the DB2 group attach name, it connected to the first DB2 on the subsystem interface. The only way to connect to the other DB2s was to explicitly specify the DB2 subsystem name (that is, there was no automatic balancing of work across the DB2s).

DB2 V9 introduced the ability to have the work requests spread across multiple DB2s from the same data-sharing group in the same z/OS system. By default, connection requests are distributed across the available DB2s in a random manner (the connection algorithm is more

effective if APAR PK79228 is applied to all your DB2s). If you are interested in the highest levels of availability, and especially if you have work that can only run in one z/OS system (for example, it has an affinity other than to DB2), consider starting two DB2s in each z/OS. That way, if one DB2 is down for a planned or unplanned outage, the other DB2 in that system can execute all new DB2 work requests until the first DB2 is restarted, at which point the balancing algorithm once again is used.⁹

If you decide to implement such a configuration in a CICS/DB2 environment, there are two keywords on the CICS DB2CONN definition to be aware of:

- ▶ **RESYNCMEMBER:** This controls whether CICS automatically reconnects to another DB2 member if the DB2 to which it was previously connected goes away. To get CICS to connect to another DB2, specify **RESYNCMEMBER(NO)**.
- ▶ **STANDBYMODE:** Once again, to get CICS to reconnect to a separate member, specify a value of **RECONNECT** for **STANDBYMODE**.

Note that if there are INDOUBT units of work when a DB2 fails, those units of work are not resolved until that CICS region reconnects back to the original DB2. For this reconnection to take place, either restart the CICS region or stop and restart the CICS connection to DB2. On the restart, CICS realizes that it has INDOUBT units of work that it must recover, and automatically connects to the correct DB2.

While this involves a second impact to the CICS region, by letting CICS connect to another DB2 (in the same data-sharing group), that region can at least continue to process DB2 transactions until a convenient time when you can stop the DB2 work in that region and connect back to the original region to complete the recovery.

Enable dynamic workload balancing for all DB2 connectors

In the modern business environment, DB2 work no longer comes from just CICS and batch. DB2 also talks to WebSphere, IMS, MQ, VTAM (for SNA DDF work), and TCP/IP (for DRDA® requests). To ensure that *all* these requestors transparently fail over to another member of the data-sharing group, ensure that they all exploit the available workload-balancing mechanisms. This is a huge topic all on its own, so we do not discuss it in more detail here. However, *DB2 9 for z/OS: Distributed Functions*, SG24-6952, and *System z Dynamic Workload Balancing*, REDP-4563, discuss this topic in detail.

VTAM GR name

DB2 supports the VTAM Generic Resources (GR) capability, whereby a number of DB2 subsystems can be accessed using a single VTAM resource name. The DB2 GR name is specified in the DB2 installation panels, in the **GENERIC LUNAME** parameter. Connections coming in to DB2 via DDF can then use the GR name. Additionally, this DB2 can send requests to another DB2 using its GR name.

However, even though DB2 supports this capability, the use of a GR name for DB2 is no longer recommended. When a DB2 requester accesses a data-sharing group using a GR name and a member is chosen, all future workload routes to that member until the member goes down (which, hopefully, is a very long time). As a result, no distribution of connections occurs. If the connector were to use the specific member name instead, information is passed back through DDF by WLM, meaning that a new connection can go to a different DB2.

Sysplex query parallelism

If you run large queries you might consider exploiting the sysplex query parallelism feature of DB2. One obvious benefit is that if you can increase the number of DB2s working on a query, then the total elapsed time should be less.

⁹ A usermod is available from DB2 Development that rolls the round-robin balancing capability back to DB2 V8.

A side effect of the reduced elapsed time for queries is that you might be able to let new queries start closer to the DB2 shutdown time. For example, if an average long-running query takes two hours today, you might have to stop new queries from being started two hours before a planned stop of DB2. However, if you can use sysplex query parallelism to reduce the average elapsed time to half an hour, then you might be able to let new queries start up to 30 minutes prior to a planned DB2 shutdown.

4.4.9 Use rolling IPLs methodology to upgrade your DB2s one at a time

Since the announcement of Parallel Sysplex, IBM has been recommending that customers use a *rolling IPL* methodology for applying service for their z/OS systems. Rolling IPLs mean that you shut down and restart one system at a time, so there is always at least one system available in the sysplex. This is particularly relevant if one system is moving to a new release or a new service level. Rather than stopping the entire sysplex and bringing it all up on the new release, restart one system and allow it to run for a period of time to verify that the new release or service level is *clean*.

Equally, in a DB2 data-sharing environment, IBM recommends that customers never stop all the members of the data-sharing group at the same time. If you are moving to a new release or service level, then IBM recommends that two subsystems be restarted at the new level and allowed to run for a reasonable time to ensure that there are no problems with the new level. DB2 supports the coexistence of at least two releases of DB2 in the same data-sharing group. If you are planning an upgrade, remember that the lower-level DB2 usually requires toleration service to coexist with the new release, so plan ahead. In fact, it is wise to install toleration service as it becomes available, rather than waiting until the last minute.

4.4.10 Avoid single points of failure for critical DB2 data sets

As very large storage subsystems and very large DASD volumes enable customers to consolidate onto smaller numbers of control units and fewer, larger volumes, it is easy to end up with a single point of failure in your DB2 data set configuration without being aware of it.

z/OS 1.10 introduced a new system service called IOSSPOF that detects single points of failure for DASD resources. To make it easier to use, download a batch job called IOSSPOFD from the Tools and Toys website (described in 2.6, “Avoiding single points of failure” on page 76). Run this job on a regular basis for critical DB2 data sets. The program can check for two types of single points of failure:

- ▶ Are there any single points of failure between this system and a named data set or DASD volume? For example, there might be four channels to the control unit, but they are all routed via a single switch. The IOSSPOFD job detects this and reports it.
- ▶ Are there any single points of failure in common between two data sets or two volumes? For example, you might use DB2 dual logs. However, both log data sets reside in the same storage subsystem. The IOSSPOFD job detects this and reports it.

4.4.11 Archive to DASD instead of tape

DB2 gives you the choice of archiving the DB2 logs to DASD data sets or to tape. While tape might appear to be an attractive option because of the volume of data involved, IBM recommends that you archive to DASD, especially in a data-sharing environment. The reason for this is that if you need to use the DB2 logs for recovery, the log data must be merged prior to the recovery, and the merge process runs faster if the archive data sets are on DASD.

Use striping if DB2 logging performance is a concern

The DB2 log data sets can be striped using DFSMS striping support. If you have concerns about the ability of your log data sets to handle the logging rate of DB2, consider the use of striping. However, for most customers, this is not necessary. Furthermore, DB2 V9 new function mode also helps address concerns about DB2 logging rates and volumes.

4.4.12 Changing DSNZPARM values without restarting DB2

Depending on how it is done, restarting a DB2 subsystem might impact other members of the data-sharing group if the shutdown causes changes in the GBP-dependency status of certain pagesets. One way to avoid this is to do a shutdown with CASTOUT(NO), as mentioned previously. An even better way is to avoid the shutdown completely. Starting with DB2 V8, a number of DSNZPARM values can be changed without having to restart DB2. The appendix titled “Directory of subsystem parameters and DSNHDECP values” in *DB2 Installation Guide*, GC18-9846, provides information about which parameters can be changed dynamically.

4.4.13 DROP unused indexes

The LASTUSED column of the RTS table SYSIBM.SYSINDEXSPACESTATS (V9) specifies the date when the index is used for SELECT, FETCH, searched UPDATE, or searched DELETE, or used to enforce referential integrity constraints. The default value is NULL, and the column can be used to identify unneeded indexes.



IMS sysplex best practices

This chapter discusses the sysplex best practices for Information Management System (IMS) to provide the most available, scalable, and high-performing environment.

5.1 Introduction

IMS is both a transaction manager and a database manager. While using these two components together is generally the most efficient and highest performing configuration possible, there are many variations that might be better suited to your specific needs. IMS as a transaction manager provides fast and efficient access to both your Data Language/I (DL/I) and DB2 data, and transactions can be input to IMS/TM from many sources, including Systems Network Architecture (SNA) terminals, TCP/IP, and other subsystems such as Customer Information Control System (CICS). IMS as a database manager provides access to your IMS database data in several ways, including from applications running on IMS/TM, CICS, WebSphere, and others via the Open Database Access (ODBA) and Open Database Manager (ODBM) interface. The recommendations in this section are primarily directed at the Parallel Sysplex environment. However, many of them apply to any IMS configuration.

The following functional areas of IMS are explored in this chapter:

- ▶ **IMS database data sharing:** This includes considerations in the areas of Database Recovery Control (DBRC), database options, Internal Resource Lock Manager (IRLM), database-related Coupling Facility (CF) structures, and applications.
- ▶ **Shared queues:** including Common Queue Server (CQS), MVS System Logger, Resource Recovery Services (RRS), CQS-related CF structures, and scheduling considerations.
- ▶ **Connectivity:** In this section we look at Virtual Telecommunications Access Method global resource serialization (VTAM GRS), IMS Connect, ODBA, and ODBM.
- ▶ **Miscellaneous:** Other items not specifically related to one of the above.

5.2 IMS data sharing

IMS data sharing provides the ability to share data across as many as 255 IMS subsystems. There are two unique types of databases that can be used, fast path and full function, each with its own advantages. In this section we discuss those items that can provide high availability, with particular emphasis on the data-sharing environment. To provide the data-sharing function, IMS takes advantage of various Parallel Sysplex functions. These are discussed elsewhere in this book, but here we look at them specifically as they relate to their use by IMS.

For database locking in a Parallel Sysplex, IMS uses the Internal Resource Lock Manager (IRLM). The IRLM can optionally be used even in a non-data sharing configuration. Database Recovery Control (DBRC) is also essential in a data-sharing environment and provides database authorization to the sharing subsystems. IRLM and DBRC in turn use a set of z/OS functions that also must be considered.

5.2.1 DBRC considerations

DBRC is used for many functions in IMS, but the primary purpose is to provide control over which subsystems are allowed to access a given database and what level of access can be granted, and to keep track of what backups are available and which logs are required to perform a recovery if necessary.

DBRC and its associated recon data sets are not heavily used during normal online processing, but when needed it is extremely important that DBRC be available. The following best practices apply to DBRC:

- ▶ Allocate the three recon data sets with various sizes, being sure to have the spare recon allocated with the largest size. In the case of one data set becoming full, this allows DBRC to reconfigure and continue operation.
- ▶ Convert the hardware RESERVE issued by DBRC to a global ENQ using a GRS RNL conversion list. Either a RESERVE or a global ENQ works, but the ENQ is preferred because it avoids issues such as the recons getting moved to a volume with other data sets that are also subject to RESERVEs. Example 5-1 shows an example of the GRS RNL definition to convert the RESERVE to an ENQ.

Example 5-1 GRS conversion list

```
RNLDEF RNL(CON) TYPE(GENERIC)
QNAME(DSPURI01)                /* IMS RECON-DS */
```

- ▶ Place each recon data set on a separate volume, along with its own user catalog. This can be considered unnecessary if the RESERVE is being converted to an ENQ. However, if this conversion ever becomes inactive, then deadlocks can occur. It is always better to be safe than sorry.
- ▶ Implement the recon loss notification function using exit routine DSPSCIX0. IMS provides a sample of this exit, which can easily be modified for your environment. Use of this function allows sharing IMS subsystems to reconfigure immediately when a problem is detected by any of the DBRC instances. Taking recovery action immediately might prevent a delay at a critical moment, such as an OLDS switch.
- ▶ Consider using Parallel Recon Access (PRA), which became available with IMS Version 10. This can reduce contention and speed up access to the recons. PRA uses Transactional VSAM (TVS), however, and TVS requires careful planning and implementation because it requires other services, such as RRS and the MVS System Logger. If implemented and tuned correctly, it can enhance recon access. However, if not tuned properly, it can also slow you down. For further implementation details, see the IBM Redbooks publication *IBM IMS Version 10 Implementation Guide: A Technical Overview*, SG24-7526.
- ▶ Set the WLM priority of DBRC to be the same or higher than the IMS control region.

5.2.2 Database considerations

The actual database data sets can represent a single point of failure even when your configuration is totally sysplex-enabled. With the high level of availability of today's DASD, as well as the availability of both synchronous and asynchronous mirroring, the chances of a failure at the data set level are small. However, there are functions within IMS that can be implemented to further reduce this risk.

Fast path multiple area data sets (MADS)

This function for fast path DEDBs allows multiple data sets (up to seven) to be used for the same DEDB area. IMS keeps multiple copies of the data updated and in sync. This means that if a particular data becomes unavailable, the actual data continues to be available, and a new duplicate copy can be created dynamically to replace the lost data set. MADS might be especially relevant to databases that are common to many applications and critical to continuous availability.

Fast path DEDBs

The use of fast path DEDBs, even without MADS, might also improve availability, depending on the how data is distributed across the areas. If a particular area data set is lost, the rest of the database remains available. This might limit the amount of data that is temporarily unavailable until a particular area is recovered. It can also speed the recovery process because the amount of data to be recovered is less than the entire database. Fast path areas can also be reorganized online.

High availability large database (HALDB)

For full-function databases, HALDB provides similar availability characteristics to the fast path areas as described above. If a specific partition of the database becomes unavailable, applications can continue to operate against the unaffected partitions, and recovery time typically is shorter due to the smaller amount of data to be recovered. Online reorganization is also available for HALDBs.

Internal database definition options

The database definition parameters, such as the number of RAPs, RAA® FRSPC, SCAN, randomizing routines, and others including pointer options, can all have an impact on performance, contention, and availability. The discussion of these options is outside the scope of this book, but they are worth reviewing, especially when in a sysplex data sharing environment. See the *IMS V10 Database Administration Guide*, SC18-9704, and the *IMS Performance and Tuning Guide*, SG24-7324, for additional information.

Database lock contention

Database lock contention can occur in any environment, but might be more noticeable in a sysplex data-sharing environment. It is best to remove these points of contention before they become a problem. There are several methods to determine the impact that contention is having in your systems, and then to identify the specific points of contention.

If using IMS V10 or later, the optional transaction statistics show how much time transactions are spending waiting for locks. Online monitors, the IMS monitor, and the IMS lock trace, in conjunction with the KBLA utility to process the lock trace, available since IMS V9, can all be used to narrow down the problem areas. The action taken depends on the severity of the problem and the impact to the database and applications involved and is beyond the scope of this document. See the *IMS V10 System Utilities Reference*, SC18-9968, for additional information about the lock analysis utilities.

5.2.3 Database cache structures

There are three types of cache structures that can be used by IMS. There is one for OSAM databases, one for VSAM databases, and one or more for fast path shared VSO areas. You can use none, some, or all of them.

- ▶ For VSAM, the structure is a directory-only cache structure. The structure must be large enough to hold a directory entry for every VSAM buffer in every IMS subsystem, online and batch, that might be part of the data-sharing group. Add up all the possible buffers, plus 20% or so to allow for expansion, and use that number as input to the CFSIZER tool on the web to determine the proper size for the structure. Use the DFSVSMxx member of IMS PROCLIB to determine the number of VSAM buffers used by each IMS subsystem.
- ▶ For OSAM, the structure can be a directory-only structure, in which case the same considerations apply as for VSAM, with the additional input of any sequential buffer specifications. The number of OSAM buffers is also obtained from the DFSVSMxx member of PROCLIB.

- OSAM also allows the use of data caching for selected subpools in the structure. Use this option only after careful consideration. Used properly for selected subpools, it can be a powerful option, but when used too generally it can add more overhead for little benefit. See IMS product publications and *IMS Performance and Tuning Guide*, SG24-7324, for an understanding of all the options and considerations.

If using data caching for OSAM, then the data in the structure and the data on DASD are always kept the same because both are updated at sync point. This means that there is no data integrity exposure if this structure is lost.

- Fast path VSO structures can be used to improve access times for selected fast path areas. These areas can be preloaded, in which case the structure size must be large enough to hold the entire area, or non-preloaded, in which case the structure storage is maintained on a least recently used basis.

VSO area updates are not written to DASD at application sync point. Instead, they are written asynchronously when an IMS checkpoint is initiated. Because the loss of a VSO structure can result in database recovery being necessary, duplex these structures with either IMS duplexing or System-Managed Duplexing, being sure to place the duplexed structures in separate CFs.

- For all of the cache structures, both an INITSIZE and SIZE are recommended to enable structure alter. If AUTOALTER is specified, then specify MINSIZE to be equal to INITSIZE to prevent the CF from stealing storage that might be needed later.
- Monitor the cache structures to ensure that no directory reclaims occur. If they are occurring, this means that the structure is too small. Look for DIR REC in the RMF Coupling Facility usage summary report (Figure 5-1).

COUPLING FACILITY USAGE SUMMARY											
STRUCTURE SUMMARY											
TYPE	STRUCTURE NAME	STATUS CHG	ALLOC SIZE	% OF CF STORAGE	# REQ	% OF ALL REQ	AVG REQ/ SEC	LST/DIR ENTRIES TOT/CUR	DATA ELEMENTS TOT/CUR	LOCK ENTRIES TOT/CUR	DIR REC/ DIR REC XI'S
CACHE	IM0A_OSAM	ACTIVE	64M	6.7	770846	6.0	1284.7	65K	22K	N/A	0
								12K	21K	N/A	0
	IM0A_VSAM	ACTIVE	15M	1.5	310757	2.4	517.93	50K	0	N/A	0
								9915	0	N/A	0

Figure 5-1 RMF CF usage summary report

5.2.4 IRLM

With IMS sysplex data sharing, the IRLM is used to provide locking services. It also can optionally be used as the lock manager on a single system. The IRLM modules being used by IMS can reside in a common library with DB2. However, the actual instance of IRLM cannot be shared between IMS and DB2.

Example 5-2 shows example IRLM procedure parameters, which are referenced in the list that follows the example.

Example 5-2 IRLM procedure parameters

```
//DXRJPROC PROC RGN=3072K,
// IRLMNM=IRLM,
// IRLMID=,
// SCOPE=LOCAL,
// DEADLOK='5,1',
// MAXCSA=8,
```

```
// PC=NO,
// MAXUSRS=,
// IRLMGRP=,
// LOCKTAB=IRLMT1,
// TRACE=YES,
// PGPROT=YES,
// LTE=
```

The recommended values for the IRLM parameters are:

- ▶ Set MAXUSRS to the maximum number of IRLMs expected in the data sharing group, but not larger than necessary. This determines the size of each lock table entry (which is either 2, 4, or 8 bytes). A MAXUSRS value between 2 and 7 yields a 2-byte entry, between 8 and 23 results in a 4-byte entry, and above 23 results in an 8-byte entry. The smaller the MAXUSRS value, the more entries there can be within a given size structure, and the more entries there are, the less likely you are to have false contention. The IRLM automatically increases the lock entry size in the event that more IRLMs are started than can be supported with the current entry size. However, if this occurs you automatically reduce the number of lock table entries by 50%, which might lead to increased false contention. The increase also triggers a rebuild of the lock structure, which pauses all activity to the structure for the duration of the rebuild.

- ▶ Set DEADLOK low. There are two values:
 - The local deadlock detection time (in seconds or milliseconds)
 - The number of local deadlock detection cycles before performing global deadlock detection

The second value is always set to 1 regardless of what is specified. If you do not experience deadlocks in your system, the time specified for local deadlock detection has no consequence except that it results in slightly more CPU at lower times. However, if you *do* experience deadlocks, then a low number is essential in getting the deadlock resolved and work flowing as quickly as possible. Consider using values of one second or less.

- ▶ IRLNM is the name used when IMS connects to the IRLM. Consider making this the same name for all IRLMs in the group so that IMS online or IMS batch can be run on any system in the data-sharing group without having to change the name being used. Only the IRLMID must be unique for each IRLM in the data sharing group.
- ▶ IRLMGRP is the name of the XCF data sharing group used by IRLM, and must be the same for each IRLM that is a member of the group.
- ▶ Do not specify LOCKTABL. The name of the lock table is passed to IRLM by IMS when it identifies to the IRLM and is specified in the DFSVSMxx CFNAMES statement.
- ▶ Specify TRACE=NO to reduce CPU overhead unless there are suspected locking problems.
- ▶ SCOPE for multi-system data sharing must be either GLOBAL or NODISCON. NODISCON stops the IRLM from disconnecting from the lock structure when all identified subsystems have terminated. If you run batch data-sharing jobs and have an IMS online subsystem that might not always be connected, then specify the NODISCON specification. This prevents the constant connection and disconnection to the lock structure and the possible failure of jobs that can result if the maximum number of connections is exceeded.
- ▶ PC=YES is the only value used with IRLM 2.2 regardless of what is specified. PC=NO can still be used with IRLM 2.1 and uses slightly less CPU, but requires more ECSA storage. With today's processors, the CPU difference is likely unnoticeable.

- ▶ MAXCSA is ignored if PC=YES is specified. Otherwise, for IRLM 2.1 with PC=NO, set it to prevent the IRLM from taking too much ECSA and possibly causing problems for other tasks that need space in ECSA.
- ▶ LTE can override the default specification of 50% of the lock structure being kept for lock table entries. Do not specify this value unless you have a good reason and understand the implications.

There are also considerations for the IRLM Lock structure:

- ▶ Structure size: the size of the lock structure should be specified as a power of 2. For production systems it should probably be *at least* 64 MB, but anything larger than 512 MB is usually unnecessary for an IMS environment. The goal is to avoid filling up the record list entries (which normally occupy half the structure) and to minimize false contention by having many lock table entries. You can avoid filling up record list entries by having frequent application checkpoints. Specify a SIZE (maximum size) that is larger than the INITSIZE value to allow for dynamic alter of the structure size. To monitor false contention, see the RMF Structure Activity report (Figure 5-2).
- ▶ Structure duplexing: If using a failure-isolated Coupling Facility, it should not be necessary to duplex the lock structure. However, if ICFs are being used, use System-Managed Duplexing to avoid a single point of failure. If the lock structure and one or more connected IRLMs are lost at the same time, all IMSs in the data-sharing group are quiesced until the failing IMS and IRLM are restarted and the lock structure is rebuilt.

COUPLING FACILITY STRUCTURE ACTIVITY													

STRUCTURE NAME = IM0A_IRLM TYPE = LOCK STATUS = ACTIVE													
SYSTEM NAME	# REQ		# REQ	REQUESTS			REASON	# REQ	DELATED REQUESTS		AVG TIME (MIC)	STD_DEV	EXTERNAL REQUEST CONTENTIONS
	TOTAL	AVG/SEC		% OF ALL	-SERV AVG	TIME (MIC) STD_DEV			% OF REQ	---- /DEL			
#@\$2	4263K	SYNC	3839K	45.1	21.3	11.6	NO SCH	6710	0.2	21.2	58.2	0.0	REQ TOTAL
	7105	ASYN	424K	5.0	73.1	58.7	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	-CONT
													-FALSE CONT
#@\$3	4259K	SYNC	3060K	35.9	22.4	12.0	NO SCH	12K	0.3	18.4	41.8	0.1	REQ TOTAL
	7098	ASYN	1199K	14.1	64.8	54.6	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED
		CHNGD	0	0.0	INCLUDED	IN ASYN	PR CMP	0	0.0	0.0	0.0	0.0	-CONT
													-FALSE CONT
TOTAL	8522K	SYNC	6899K	81.0	21.8	11.8	NO SCH	19K	0.2	19.4	48.2	0.0	REQ TOTAL
	14204	ASYN	1623K	19.0	67.0	55.8	PR WT	0	0.0	0.0	0.0	0.0	REQ DEFERRED
		CHNGD	0	0.0			PR CMP	0	0.0	0.0	0.0	0.0	-CONT
													-FALSE CONT

Figure 5-2 CF structure activity report

5.2.5 Other database-related considerations

The following items do not logically fall into any of the above categories, so they are listed here:

- ▶ DFSVSMxx: Use the same member for multiple systems if they are clones. This prevents possible oversights if using multiple members and only one is changed. Conversely, you might want to use a new member for verifying changes on one system before rolling that change into the other IMS images, in which case you must have a separate member for that system.
- ▶ FDBR: Use FDBR and be sure that it runs on a separate LPAR and preferably on a separate physical machine to provide failure isolation. This allows retained locks to be released quickly and provide full data availability. FDBR also provides the capability to

drive an exit routine called DFSFIDN0, which can help you resolve any in-doubt units of work that might exist with other subsystems like DB2.

- FDBR PSB-related pool sizes must be the same size as, or slightly larger than, the corresponding pools in the IMS that FDBR is monitoring to ensure that backout and redo processing do not fail. In addition, if the RESIDENT option is used for PSBs in the active IMS, then the pool sizes of the active IMS might not be appropriate for use with the FDBR region since FDBR does not use this option. In this case you must take the size of any resident PSBs into account for the FDBR region PSB pool.
- LOCKTIME: Use either the DFSVSMxx LOCKTIME settings or the IRLM modify command to set the lock timeout value to something other than the default of 300 seconds. The DXR162I message can be monitored to know if long lock waits are occurring and the RMF ILOCK function will show which programs and resources are involved. The IRLM DEADLOCK value might affect when a locktime value has been exceeded because the IRLM only checks for programs exceeding locktime during a deadlock cycle. Another thing to remember is that the use of LOCKTIME in the DFSVSMxx member causes an action to be taken (status or abend), whereas not using that function only causes the DXR162I message and the SMF 79.15 record to be written. Example 5-3 shows a sample of the data in the 79.15 SMF record.

Example 5-3 Sample of data from 79.15 SMF record

- DATE	TIME	DEADLOCK	BLK	ELAP	DB/AREA	MVS	IMS	TRAN/JOB	RGN	PSB	NAME	PST	MAX	CICSID
YY.DDD HH:MM:SS.TH		CYCLE	WT	TIME	NAME	ID	ID	NAME	TYPE			NBR	LOCK	
09.286	06:08:37.17	000003F0	W	0020	AREADI01	3090	IM1A	DDLOCK2	BMP	FPSBPA		0002	0000	
09.286	06:08:37.17	000003F0	B	0061	AREADI01	3090	IM1A	DDLOCK1	BMP	FPSBPA		0001	0001	

-LOCK	-----	NAME	RECOVERY	-----	TOKEN
LL-----	RBA-DMBDSTTPART				
0900000002C800201C60000	C9D4F1C14040404000000000200000000				
0900000002C800201C60000	C9D4F1C14040404000000000100000000				

- IMSGROUP can be used to allow BMPs to run on any IMS that is active on the system where they are started, allowing much more flexibility in scheduling. Do not forget, however, that the x'18' log records must be available to the BMP in case of restart on another IMS. These are provided with the IMSLOGR DD statement. You might want to consider an additional product such as the IMS Program Restart Facility to assist with this process.

5.2.6 Applications

The way that applications are designed and coded can have a major impact on availability and performance in a sysplex environment. Some of the key items to be considered are:

- ▶ PSB PROCOPT: Many applications use PROCOPT=A even when no updates are made. This might cause unnecessary serialization due to lock contention and possibly even deadlocks. Review the PROCOPT setting, especially for high-volume applications, and set it to G or GO if possible.
- ▶ Database control records: Certain applications have implemented databases to provide unique sequence numbers or time values. These databases can become bottlenecks, as well as single points of failure. There are several methods to accomplish the same function without creating bottlenecks, but application changes might be required. Use the IMS monitor and IMS lock trace with the KBLA lock analysis utility to identify any problem areas.
- ▶ Avoid calls outside of IMS: Calls to APPC, MQ, or using synchronous callout within the application must be considered carefully. Avoid making these calls while holding DL/I or DB2 locks, which could cause contention or deadlocks with other applications.

5.3 IMS shared queues

IMS shared queues provide a way for any IMS subsystem in the same shared queues group to process a transaction, no matter on which system it arrived. There are also two unique shared queues:

- ▶ One for fast path
- ▶ One for full function

They both have the same goal of allowing work to be processed on any IMS, but they each have unique characteristics.

The shared queues function of IMS uses the common queue server (CQS) for managing the queues, and CQS in turn interfaces with XES for Coupling Facility structure access and with the MVS System Logger to provide recovery of the queues. In addition, IMS might use the services of RRS for coordinating units of work across multiple subsystems.

Figure 5-3 can be used to help understand how the various functions and data sets are related in the following discussion.

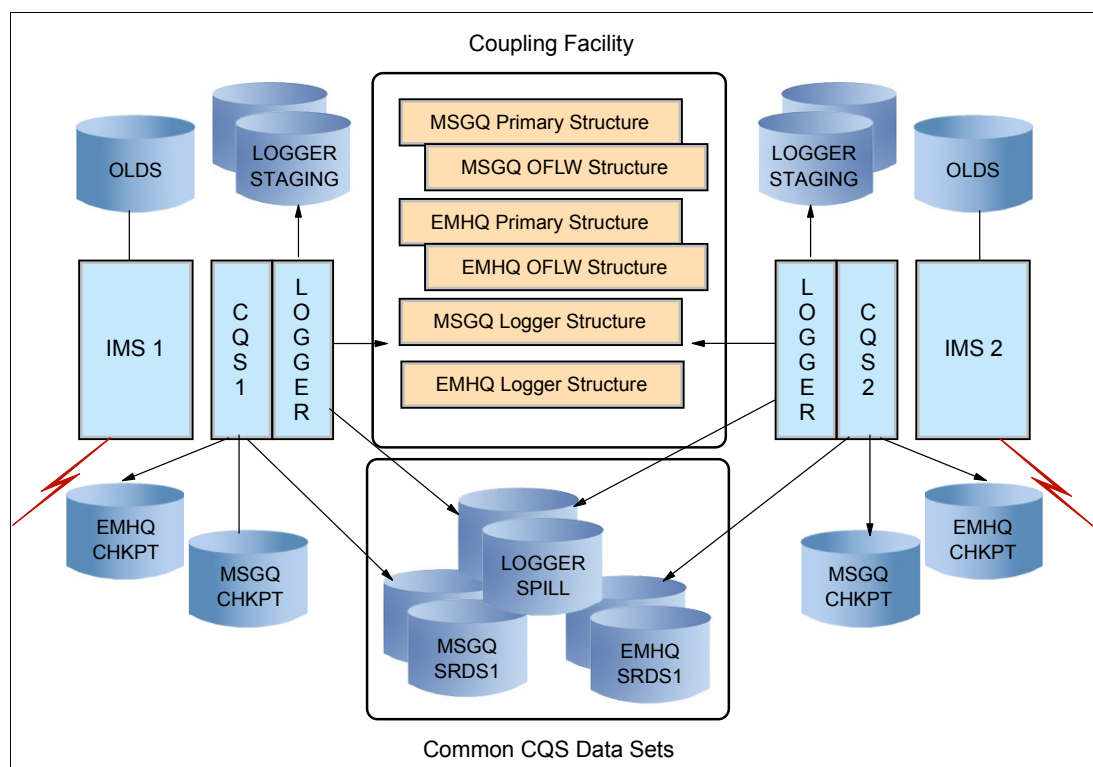


Figure 5-3 Shared queues components

5.3.1 CQS system checkpoints

CQS system checkpoints provide a recovery point for CQS restart in the event of a CQS failure. They can be initiated by a structure checkpoint command or by the SYSCHKPT value in the CQSSLxxx member of IMS proclib.

The number of log records that must be read by CQS during restart affects the time to complete a restart. Internal tests using DS8000 DASD showed it taking about 40 seconds to read 2.25 million records or about 56,000 records per second. One additional factor to remember is that the SYSCHKPT value is per CQS, but the log records being read are for all systems.

There is only a single CQS checkpoint data set for each CQS log stream, FP and FF. However, in the unlikely event that this data set is lost, it is possible to enter the log token from which to restart. This token can be obtained from the CQS0030I message issued when a system checkpoint is taken.

5.3.2 CQS structure checkpoints

CQS structure checkpoints are used to recover the CQS shared queue structures if any structure becomes unusable. There is no parameter to cause a structure checkpoint to be taken automatically, so the IMS Time Controlled Facility (TCF) or other automation must be used to facilitate these checkpoints. The frequency of these structure checkpoints determines the time that it takes to recover a structure and how much log data must be kept by the MVS System Logger.

There are two structure recovery data sets (SRDS) per log stream. The SRDSs are used in alternating sequence. CQS attempts to use the most recent SRDS and the log data from that point forward to perform a structure recovery. The following recommendations might help improve CQS performance:

- ▶ Automate the taking of structure checkpoints. Set the frequency and time of structure checkpoints to minimize disruption to your IMSplex. Remember that all activity to a given structure is quiesced from all CQSs during the checkpoint.
- ▶ Recovery time is mostly a function of the response time to the shared queues structures. The log read time is typically very fast, and while it adds slightly to the recovery time, it is not a major factor. There are approximately 1.5 structure accesses for each log record processed. The number of log records processed per period of time can be extracted from the SMF Type 88 records created by System Logger. Using that number and the average response time for the shared queues structure multiplied by 1.5, you can estimate the recovery time for that number of log records and determine whether that time is acceptable in the event of a failure. Remember, all log records from all CQSs must be processed to recover the structure.
- ▶ Use MSGBASED processing for the CFRM couple data sets. This minimizes the time to quiesce and resume structure activity because the IXLUSYNC function is used by CQS to coordinate structure checkpoints across multiple CQS images.
- ▶ If using both EMH and full-function shared queues, stagger the structure checkpoints.
- ▶ Place the two SRDS data sets on separate DASD volumes, and separate controllers if possible to minimize the risk of losing both data sets.

5.3.3 CQS shared queue structures

The structures used by shared queues include the primary and overflow message queue structures, and, if using fast path, the primary and overflow EMH queue structures. Only the primary message queue structure is required. However, we also recommend defining the overflow structures. In addition, consider the following:

- ▶ Specify both a SIZE and an INITSIZE for the structures. This allows dynamic altering of the structure size in case a structure starts to become full.
- ▶ Allocate the overflow structures slightly larger than the primary structure maximum size. In the situation where a queue must go into the overflow structure, then all of those messages must be moved or none of them can be moved. If only one or two queue names are causing the primary structure to fill, the overflow structure must be large enough to hold the entire queue of messages.
- ▶ Specify the correct entry-to-element ratio (controlled by the OBJAVGSZ value in CQSSGxxx) to facilitate CQS overflow processing. CQS monitors only the number of elements in use and does not detect whether the entries become full. The average message size is for both input and output messages combined. If in doubt, specify 512 to get a 1:1 ratio.
- ▶ Enable AUTOALTER for the CQS structures to allow XES to dynamically alter the internal characteristics of the structures including the ratio mentioned above. However, if you enable AUTOALTER for the CQS structures, it is advisable to specify MINSIZE as being equal to INITSIZE to prevent the CF from reducing the size of the shared queues structures.

5.3.4 CQS and the MVS System Logger

CQS uses the MVS System Logger to log message activity. The System Logger always keeps two copies of the log data in its log streams. This duplexing can be to an MVS data space, to staging data sets, or to another structure using System-Managed Duplexing. The method of duplexing is determined by the hardware configuration and how the log streams are defined in the Logger policy. System Logger also offloads data from the structure to DASD when the structure reaches a predefined full threshold. The following list details the best practices for the MVS Logger as related to CQS log streams:

- ▶ Use failure-isolated CFs to avoid the necessity of using staging data sets or duplexed structures when possible. For GDPS environments or if you use DASD mirroring for disaster recovery, this might not be possible, and the use of staging data sets might be necessary.
- ▶ If using staging data sets, be sure to allocate enough space (much more than the structure) so that the full threshold is triggered by the structure and not by the staging data sets filling.
- ▶ Set the size of the Logger structure and the CFRM FULLTHRESHOLD value to provide ample warning time to take action in the event of an offload failure.
- ▶ Make the offload data set size (LS_SIZE in the Logger policy) large to avoid frequent new data set allocations. New allocations are more costly and time consuming than offloading to an existing data set.
- ▶ Set the log stream LOWOFFLOAD value to 0 so that all data is moved to DASD after the offload process begins. For this type of log stream, sometimes called a funnel log stream, there is no benefit in keeping some of the log blocks in the structure.
- ▶ Set the log stream HIGHOFFLOAD value to between 50 and 70% to avoid filling the structure before offload can complete. Together with the structure size, you can also provide early warning if the offload fails for any reason.
- ▶ Set MAXBUFSIZE to 65276 (the largest possible buffer size) to use a 265-byte element size. This setting optimizes the use of the storage in the CF.
- ▶ Specify ALLOWAUTOALT(NO) for the Logger structures. Logger monitors and alters the structure characteristics itself if necessary.
- ▶ If you are using both full function and fast path shared queues, use separate structures for the full function and fast path log streams. They are used differently, and therefore management is easier when separate.
- ▶ Use the information provided in the IXGRPT1 program to determine how the CQS log streams are performing. Information about this program, and much more detail about CQS's use of the System Logger, are explained in the *Systems Programmer's Guide to: z/OS System Logger*, SG24-6898.

5.3.5 False scheduling

False scheduling occurs primarily in a shared-queues environment. There is local false scheduling and global false scheduling. It can cause unnecessary overhead depending on various options that have been specified for your transactions and regions:

- ▶ Use PWFI and dedicated regions for your high-volume transactions. This avoids the scheduling process even if the region is posted to check for additional messages, which might not exist. This setting also tends to keep more transactions local to the system where they arrived, thereby reducing overhead and interactions with RRS.
- ▶ Use a PARLIM of 2 or greater to avoid local false schedules in IMS V10 and previous versions. IMS V11 contains enhancements to avoid local false schedules even with PARLIM 0 or 1.
- ▶ CF Level 16 might help reduce global false schedules due to an enhancement in the CF whereby it initially only notifies one IMS when a queue transitions from being empty to non-empty. By default, the IMS initially notified is given 5000 microseconds to retrieve the message before the other interested IMSs are informed. Prior to this CF Level, all interested IMSs would be informed of the queue transition, but only one of them would be able to retrieve the message. This resulted in the global false schedules that might be observed in such an environment.

Subsequent to CF Level 16, APAR OA30994 was released. This APAR allows you to control, at the structure level, the amount of time that the CF waits before it informs the other IMSs of the queue transition. This is specified with the SUBNOTIFYDELAY setting in the CFRM policy.

Just as it is prudent to stay on current software service levels, also check that you are on a current CF service level to avoid possible problems.

5.3.6 Resource Recovery Services (RRS) considerations

IMS uses the services of RRS to provide coordinated commit processing across multiple subsystems. Because this section refers to IMS shared queues, we are primarily interested in the cascaded transaction support function of RRS. The following recommendations can help you avoid RRS-related problems:

- ▶ Determine whether you need the services of RRS for either shared queues cascaded transaction support or coordinated commit with other subsystems. If it is not required, then specify RRS=N in the IMS startup parameters. Even if RRS is not required there will be a certain amount of overhead if Y is specified.
- ▶ Disable the RRS archive log stream to reduce overhead. With z/OS 1.10 or later, this can be done using the SETRRS ARCHIVELOGGING,DISABLE command. With earlier releases, you must shut down RRS across the entire sysplex and delete the archive log stream definition from the Logger policy.
- ▶ If you want to avoid the use of RRS cascaded transaction support, use commit mode 0 or commit mode 1 transactions with SYNCLEVEL=SYNCPPOINT.
- ▶ Prudent use of PWFI means that more transactions can be processed locally on the system where they arrive, thus avoiding the use and overhead of RRS cascaded transaction processing.
- ▶ The RRS delayed log stream is a so-called *active* log stream. This means that RRS deletes log records from the log stream shortly after they are created. Therefore, for the most efficient use of this log stream, ensure that the associated structure is large enough to avoid offloading any data to the Logger offload data sets. You can use the IXGRPT1

program and look for bytes written to DASD of greater than 0. For more information, see *Systems Programmer's Guide to: z/OS System Logger*, SG24-6898.

5.4 IMS connectivity

There are many ways to connect to IMS. You can connect to the IMS transaction manager from VTAM SNA devices, through MVS APPC, MQ, and TCP/IP to IMS Connect. In addition, you can connect to the IMS database manager directly with CICS, ODBA, or other means through the architected interface.

Consider the following information:

- ▶ For IMS Connect, add an entry to the program properties table for HWSHWS00 and set it as non-swappable. If this is not done, response times can vary widely, especially at peak demand times.
- ▶ Also for IMS Connect, specify a datastore for each IMS and provide either an exit routine or use IMS Connect Extensions to allow routing to multiple IMSs for workload balancing or to address the unavailability of a given IMS subsystem.
- ▶ Set ECB=Y and IPV6=Y in the HWSCFGxx member for best performance.
- ▶ Specify NODELAY=Y in the HWSCFGxx member to avoid delays when IMS Connect sends data.
- ▶ TCP/IP settings of TCPNODELAY=ENABLE and NODELAYACK minimize delays.
- ▶ Use VTAM generic resources with IMS to balance sessions across the available IMS subsystems and minimize disruption to users by routing logons to any member of the group.
- ▶ Use the IMS Resource Manager with Sysplex Terminal Management to allow conversations and other terminal status to be continued on another IMS in the event of a failure. For more information about the Sysplex Terminal Management function, see the *IMS Version 8 Implementation Guide A Technical Overview of the New Features*, SG24-6594.
- ▶ For DBCTL, use care when defining the DRA resource (DFSPZPxx). These resources have an impact on various IMS specifications such as PSTs, scheduling pools, and fast path buffer usage. This can be especially critical if adding new CICS AORs because the resource demand can easily be multiplied.
- ▶ ODBA and ODBM: The same considerations apply here as with DBCTL.

5.5 Other miscellaneous considerations

This section contains various recommendations and considerations that did not specifically belong in the previous sections but are nonetheless important.

CF microcode upgrades

A word of warning is appropriate here. New CF levels generally result in an increase in the amount of control block space within the CF structures, meaning that there is less user storage available for the same size structure. Be sure to use the CFSIZER tool for your IMS-related structures prior to any microcode upgrade to be sure that the current size is adequate.

DASD mirroring

Whether you are exploiting sysplex or not, you might be using some form of DASD mirroring, which can be asynchronous, synchronous, or both. Both might impact your production IMS. Typically, the most critical data sets are the WADS and, if you are using them for duplexing, the staging data sets for the CQS log streams and the RRS Delayed log stream.

Synchronous mirroring increases your DASD response time by a certain amount depending on the distance between controllers and the type of hardware. Because these data sets are performance-critical to IMS, the impact of synchronous mirroring on these data sets must be carefully monitored.

Asynchronous mirroring can also impact response times, especially for the data sets mentioned above, although the impact is the same regardless of the distance between the primary and secondary devices.

The key to avoiding response time problems with these data sets is to avoid rewriting the same block of data in the same consistency period. This consistency period can vary depending on several factors, but usually is a few seconds. Because these data sets are relatively small and are used in a wrap-around fashion, it is good practice to make them large enough to avoid wrapping within a consistency period. Your colleagues who are responsible for the remote mirroring environment should be able to help you identify what this time is.

- ▶ **WADS:** Use the logger statistics provided by IMS to determine the logging rate and calculate how many buffers will be filled per second at peak times. Multiply this value by the number of seconds in a consistency period. This gives you the minimum number of log buffers and WADS track groups that you should have to avoid a problem. Allocate somewhat more than the minimum, however, to be on the safe side.
- ▶ **RRS and CQS Logger staging data sets:** These data sets should be allocated large enough, like the WADS, to avoid reuse within a consistency period.

Automatic Restart Manager (ARM)

IMS, as well as related address spaces such as IRLM, DB2, and CICS, can be restarted by the MVS Automatic Restart Manager. In addition to the ability to restart a single subsystem, ARM allows for specifying groups of address spaces to be restarted together in case of a system failure. This grouping is essential when IMS is coordinating with other subsystems to resolve any in-doubt units of work. ARM can be used together with various automation tools to provide quick restart following a system or subsystem failure.

External subsystem indoubt notification exit

This exit, DFSFIDN0, when used together with FDBR and DB2 Restart Light, provides a method to quickly resolve in-doubt units of work and allow full access to DB2 data. Without the use of this exit, any DB2 data held by an in-doubt IMS transaction at the time of a system failure is held and unavailable to surviving systems until both IMS and DB2 complete restart.

Workload Manager (WLM)

IMS server address spaces such as the control region, DLI, DBRC, and IRLM, must always be to a higher importance WLM service class than the associated dependent regions. A good practice is to avoid classifying IMS application work lower than the transaction class level. The reason for this is that IMS schedules the programs into dependent regions by class, and WLM manages the priority of those regions. If multiple transactions process in the same class, then it is unproductive to manage at the transaction level.

IMS Operations Manager

The Operations Manager (OM) component of the Common Service Layer, together with TSO SPOC, provides comprehensive IMS sysplex operations support. All future command enhancements will be done with the OM interface, so install and use this interface.



MQ sysplex best practices

WebSphere MQ is IBM's premier messaging product, providing messaging-oriented communications across many different platforms. It allows applications coded in many different languages to communicate using a common API. It provides for multiple qualities of service levels for messages. The highest level of availability for a WMQ message is provided by exploiting the MQ shared queues capability, which is only available with WebSphere MQ for z/OS.

6.1 Introduction

WebSphere MQ enables applications to participate in message-driven processing across the same or different platforms. The WebSphere MQ flavour that meets the most stringent requirements for both the response time for messages and also the availability of the service (minimizing planned, unplanned outages) is the use of MQ shared queues in a Parallel Sysplex.

With traditional private (or non-shared) queues in MQ, the messages are stored in a pageset on DASD, and are only accessible to one queue manager. If that queue manager is down for any reason, those messages cannot be accessed by any other queue manager. Further, no logging is done for non-persistent messages, so if a queue manager address space fails, any non-persistent messages in that queue manager at the time of the failure will be lost.

By contrast, the messages in a shared queue are kept in a structure in a CF, and therefore are accessible to multiple queue managers, on the same or different systems, as long as they are all in the same sysplex. This means that if one queue manager is down for some reason, the messages are still accessible via the other queue managers. In fact, as long as just one queue manager is still up, the messages can still be accessed.

And if we take it a step further - let us say that for some reason, every queue manager in the sysplex has to be shutdown. In a non-shared queues environment, all non-persistent messages that were still in a queue would be lost. However, in a shared queue environment, if all the queue managers are shut down (but the CF remains available), when the queue managers come back, all messages (both persistent and non-persistent) that were on a shared queue before the shutdown would still be available.

Based on these characteristics of MQ shared queue, the benefits are:

- ▶ High availability of queue managers and related server applications.
- ▶ Scalability and workload balancing based on a “pull” mechanism.
- ▶ Near-continuous availability of WMQ messages.
- ▶ Low cost messaging within the sysplex.
- ▶ Queue Manager Peer Recovery.
- ▶ Reduced administration.
- ▶ Increased channel availability.

A group of queue managers that can access the same WebSphere MQ object definitions and message data is called a Queue Sharing Group (QSG). A QSG on WebSphere MQ for z/OS consists of the following components:

- ▶ A number of MQ for z/OS queue managers (running in a single sysplex), along with their associated channel initiators.
- ▶ A Coupling Facility (CF) containing one or more structures that are used to hold messages on shared queues.
- ▶ A shared DB2 database (known as the DB2 shared repository) that is used to hold shared object definitions.

The use of MQ shared queues, however, is not cost-free. The medium that is used to store the messages is very different for shared queues than for private queues. This results in a number of considerations, including whether the benefits of using shared queues for a particular application are justified compared to the cost, and whether the application being

considered is suitable for implementation in a shared queues environment given the amount of storage that would be required and is available.

For more information about configuring WebSphere MQ for high availability, see the SupportPac titled WebSphere MQ for z/OS: Highly Available System Design, available on the web at:

<ftp://public.dhe.ibm.com/software/integration/support/supportpacs/individual/md17.pdf>

6.2 Location and sizing

It is important for maintaining smooth operations that the shared queue-related structures are appropriately sized and placed in CFs that will deliver the required performance and availability.

6.2.1 Coupling Facility structures.

WebSphere MQ shared queues use list structures in two ways:

- ▶ The first type of structure is the Administration structure (CSQ_ADMIN). This is used by the queue managers in the QSG to ensure data integrity when recovery is required, to coordinate serialized access to queues, and generally to manage the shared queues environment. No user data is held in this structure. It has a fixed name of qsg-nameCSQ_ADMIN (where qsg-name is the name of your queue-sharing group).
- ▶ The second structure type is the application structure. These are the structures that will contain the actual MQ messages. They are called application structures because they are primarily used by applications for message queues.

Application structures

The use of CF structures to store its messages means that MQ can take advantage of many of the facilities provided by the Coupling Facility to efficiently implement queuing primitives, including; the ordering of messages, atomic updates to the data, and list depth monitoring (for triggering and get-wait processing).

Each shared queue is represented in the list structure by a separate list, and up to 512 queues can be defined in each structure.

When messages are stored in the list structure, each message receives a “list entry” and one or more list elements (more about the elements later). The list entries contain a key, which enables the list to be sorted. The order is based on the message age and priority, and optionally by the message ID or the correlation ID. The format of the key used when MQPUTting the messages means that when an MQGET is subsequently performed, the selection of the message can be performed by the CF, rather than the queue manager. The CF will return the highest priority oldest message that matches the selection criteria (assuming the queue is defined to use priority order).

Most of the management of the CF structures is performed using the standard MVS commands which enable the attributes (size, location etc) of the structures to be altered while the queue managers are still connected to the structures.

A QSG requires a minimum of two list structures - the administration list structure and an application list structure. However, your application environment might demand that you define multiple application list structures - for data isolation reasons, for example. Another reason to split the application structure could be the persistence of the messages - you might

decide to separate queues that contain persistent messages from those that contain non-persistent messages, and keep each type in a separate structure.

For performance reasons, it is recommended that you use as few CF application structures as reasonable. In particular, if an application issues MQGETs from one queue, and MQPUTs to another queue within a single sync point, it is recommended that both queues are placed in the same application structure.

Structure availability

If a queue manager loses connectivity to either an application structure or the admin structure, that queue manager will abend. This could happen, for example, if the system that the queue manager is running on loses its last link to that CF.

If the CF containing either the admin structure or an application structure were to fail, *all* MQs in the queue sharing group would abend.

There are also considerations in relation to the contents of the structures. Non-persistent messages would be lost if the structure containing them, or the CF containing that structure, were to fail. This is no different to what would happen to non-persistent messages in a non-shared queue if the owning queue manager were to fail, with the exception that the structure probably contains messages from multiple queue managers.

Persistent messages *can* be recovered if the structure they are in, or the CF containing that structure, were to fail. This depends on you taking frequent structure backups - this is discussed further in 6.2.2, “Structure backup and recovery” on page 146.

While it would be very unusual for either a CF link or an entire CF to fail, such a failure is possible. It is prudent, therefore, to determine how such an outage would affect you.

You can protect against the impact of a CF failure by using System-Managed Duplexing for the MQ structures. Duplexing all MQ structures (administration and application) eliminates the CF as a Single Point of Failure. Duplexing the MQ structures provides two levels of benefits:

- ▶ It avoids the outage of one or more queue managers resulting from a CF connectivity failure or a complete CF failure.
- ▶ It ensures that the loss of a CF structure will not result in the loss of any messages that were in that structure at the time of the failure.

However, System-Managed Duplexing is expensive in terms of CF and z/OS CPU and increased response times to structure requests, and can potentially impact throughput.

It is up to each installation to look at your MQ applications and determine if the availability benefits of duplexed structures justifies the additional CPU cost and the increased response times.

Note that if you determine that you cannot afford an outage of all queue managers in the queue sharing group, this effectively means that *all* MQ structures must be duplexed. Duplexing just a subset of your MQ structures would protect the contents of the duplexed structures, however if a CF containing a simplex MQ structure were to fail, then all queue managers would abend. If your objective is to ensure that you never lose all queue managers, then you really have to duplex all MQ structures used by that queue sharing group.

MQ CF structure sizes

The administration structure size depends on the number of queue managers in the QSG, and the CFCC level. It is recommended that the size of the administration structure is at least 20 MB.

The size of each 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 that application structure. Of course, you might also be limited by the amount of available storage in the Coupling Facility.

You should use the IBM Coupling Facility Structure Sizer tool to calculate the size of the structures to be defined. The CFSizer tool is available on the web at:

<http://www.ibm.com/systems/support/z/cfsizer/>

See “Exploit new CF functions” on page 44 for information about the impact of CF Level changes on structure sizes.

Adjusting CF structure sizes

In the CFRM policy it is possible to enable a structure for Auto Alter. Having this option enabled allows the system to automatically adjust the size of the structure and also the entry to element ratio used.

Messages are stored in an MQ application structure as a “list entry”, which contains control information for that message, and 1 or more “list elements” which contain the message data. (By message data here we mean both MQ internal information, any MQ headers (for example, MQMD, MQXQH) and user data). Each list element can hold 256 bytes of information. Empirical evidence suggests that messages stored on shared queues are, on average, between 1 KB and 2 KB in size. Based on these message sizes, MQ requests an entry-to-element ratio of 1:6 for the application structures. This means that the CF will most efficiently store messages where each message has an associated 1.5 KB (6*256) of data. You do not have any control over the entry-to-element ratio, however the Auto Alter function *does* have the ability to adjust the ratio.

When messages are larger than this, the CF structure will tend to run out of list elements before all the list entries are used. And, conversely, if messages are smaller than this, the CF structure will tend to run out of list entries first. In extreme cases, where the average message size stored in the CF Structure is much smaller than this 1.5 KB average, the CF structure will become *full* due to running out of list entries for new messages even though there is still storage available for more list elements.

To address this situation, we recommend that you enable the MQ structures for Auto Alter by specifying the ALLOWAUTOALTER keyword in the structure definition in the CFRM policy. Having Auto Alter enabled for the structure means that the system can adjust the ratio that was initially requested by MQ and use a ratio that is more suitable based on the sizes of messages that are actually being placed in the structure.

Additionally, as the use of MQ shared queues grows, the number of messages placed in the application structure is likely to increase. Naturally, if the structure is holding more messages, then it will need more space. Also, if the messages on one or more queues are not being processed for some reason, the number of messages in the structure can start to build up, again requiring additional space. Auto Alter can also help in these cases by increasing the structure size when a certain threshold (specified on the FULLTHRESHOLD keyword in the CFRM policy) is reached.

6.2.2 Structure backup and recovery

In this section we present some best practices related to backup and recovery for the MQ structures.

Backing up the application structure

In the same way that it is necessary to take regular backups of pagesets to ensure that messages can be recovered in the event of a DASD failure, you should also ensure that regular backups are made of shared queue messages in the CF structures.

MQ provides a command, `BACKUP CFSTRUCT(structname)`, that can be used to perform a backup of the Coupling Facility application list structures. You can use the `BACKUP CFSTRUCT` command to backup persistent messages held in CF structures to queue manager logs. It copies the current contents of the named structure to the log of the queue manager where the command is issued.

You can only backup those structures that are defined in MQ as being recoverable. Persistent messages can only be written to queues defined in recoverable structures. You must take into account that you cannot put persistent messages to a shared queue if the structure that the queue uses is defined in MQ with `RECOVER(NO)`. Either put only non-persistent messages to this queue, or change the structure definition to `RECOVER(YES)`. If you put a persistent message to a queue that uses a `CFSTRUCT` with `RECOVER(NO)` the put will fail with `MQRC_PERSISTENT_NOT_ALLOWED`.

In the unlikely event of a CF failure, persistent messages in a shared queue can be recovered using the `RECOVER CFSTRUCT(structname)` command. The `RECOVER CFSTRUCT` process has a number of phases:

1. Find the most recent structure backup and restore it into the CF. The most recent backup might have been taken on any queue manager in the QSG.
2. Forward-recover the contents of the structure to the point of failure by applying log records. Again, any queue manager in the queue sharing group might have done a `PUT` or `GET` of a persistent message and recorded data about the operation on its log. This means the forward recovery process involves reading log data from all queue managers in the queue sharing group.

In practice, `RECOVER CFSTRUCT` reads the queue manager logs backwards to optimize recovery. There is no need to recover a message that has been written, read, and committed.

3. Tidying up the recovered state of the structure to ensure transactional integrity of transactions running at the time of failure.

Take frequent structure backups

The amount of time that it takes to recover the contents of a CF structure is overwhelmingly governed by the time to read and merge log data from queue managers which touched the structure between the time of the last backup and the time of failure. If structure recovery time is important to you, you should ensure that structure backups are taken regularly and often.

A small example can illustrate the importance of frequent backups. Suppose your queue manager is logging persistent data at an average rate that is half that sustainable by your logging hardware, let us say 8 MBps. You took a structure backup at 08:00 this morning, and the CF failed three hours later, at 11:00. Now let us imagine that all the log data from these 3 hours is available on your fastest disks, that there is no contention, and that you can read at the full sustainable rate, that is, 16 MB per second. In this example, it will take at least 1hr 30mins to recover the CF structure.

The overhead of a structure backup has been designed to be low. Only persistent messages which have been resident on a shared queue for longer than a threshold interval are backed up. The threshold defaults to 30 seconds, but can be specified on the `BACKUP CFSTRUCT()` command with the `EXCLINT()` parameter. Increasing `EXCLINT` will potentially reduce the size of the structure backup.

It is recommended that you take a backup of all your CF structures about every hour, to minimize the time it takes to restore a CF structure. You could perform all your CF structure backups on a single queue manager, which has the advantage of limiting the increase in log use to a single queue manager. Alternatively, you could spread the backup processes across all the queue managers in the queue-sharing group, which has the advantage of spreading the workload across the queue-sharing group. Whichever strategy you use, MQ can locate the backup and perform a `RECOVER CFSTRUCT` from any queue manager in the queue sharing group. The logs of all the queue managers in the queue sharing group need to be accessed to recover the structure.

Ideally, the `CFSTRUCT BACKUP` command would be issued on a lightly-loaded queue manager. If all queue managers are equally busy, the `CFSTRUCT BACKUP` commands should be issued from multiple queue managers to distribute the workload. If all the systems are very busy and the impact (in terms of log I/O or queue manager CPU consumption) of taking a structure backup is too large, consider starting another queue manager whose sole function is to perform structure backups

6.2.3 Peer recovery

To further enhance the availability of messages in a queue sharing group, MQ detects if another queue manager in the group disconnects from the CF abnormally and, where possible, completes pending units of work for that queue manager. This process is known as peer recovery.

Suppose a queue manager terminates abnormally at a point where an application has retrieved a request message from a queue in sync point, but has not yet done a `PUT` for the response message or committed the unit of work. Another queue manager in the queue sharing group can detect the failure, and back out the in-flight units of work that were being performed on the failed queue manager. This means that the request message is put back on to the request queue and is available for one of the other server instances to process, without waiting for the failed queue manager to restart.

If the connection between a z/OS instance and the Coupling Facility fails, the affected queue manager will abend. This situation is also detected by the other queue managers in the same queue sharing group, and all attempt to initiate peer recovery for the failed queue manager. Only one of these queue managers will actually initiate peer recovery, but they all cooperate in the recovery of units of work that were owned by the queue manager that failed. If a queue manager fails when there are no peers connected to a structure, recovery is performed when another queue manager connects to that structure, or when the queue manager that failed restarts. For this reason, we recommend that you try to configure your queue sharing group so that every structure has at least two connectors, and, if possible, those connectors should be located on two CPCs, maximizing the chance of there always being at least one active queue manager still connected to the MQ structures.

Peer recovery is performed on a structure by structure basis and it is possible for a single queue manager to participate in the recovery of more than one structure at the same time. However, the set of peers cooperating in the recovery of different structures might vary depending on which queue managers were connected to the different structures at the time of failure.

When the failed queue manager restarts, it reconnects to the structures that it was connected to at the time of failure, and recovers any remaining unresolved units of work that were not recovered by peer recovery.

Unit of work scope and peer recovery

Peer recovery is a multi-phase process. During the first phase, units of work that had progressed beyond the in-flight phase are recovered; this might involve committing messages for units of work that are in-commit, and locking messages for units of work that are in-doubt. During the second phase, queues that had threads active against them in the failing queue manager are checked, uncommitted messages related to in-flight units of work are rolled back, and information about active handles on shared queues in the failed queue manager are reset. This means that WebSphere MQ resets any indicators that the failing queue manager had a shared queue open for input-exclusive, allowing other active queue managers to open the queue for input.

You will need to consider the unit of work (UOW) scope and peer recovery. UOW scopes include:

- ▶ Local only UOWs: These exist when messages are placed or retrieved, within the scope of a sync point, from private local queues only.
- ▶ Shared only UOWs: These exist when messages are placed or retrieved, within the scope of a sync point, from shared local queues only.
- ▶ Mixed UOWs: These exist when messages are placed or retrieved, within the scope of a sync point, from both private local and shared local queues.

In case of a queue manager failure, peer recovery is not possible for local only UOWs. The failing queue manager must be restarted to recover the local only UOWs.

MQ and Automatic Restart Manager

The MVS Automatic Restart Manager (ARM) provides the ability to automatically restart address spaces, either after an address space failure or a system failure.

We recommend that you always use ARM to restart an MQ address space following a failure of that region.

However, in the case of a system failure, it might not be necessary to restart MQ on another system. Assuming that you have multiple queue servers in the queue sharing group, it should not be necessary to start MQ on another image from the perspective of providing access to the queues. transaction processing capability. However, if you restart a connected subsystem on another system, it might be necessary to also restart an MQ that was connected to that subsystem in order to resolve INDOUBT units of work. If you do not do this, locks associated with these units of work will not be released, potentially impacting other transactions trying to use those resources.

ARM provides the ability to specify a group of address spaces that must all be restarted on the same system in case of a system failure. So, if you use ARM to restart DB2 elsewhere, you should include the MQ address spaces associated with that DB2 in the same group.

Unit of work size and duration

The size and duration of a UOW can have an adverse impact on the running of a system. This is true for both shared queues and private queues. The two main areas that are affected by the UOW size are the availability of messages and the amount of recovery that is required in the event of a failure.

Any messages on a shared queue that have an MQPUT or an MQGET issued for them inside sync point will require space in the CF. They will take up this space from the time of the MQPUT, whether in or out of sync point, until the point of the MQGET and the get has been committed. This means that messages that are being processed by a long-running UOW will be taking up space on the CF. If messages are being placed on the queue to be processed by another UOW, they will not be available to be processed by the UOW until the putting UOW has been committed. The sooner this UOW can be committed, the sooner they will be available to be processed, thus enabling the space in the CF to be freed.

The duration of a UOW can also have a significant impact on the amount of recovery that is required in the event of a failure. If there is a queue manager failure, it will be necessary to read the logs from the start of the oldest UOW that is still active in the queue manager at the point of the failure. The amount of log data that is required for recovery during restart is going to directly influence the time that restart is going to take. In fact, the UOW does not necessarily have to perform much work to adversely affect the restart time. If the UOW performs a single operation in sync point and then does not do any more MQ work for the next hour, it would still be necessary to read the log data for the whole hour even though only a single operation had occurred.

For these reasons, the size and duration of a UOW should be kept to a minimum. In particular, you should avoid prompting a user for input while having a UOW in progress—if the user takes a long time to respond to the request, a long-running UOW might be created.

There are a variety of ways in which the size of a UOW can be reduced. The method that is employed depends on the exact operations that need to be performed. If a new application is being developed, it is important to give consideration to the size of UOW that might be created.

For example, if an application needs to read records off an MQ queue and use the information in the message to update a database, there are two extremes that could be taken. The first would be to read a single message off the queue, update the database and then commit. The other extreme would be to read a message off the queue, update the database and repeat the operation until there are no messages remaining on the queue, and then commit.

The second method might be fine most of the time if the number of messages on the queue is typically small, so a large UOW would not be created. However, if there is an outage of the application processing the queue, there could be a build up of messages. When the application is restarted, if it processes all the messages on the queue, a very large UOW could be created.

The other alternative of processing a single message at a time might not be the best either, as this would mean that the application would be committing frequently, and there is a cost associated with doing a commit. Of course, neither extreme is ideal, and a balance needs to be found between having too short a unit of work, and too long a unit of work.

6.3 General recommendations

In this section we discuss some other general recommendations related to the best practices for WebSphere MQ.

6.3.1 Which non-application queues should be shared

As far as non-application queues are concerned, there are some considerations to bear in mind:

- ▶ You might not want to define your dead-letter queue (DLQ) as a shared queue because of the maximum message size. The maximum supported message size will depend on the MQ version you are running; as of WebSphere MQ V6, MQ can handle messages up to 100 MB on shared queues.
- ▶ 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 might be useful in making some deductions about how the events were distributed across the QSG.
- ▶ If you plan to use shared channels or intra-group queuing, you must define `SYSTEM.QSG.*` queues as shared queues.

6.3.2 Shared queues and clustering

Just as private queues can be defined to be in a cluster, shared queues can also be defined to be in a cluster. The shared cluster queue would be advertised to the cluster by each queue manager in the QSG that is also in the cluster. This means that if there are three queue managers in a QSG that are also in the same cluster, and a shared queue is defined to be in the cluster, there will appear to be three instances of the queue to the rest of the cluster. Even though there appears to be three instances, these all end up with the message being put to the same shared queue.

It should be noted that it is not possible to define a clustered channel to also be a shared channel, and by implication it is not possible to have a cluster repository hosted by the QSG rather than a particular queue manager.

6.3.3 Inbound shared channels and clustering channels

A common question is whether Inbound shared channels should be used in a shared queues environment, rather than clustered channels.

The answer will depend on exactly what is trying to be achieved. In both cases, high availability will be achieved as there are multiple remote instances that can be connected to, so there will not be a single point of failure at the remote end of the channel.

Using clustered channels, each message will be targeted to a specific destination queue manager, but if that queue manager cannot be reached, the message could be redirected to a different queue manager.

In the case of an inbound shared channel, the message is destined for the queue sharing group, the channel will be started into an available queue manager, and all the messages will be delivered down this channel.

Fundamentally, this all boils down to the fact that if clustered channels are being used, then there will be multiple concurrent routes available, whereas with shared channels there will only be a single channel available. Therefore, if ordering is not an issue, using clustered channels should provide greater throughput compared to shared channels.

6.3.4 What to do with non-persistent messages on shared queues after a Queue Manager restart

As discussed at the start of this chapter, non-persistent messages on private local queues are deleted following a restart of a queue manager. However, non-persistent messages on shared queues are *not* deleted from the CF list structures following a restart of a queue manager.

While most applications will probably view this as an advantage, if your applications have a dependency on non-persistent messages being deleted following a restart of a queue manager, you will either need to create a clean up jobs to deal with these unprocessed non-persistent messages, or you will need to change your application program.

Also, it might not simply be a matter of clearing the queue. If the queue also has persistent messages on it, then you might not want these to be deleted (because those persistent messages would not be deleted by a queue manager restart if they were on local queues). One way of ensuring that you remove the unwanted non-persistent messages safely is to disable the queue for puts (this prevents the arrival of any new messages on the queue), then have a program issue an MQGET of all messages off the queue with the MQGMO_SYNCPOINT_IF_PERSISTENT option (this ensures that all persistent messages are retrieved within sync point scope), and then issue a backout (this restores all persistent messages to the queue). You 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 against the queue (and there is only one instance of it at the same time), then you do not necessarily need to use the MQGMO_LOCK option.

6.3.5 Use backout queue

It is highly recommended that the MQ administrator sets the backout parameters in order to allow MQ applications to remove potential “poisoned” messages from the queue, or messages that cannot be processed due to other issues. This is important for both shared queues and private queues, though the symptoms might differ. The parameters are:

- ▶ **BOQNAME('PRT.REQUEST.BOQ')**
The name of the queue to which applications should write messages that have been backed out.
- ▶ **BOTHRESH (3)**
The number of processing attempts for each message.
- ▶ **HARDENBO**
Harden the backout counter to disk when sync point is done.

A poison message is any MQ message that cannot be processed correctly by the receiving application, and is therefore backed out and remains on the application queue. This would typically be where an application removes the messages from the queue inside sync point control and then the application ends abnormally without having completed processing of the message. If the application termination is due to an error with the message, if the error is not handled correctly, the cycle of getting the message and ending abnormally might continue indefinitely. The application needs to be able to detect this situation and respond

appropriately. If the application ignores the backout counter, the first indication of a poison message is that a queue depth begins rising and the getting processes continues to run.

On a shared queue, when there are multiple server processes running, queue depth might not increase dramatically, because when the poison message is being processed, other messages from the queue are being picked up and processed. There is just a rogue message, which keeps getting processed over and over again - eating up CPU cycles, but not stopping work.

In some cases, backing out a message and allowing a few processing attempts is reasonable. For example, if a database row or table is unavailable, it might become available the next time the message is processed.

A well-behaved application will check the backout count on every message read, and if it is non-zero, compare it to the backout threshold defined on the queue. If the count is greater than the threshold, the message should be written to the queue specified in the BOQNAME parameter and committed. Often a Dead Letter Header (MQDLH) is attached to the message to indicate why the message was written to the backout queue.

This can also be an effective technique for any situation where a queue or a structure list might fill, provided that the BOQNAME is a local queue. The application could even use a defined backout queue for this. This option requires some kind of recovery capability to process the messages or move them to the shared queue when it becomes available. A common technique is to apply the MQ DLQ header to each of these message, so that one of the generic DLQ handlers can be used to move the message to the shared queue without having to write a processor for it.

- ▶ Use one backout queue per application, not per queue. One backout queue can be used for multiple request queues.
- ▶ Do not use the queue-manager-defined dead letter queue as the backout queue, because the contents of the backout queue are usually driven by the application. The dead letter queue should be the backout queue of last resort.
- ▶ Do not make the backout queue a shared queue, because backout queues are usually processed in a batch, and therefore any queue depth build-up on these queues might impact other applications using the same CF structure. Therefore, if you are defining a backout queue for a shared queue, you need an instance of the backout queue on every queue manager in the QSG that can host the server application.
- ▶ If a DLH is applied to backout messages, use one of the many dead letter queue handlers to determine if a message should be restored to the original queue. Many customers use the standard handler, or tailor it to meet the needs of the application.

6.3.6 Shared queue definitions

The queue objects that represent the attributes of a shared queue are held in the shared DB2 repository used by the queue-sharing group. You should ensure that adequate procedures are in place for the backup and recovery of the DB2 tables used to hold MQ objects. You can also use the MQ CSQUTIL utility to create MQSC commands that you would then feed to a queue manager to redefine MQ objects, including shared queue and group definitions stored in DB2.

6.3.7 Intra-group queuing

Intra-group queuing (IGQ) now supports large messages, the largest being 100 MB less the length of the transmission queue header. You can perform fast message transfer between

queue managers in a queue-sharing group without defining channels. This uses a system queue called the `SYSTEM.QSG.TRANSMIT.QUEUE`, which is a shared transmission queue. Each queue manager in the queue-sharing group starts a task called the intra-group queuing agent, which waits for messages to arrive on this queue that are destined for their queue manager. When such a message is detected, it is removed from the queue and placed on the correct destination queue. Standard name resolution rules are used, but if intra-group queuing is enabled and the target queue manager is within the queue-sharing group, the `SYSTEM.QSG.TRANSMIT.QUEUE` is used to transfer the message to the correct destination queue manager instead of using a transmission queue and channel.

You enable intra-group queuing through a queue manager attribute. Intra-group queuing moves non-persistent messages outside sync point, and persistent messages within sync point. If it finds a problem delivering messages to the target queue, intra-group queuing tries to put them to the dead-letter queue. If the dead-letter queue is full or undefined, non-persistent messages are discarded, but persistent messages are backed out and returned to the `SYSTEM.QSG.TRANSMIT.QUEUE`, and the IGQ agent tries to deliver the messages until it is successful.

6.3.8 Double hop avoidance

An inbound shared channel that receives a message destined for a queue on a different queue manager in the queue-sharing group can use intra-group queuing to hop the message to the correct destination. There might be times when you want the local queue manager to put a message directly to the target queue if the target queue is a shared queue, rather than the message first being transferred to the target queue manager.

You can use the queue manager attribute `SQQMNAME` to control this. If you set the value of `SQQMNAME` to `USE`, the `MQOPEN` command is performed on the queue manager specified by the `ObjectQMgrName`. However, if the target queue is a shared queue and you set the value of `SQQMNAME` to `IGNORE`, and the `ObjectQMgrName` is that of another queue manager in the queue-sharing group, the shared queue is opened on the local queue manager. If the local queue manager cannot open the target queue, or put a message to the queue, the message is transferred to the specified `ObjectQMgrName` through either IGQ or an MQ channel. If you use this feature, users must have the same access to the queues on each queue manager in the queue-sharing group.

6.3.9 Increase MQ network availability

MQ messages are carried from queue manager to queue manager in a MQ network using channels. You can change the configuration at a number of levels to improve the network availability of a queue manager and the ability of a MQ channel to detect a network problem and to reconnect.

TCP Keepalive is available for TCP/IP channels. It causes TCP to send packets periodically between sessions to detect network failures. The `KAINT` channel attribute determines the frequency of these packets for a channel.

`AdoptMCA` allows a channel, blocked in receive processing as a result of a network outage, to be terminated and replaced by a new connection request. You control `AdoptMCA` using the `ADOPTMCA` channel initiator parameter, set on the queue manager attributes macro.

`ReceiveTimeout` prevents a channel from being permanently blocked in a network receive call. The `RCVTIME` and `RCVTMIN` channel initiator parameters, set on the queue manager attributes macro, determine the receive timeout characteristics for channels, as a function of their heartbeat interval.

Large messages applications

The processing of large messages in an MQ network often necessitates special design considerations (for example, separate channels) that should be reviewed before simply transferring an application architecture to a parallel implementation. For example, in an image archiving system, immediate message availability might not be of prime importance.

Batch applications

MQ applications designed to gather messages for later processing (that is, batch-oriented applications), can store large amounts of message data before the 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 queues might be more appropriate.

Migrating to shared queues

For real time applications, the amount of message data that might arrive during an application outage (planned or unplanned), and the fact that messages might be trapped, needs to be considered. A solution that uses shared queues and supports parallel servers might be more suitable in this case. 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 might be appropriate to leave these unchanged.

Old data in queues

It is a good idea to periodically review the CSQE1211I message provided by the queue manager when the structure backup is finished:

```
CSQE121I +QMP1 CSQELBK1 Backup of structure CSQAPPL completed at RBA=0051AE81B541, size 5 MB.
```

You might detect a constant size which is a possible indicator that old data is on the queues and not being processed. This results in wasted CPU and adds unnecessary work to the backup process.

Shared queues message management improvements in CFCC level 16

CF Level 16, delivered on System z10® servers, changes the way workload balancing works across the members of the queue sharing group.

Prior to CF Level 16, when the state of a shared queue changed, such as when it gets a new message, the CF would notify all the message queue connectors.

The first one to respond would read the message, while the others would attempt to read, but find the message had already been read. As a result, the system that could respond in the least time would tend to get more than its “fair share” of the work.

With CF Level 16, the CF will notify a single message queue connector about the state change, then wait 5 milliseconds for that connector to retrieve the message. If the message is not retrieved in that time, the CF fall back to the “old” way and notify all connectors. When the next state change occurs, the CF will first notify the next connector to express an interest. This has the effect of implementing a more round robin-like workload balancing mechanism.

z/OS APAR OA30994 lets you change the amount of time that the CF will wait for a connector to respond. If you prefer a round robin type of workload balancing, you should accept the default interval. If you prefer to previous style, which tended to send different amounts of work to different connectors based on their performance, then change the SUBNOTIFYDELAY keyword on the structure definition in the CFRM policy.

Defining subsystems to each other

In a Parallel Sysplex designed for high availability, you want to have the ability to quickly and easily move work and work managers from one member of the sysplex to another. One thing that you can do to enable this is to define the connections between subsystems in a way that will allow any member of one subsystem group to communicate with any member of another subsystem group.

So, for example, CICS TS 4.1 added the ability to specify the QSG name in the MQCONN definition, meaning that any CICS TS 4.1 region could connect to any member of that queue sharing group, giving you the ability to move CICS regions from one system to another and still be able to communicate with MQ with no changes required.

Similarly, in the QSGDATA parameter on the CSQ6SYSP macro, you can specify either the name of a specific DB2 subsystem, or the DB2 group attach name. We recommend that you always specify the DB2 group attach name on this parameter.



WebSphere Application Server sysplex best practices

This chapter discusses the special considerations and recommendations for deploying and running WebSphere Application Server in a data-sharing Parallel Sysplex.

WebSphere Application Server on System z has some of the same features and functions of WebSphere Application Server on other platforms. One of the advantages on z/OS is very high levels of scalability and availability that come from the data-sharing Parallel Sysplex architecture. The recommendations in this chapter provide guidance for realizing the possible advantages of running WebSphere Application Server in a Parallel Sysplex.

Before reading this chapter, read Chapter 1, “Introduction” on page 1, and Chapter 2, “z/OS sysplex best practices” on page 9, to find the set of hardware and z/OS recommendations that apply to all subsystems running in a Parallel Sysplex. This chapter assumes that fundamental Parallel Sysplex best practices, such as having no single points of failure, rolling-in changes, and exploiting z/OS Workload Manager (WLM) workload balancing, have all been implemented.

This chapter is divided into the following categories:

- ▶ WebSphere Application Server for z/OS configuration
- ▶ Application server
- ▶ System z hardware
- ▶ z/OS components

7.1 WebSphere Application Server configuration

The recommended WebSphere Application Server for z/OS configuration depicted in Figure 7-1 exploits the unique advantages of the Parallel Sysplex. It is an example of the WebSphere Application Server Network Deployment. With Network Deployment, you can build a multiple-system, multiple-application-server WebSphere Application Server configuration, with a single point of administration, taking full advantage of Parallel Sysplex.

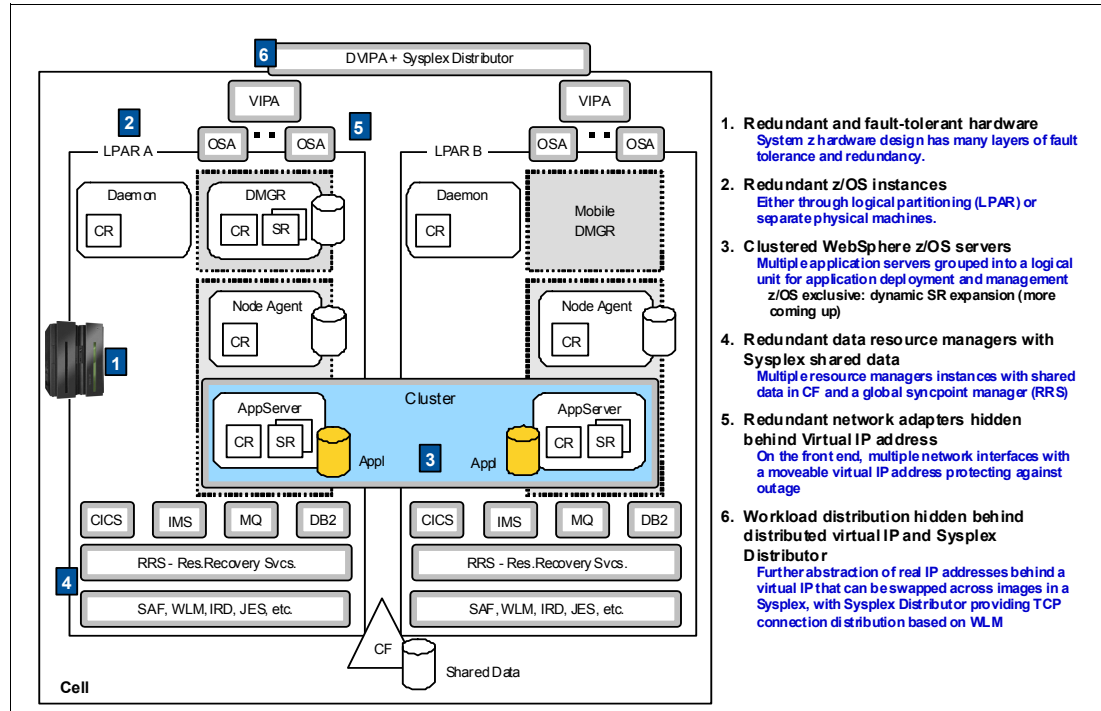


Figure 7-1 Recommended Network Deployment configuration for Parallel Sysplex

In this environment, you integrate one or more application servers into a cell that is managed by a deployment manager (DMGR). It is possible to have more than one cell in a Parallel Sysplex. For example, a test sysplex might have different cells for different levels of pre-production testing.

Within the cell, you can cluster application servers to allow for workload management and failover capabilities. Applications that you install in the cluster are replicated across the application servers that are members of the cluster. If one member fails, another application server in the cluster continues processing. Figure 7-1 shows an application server cluster deployed across two z/OS LPARs (ideally on separate System z CPCs).

A node, which is typically associated with one LPAR, is a logical grouping of at least one instance of an application server and its component processes, which all share common configuration and operational control data. There is one node agent for each z/OS node that manages administration and configuration changes. The application server operations are not dependent on the node agent being available at all times.

There is one daemon per LPAR per cell. The daemon provides the location service that resolves the location of objects in the cell. For WebSphere Application Server for z/OS, it is also the central point of cross-memory communication in an LPAR and provides local WebSphere Application Server components access to key z/OS services. If the daemon stops, all the servers on that z/OS system stop. When the MVS STOP command is used to

stop the daemon, in-flight work is allowed to complete before the daemon stops. This makes it possible for a cell that spans two or more z/OS images to have maintenance applied non-disruptively. Stopping the daemon on LPAR A does not affect the servers on LPAR B.

A resource manager (RM) is a subsystem or component such as CICS, IMS, or DB2 that manages resources that can be involved in a transaction. Resource managers can be categorized as work managers, transaction managers, data resource managers, and communications resource managers. They are often referred to as Enterprise Information Systems (EIS). Over the past several years these subsystems have been developed to fully exploit the Parallel Sysplex to achieve near-continuous availability and high scalability.

The fact that there are multiple WebSphere Application Server instances can be hidden from the user by the use of a Dynamic Virtual IP Address (DVIPA) that can be swapped across the LPARs. Sysplex Distributor provides TCP connection distribution among the LPARs based on z/OS Workload Manager (WLM) policies.

7.1.1 Configure a mobile deployment manager

In a System z server configuration like the Parallel Sysplex depicted in Figure 7-1 on page 158, a cell can consist of multiple nodes that are all administered from a single point. The configuration and application files for all nodes are centralized into a cell master configuration repository.

This centralized repository is managed by the deployment manager and synchronized with local copies that are held on each of the nodes. The deployment manager is critical to the administration of each cell, but application server operations are not dependent on it being available at all times.

A *mobile* deployment manager is one that is capable of being started on separate z/OS images in the sysplex. This provides higher availability because it allows you to maintain a deployment manager instance even when z/OS images need to be stopped. A mobile deployment manager is considered a best practice when configuring a network deployment configuration that spans multiple z/OS images in a sysplex.

The key to this is making use of Sysplex Distributor and the VIPADistribute function of TCP/IP so that the deployment manager can be started on another LPAR and still be accessible. Without that, the deployment manager is locked to the LPAR-specific IP host name where it was first built. This means that when it is moved to the other LPAR, it fails to start.

The prerequisites to support a mobile DMGR are:

- ▶ The use of Sysplex Distributor and DVIPA. This is what allows traffic destined for the deployment manager to be routed to whatever z/OS image that the deployment manager is currently started on.
- ▶ The use of VIPADistribute statements in the TCP Profile. This allows Sysplex Distributor to know what ports to watch for.
- ▶ The PROCLIB containing the deployment manager JCL must be accessible from those systems on which the deployment manager can be started.
- ▶ Either a shared configuration file system for the deployment manager's node, or the ability to move that configuration file system to the other z/OS image. Specifying the AUTOMOVE parameter on the mount statement makes the file system automatically mount on a separate LPAR, if necessary. The deployment manager must have access to its configuration file system for it to start.
- ▶ A configured and started daemon instance on each z/OS image.

7.1.2 Cell and node isolation

In general, whenever multiple WebSphere Application Server environments are present, the need to isolate them from one another is also present. One activity that brings this issue to the forefront is maintenance. Node isolation is necessary to do nondisruptive rolling maintenance by upgrading one node at a time. The following list is a summary of recommendations for achieving node isolation:

- Have a separate configuration file system (HFS or zFS) for each node.

Each node has a directory structure that contains the information about the node and its servers. While it is not technically necessary, we recommend that rather than combining the node information for multiple nodes into one file system, each node have its own file system. This makes backup and restore more granular.

- Use intermediate symbolic links as aliases between the configuration files and the SMP/E product file system.

As part of a cell deployment, you create a configuration file system that contains the cell, node, and server-specific configuration files. It also contains many symbolic links back to the product file system that contains the SMP/E-installed binaries. While this is done for good reasons, it does mean that if all your nodes are linked to the same product HFS, it becomes impossible to perform code upgrades on a node-by-node basis.

The solution is to use an *intermediate symbolic link*, which provides two layers of aliasing between the configuration file system and the product file system. The result is two symbolic links:

- One in the configuration HFS pointing to the intermediate link
- The intermediate link then pointing to the product HFS

The intermediate link can be specific to a node or to a group of nodes if you want to apply maintenance on a group basis.

This allows you to bring a new copy of the product file system into the environment and make nodes reference this new file system by simply changing that node's intermediate symbolic link. This mechanism provides a capability similar to the Symbolic Alias Facility that can be used with traditional non-VSAM data sets.

This approach is discussed in the IBM Washington Systems Center white paper available on the web at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/webindex/wp100396>

- The JCL procs for a node should have an LPAR identifier to make them unique. Or, with proper use of system symbols, it is possible to make the procs LPAR-independent and still applicable across the sysplex.
- If using a version prior to WebSphere Application Server for z/OS V7, use data set aliases.

7.2 Application server recommendations

The Parallel Sysplex Network Deployment configuration uses application server clustering to enhance workload distribution. A cluster is a logical collection of application server processes that provides workload balancing and high availability. Application servers that belong to a cluster all have identical application components deployed on them.

As shown in Figure 7-2, with WebSphere Application Server for z/OS, the application server design is unique among WebSphere Application Server offerings. Similar to IMS, it is structured as a multiple address spaces subsystem. It has a control region where the authorized code runs and multiple servant regions where the application code runs. These server regions are dynamically started based on installation workload management policies for throughput and response time by WLM services.

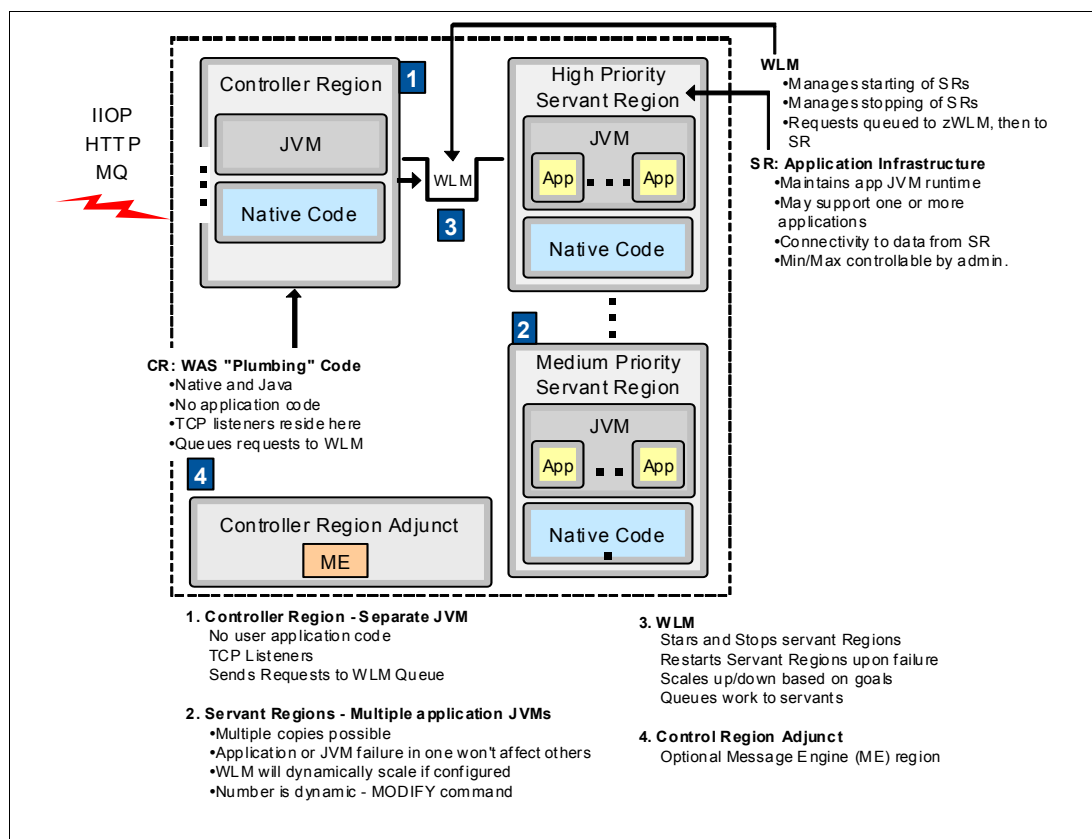


Figure 7-2 z/OS Application Server design

7.2.1 Determine the optimal number of servant regions

The use of multiple servant regions provides transaction redundancy and scalability. If a servant region fails, another can handle the work of the next transaction. If the workload increases or decreases, WLM can transparently increase or decrease the number of servant regions to meet the need.

In the WebSphere Application Server Administration Console General Properties dialog, the default setting for multiple instances is min=1, max=1. Increase these so that there are at least two servant regions and there is the ability for WLM to expand and decrease the number of them as the workload changes. To determine the optimal number of servant regions, take into account that additional resources will be needed, and restart times might be elongated.

7.2.2 Monitor JVM heap use

Each control region and each servant region contains a JVM. These JVMs have special purposes. The controller region JVM is used for communication with the outside world, as well as for certain base WebSphere Application Server services. The servant region JVM executes the user application. This specialization reduces the maximum amount of heap

storage required for the various heaps because not all data and metadata needs to be loaded and kept inside memory. It also separates the user data from most of the system data needed to run the WebSphere Application Server base services.

Usually the z/OS version needs smaller maximum heap sizes for any one JVM than the distributed version because it has specialized heaps in its structure.

However, it is critical that the heaps defined in a WebSphere Application Server environment fit in real memory. The impact of not having the heap in real memory is a negative performance impact due to paging during each garbage collection. The garbage collection for a JVM heap requires all pages of the heap to be in exclusive access and in real memory. If any pages are not in real memory, they first need to be paged in.

Make sure that the LPAR that is used for the installation has enough real memory defined. Also, remember to add storage for the operating system and other applications such as CICS and DB2 that might be running in this LPAR.

When migrating an application from another platform to WebSphere Application Server for z/OS V7.0, the memory size from the distributed environment is often carried over from the distributed environment and reused for the controller and servant regions settings. Not only is this a waste of memory resources, it can also affect performance. If the heap is sized too large, then the garbage collection runs less often, but when it runs, it takes more time. This might reduce the general throughput.

Important: If an application is migrated to WebSphere Application Server for z/OS V7.0 from another operating system family, perform a verbose garbage collection analysis. This allows you to size the heap to a good minimum and maximum value, so that the performance is good and no resources are wasted.

7.2.3 Exploit the rollout update option

When you install, update, or delete an application, the updates are automatically distributed to all members in the cluster. As of WebSphere Application Server V6, there is a rollout update option that allows you to update and restart the application servers on each node, one node at a time, providing continuous operations of the application when planned application changes are made.

Alternatively, you can manually control when updates made in the master configuration are synchronized out to the node. This means that you can perform an update rollout across your cluster by manually controlling when each node is synchronized. Upon synchronization, the updated application is copied out to the node and refreshed in the server.

7.2.4 Configure DB2 data sharing when using WebSphere solutions

WebSphere Portal, WebSphere Process Server, and WebSphere Enterprise Server Bus (ESB) are examples of solutions that run as WebSphere Application Server applications and store configuration information in DB2. With data sharing, another DB2 member in the sysplex can access the data if a DB2 instance fails. Alternatively, if the WebSphere Application Server application instance is stopped, or restarted on another LPAR, the surviving application instance still has access to its configuration data.

7.2.5 Configure WMQ as the JMS provider and implement queue sharing

WebSphere MQ uses Parallel Sysplex capabilities to achieve high availability and scalability. Configure it as the JMS Provider. In addition, configure WebSphere MQ in a shared queue configuration.

When you configure WebSphere MQ for shared queues in the Parallel Sysplex, WebSphere MQ messages sent to and from WebSphere Application Server can be physically stored in shared queues inside the CF and available to all the queue managers that are part of the queue sharing group in the Parallel Sysplex. This makes it possible for another MQ Queue manager to continue processing messages if one fails.

7.2.6 Configure EIS regions in same LPARs as WebSphere Application Server for z/OS

One of the long-standing value propositions of WebSphere Application Server for z/OS has been the ability to co-locate the application layer with the data layer in the same z/OS operating system instance. This leverages many of the inherent advantages of the System z hardware and z/OS operating system.

Figure 7-1 on page 158 shows that, in keeping with full exploitation of Parallel Sysplex, there is an instance of the EIS Resource Managers in their relevant sysplex modes on each LPAR. With WebSphere Application Server for z/OS, locate the EIS instances on the same System z server and LPARs as the application servers. This is because of the tight integration of hardware and software technologies in the Parallel Sysplex. Capabilities like specialized System z Application Assist Processors (zAAPs) and System z Integrated Information Processors (zIIPs), high-performance TCP/IP sockets (HiperSockets™), optimized cross-memory and cross system communications (XCF), z/OS Automatic Restart Manager (ARM), and the Coupling Facility (CF) can come into play to achieve very high levels of availability and performance.

System z Dynamic Workload Balancing, REDP-4563, provides recommended configuration options for connecting from WebSphere Application Server for z/OS to CICS, WMQ, DB2, IMS TM, and IMSDB.

The Washington System Center has conducted performance studies that validate and support co-locating WebSphere Application Server for z/OS with the data. Highlights of the report are presented below. The full white paper, titled *The Value of Co-Location*, is available on the web at:

<http://www3.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101476>

The value of co-location of WebSphere Application Server for z/OS and the data falls into two broad headings:

- ▶ Performance and efficiency: Collocation within the same z/OS operating system instance enables taking advantage of cross-memory data transfer, reducing overall request latency, improving overall throughput, and reducing overall CPU utilization through the elimination of network traffic handling.
- ▶ Operational benefits. Collocation within the same z/OS operating system instance allows for things such as the assertion of security identity, maintaining the same thread of execution, and being able to manage within a single Work Load Manager (WLM) classification.

There can be sound business reasons for separating infrastructure components. One reason sometimes cited is high availability, so that the loss of one server instance will not impact all

transactions. But this concern does not apply to a Parallel Sysplex, which can be configured to maintain service despite the loss of a hardware or software component. In addition, there are many shortcomings of a remote location that are often overlooked:

- ▶ Network latency and overhead: Network delays, however small, add up. That is particularly true in high-volume, high-scale solutions. The overhead of SSL encryption, even with hardware assist, is greater than zero. However, if the application can be located in the same system as the data, the network interactions between one and the other, and the cost of those interactions, can be eliminated.
- ▶ Serialization of query parameters: To pass over the network, these need to be serialized. That represents overhead, and in a high-volume, high-scale solution, which can add up to impact overall performance.
- ▶ New thread of execution: Remote calls received by the data system on z/OS require a new thread of execution, which is not true when WebSphere Application Server is co-located. There the thread of execution is carried through, reducing switching overhead.
- ▶ Separate WLM classification: Remote calls eliminate the possibility of having WLM manage the entire request flow under a single classification, thereby reducing manageability and control.
- ▶ Definition of identity aliases outside central repository: Flowing security over a remote connection often requires the definition of a remote security identity. The resulting ID/password definitions are scattered across multiple remote servers, making updates and coordination more difficult.
- ▶ Less efficient two-phase commit sync point coordination: Remote calls often require distributed two-phase commit protocols such as JDBC XA. Those are effective, but not as efficient as the local sync point coordination offered by z/OS Resource Recovery Services, particularly during transaction recovery.

7.3 System z hardware-related recommendations

In this section we discuss hardware components that are unique to System z that can be exploited to improve the performance and availability of WebSphere Application Server on z/OS.

7.3.1 Use System z specialty processors

The System z Application Assist Processor (zAAP) is a processor dedicated to the execution of Java™ and XML work. The System z Integrated Information Processor (zIIP) is a processor dedicated to the execution of selected data and transaction workloads, such as SQL requests running on type 4 connectors to DB2.

These processors are configured through the System z Hardware Management Console (HMC). For an LPAR to have access, they must be included in the LPAR's image profile on the Processor definition page. zAAPs and zIIPs can be dedicated or shared among LPARs. Like CPs, there can be initial and reserved numbers for each type. Initial zAAPs and zIIPs are brought online when the LPAR is activated and IPLed. Reserved zAAPs and zIIPs can be configured ON later, perhaps after they are added to the machine by a concurrent CIU upgrade or temporarily by CBU or On/Off Capacity on Demand.

The main reasons for using a specialty processor in your WebSphere Application Server for z/OS environment are:

- ▶ Reduced software costs. Workload that runs on a specialty engine does not count toward the monthly z/OS software bill.
- ▶ Performance gain. The specialty processors are dedicated to a subset of the work on z/OS, so there is less competition for access to the processor.

Note: If you currently do not have a zAAP or zIIP installed, you can use the Resource Measurement Facility™ (RMF) to identify the amount of CPU seconds that can be run on a specialty processor. For details see the documentation available at the following websites:

<http://www.ibm.com/systems/z/advantages/zaap/resources.html>

<http://www.ibm.com/systems/z/advantages/ziip/resources.html>

7.3.2 Exploit the Coupling Facility to improve logging performance

WebSphere Application Server for z/OS uses an error log to record error information when an unexpected condition or failure is detected within the product's own code

Although the log streams can be written to the CF or to DASD, writing them to the CF is recommended. Doing so helps transactions complete quickly and not require any DASD I/O. The use of Logger staging data sets generally is not required, as the information in the error log stream is transient and the loss of it (if you have a double failure that impacts both the log stream and one or more connected systems) is unlikely to be a problem.

Remember that a CF-based log stream is required to have a single merged log stream that combines the logs of connectors on multiple systems. Also, if the RRS group used by WebSphere spans multiple z/OS systems, CF log streams must be used for RRS.

For more information about optimizing log stream performance and availability, see the IBM Redbooks publication *Systems Programmer's Guide to: z/OS System Logger*, SG24-6898.

Note: Setting up a log stream for WebSphere is necessary if you plan to use automated reporting or a monitoring product like IBM Tivoli Composite Application Manager (ITCAM) for WebSphere Application Server. As an alternative, it is also possible to have the error messages directed to SYSOUT for printing.

7.4 z/OS component recommendations

In addition to the hardware components that are unique to System z, there are also operating system components that are used by WebSphere Application Server for z/OS. This section discusses recommendations for getting the optimum benefit from those components.

7.4.1 Logger offload data set sizes

By default, the size of the System Logger offload data sets is only two tracks. To prevent excessive allocation of offload data sets, the log stream definitions should specify a larger size on the LSSIZE keyword or a data class with a larger allocation amount. Having offload data sets that are too small is a frequent problem. Use the CFSizer tool to help you determine the correct size for your RRS and WebSphere Application Server log streams.

The correct size requires making a trade-off between having a data set that is too large for the amount of DASD free space available and having one that is so small that it forces frequent allocations. On balance, select a size that holds at least 10 offload's worth of log data, while staying within an amount of space that is easily obtainable in your installation. Also, try to select a size that does not hold more than one day's worth of data.

7.4.2 Disable the RRS archive log

The RRS archive log stream contains the results of completed transactions, and multiple archive log records can be generated from one transaction. As a result, the archive log stream can hold large amounts of data, and this data is rarely used. To reduce the overhead of collecting this data on your production system, consider using the SETRRS ARCHIVELOG command to turn off recording to this log stream. Because the information in the archive log stream might be helpful in diagnosing problems, you might decide to not disable the log stream on your test or development systems.

7.4.3 Exploit 64-bit operating mode

With WebSphere Application Server for z/OS V7.0, 64-bit mode is the default setting, and the 31-bit operating mode is deprecated. While virtually all versions of purchased software support the usage of 64-bit, this point might be of concern for user-built applications that are migrated from a 31-bit environment.

The addressing mode (AMODE) is a JCL parameter, introduced with WebSphere Application Server Version 6.1, used with the START command to determine whether the server is started in 64-bit or 31-bit mode.

The AMODE parameter is still supported in V7.0. We suggest that you do not modify the default value. In the procedures that are generated during the installation process, the value 00 is default. This means that the value specified inside the application server's XML files is used for the decision of running 64-bit or 31-bit mode.

If you start the server with, for example, AMODE=64, and the XML files reflect a 31-bit installation (or vice versa), then the server does not start.

Note: Use the default value for the AMODE (AMODE=00) in the startup JCL for the WebSphere Application Server components. Double-check your automation settings.

7.4.4 Use XCF communication for the HA manager

The WebSphere Application Server high availability manager discovery and failure detection protocol checks to see whether a component is alive or whether a new component is available. With WebSphere Application Server for z/OS V7.0, XCF is an alternative protocol that uses less CPU.

Configure XCF as the discovery and failure protocol when running WebSphere Application Server for z/OS V7.0. After enabled, be sure to test that it is working as expected.

7.4.5 Use zFS rather than HFS

zFS does not replace HFS, but it is the z/OS UNIX® strategic file system, and IBM recommends migrating HFS file systems to zFS. The HFS has been stabilized and

development of new functionality is focused on zFS. Beginning with z/OS 1.7, there are no restrictions for file system structures that can be placed in a zFS.

7.4.6 Determine which file systems to make sharable

The ability to share file systems in a sysplex improves availability because the data can be accessed from a surviving LPAR if another LPAR, or file system user, fails, and a cross-system restart is needed. However, initiating I/O from one system to a file system owned by another system has a negative performance impact. There is no negative impact on a node configuration file system being shared, but the WebSphere Application Server instance that writes to it should be running on the same system as the system that owns the file system.

Because of this, the following things should be taken into consideration regarding the file system, regardless of whether HFS or zFS is used:

- ▶ Sharing static information is recommended, but sharing data that is frequently updated across the sysplex can negatively impact performance.
- ▶ The deployment manager configuration files should be sharable to allow for a cross-system restart.
- ▶ Node configuration files should be sharable across the sysplex in case of a cross-system restart.
- ▶ Mount the WebSphere Application Server for z/OS product binaries in a file system that is mounted read only. This improves performance and protects the file system contents.

7.4.7 Implement Automatic Restart Manager (ARM)

WebSphere Application Server for z/OS is able to use the z/OS Automatic Restart Manager (ARM) to recover failed application servers. Each application server running on a z/OS system (including servers that you create for your business applications) is automatically registered with an ARM group. Each registration uses a special element type called SYSCB, which ARM treats as restart level 3, ensuring that RRS restarts before any application server. (WebSphere Application Server for z/OS does not start without RRS being available, and it abends in the event of an RRS failure.) When using ARM, the following actions are recommended:

- ▶ If you have ARM enabled on your system, you might want to disable it for the WebSphere Application Server for z/OS address spaces until your customization and testing of WebSphere Application Server for z/OS is complete. During customization, job errors might cause WebSphere Application Server for z/OS address spaces to abend, and you do not want ARM restarting them before you have a chance to resolve the error. After installation and customization, consider re-enabling ARM for WebSphere Application Server.
- ▶ It is a good idea to set up an ARM policy for your deployment manager and node agents.
- ▶ If you start the location service daemon on a system that already has one in the same cell, it terminates. Therefore, do not enable cross-system restart for the location service daemon.

- ▶ Every other server comes up on a dynamic port unless the configuration has a fixed port. Therefore, the fixed ports must be unique in the sysplex.
- ▶ If you issue STOP, CANCEL, or MODIFY commands against server instances, Table 7-1 shows how automatic restart management behaves regarding WebSphere Application Server for z/OS server instances.

Table 7-1 ARM response to operator commands

If you issue:	ARM will:
STOP address_space	Not restart the address space.
CANCEL address_space	Not restart the address space.
CANCEL address_space, ARMRESTART	Restart the address space.
MODIFY address_space, CANCEL	Not restart the address space.
MODIFY address_space, CANCEL , ARMRESTART	Restart the address space.

7.4.8 Tailor Workload Manager policies

The use of WLM classification for the control and servant region address spaces is unique to z/OS and is part of the installation process of WebSphere Application Server for z/OS V7.0. The following recommendations apply:

- ▶ Assign control regions should a service class with high priority in the system, for example, the highest STC below the SYSTEM and SYSSTC. A high priority is needed because the control regions do some of the processing that is required to receive work into the system, manage the HTTP transport handler, classify the work, and do other housekeeping tasks.
- ▶ The servant classification is used during servant startup and when there is no user work in the servant. Use a high-velocity goal so that it runs suitably fast in these circumstances.
- ▶ Enclaves for WebSphere Application Server for z/OS are classified using the subsystem type of CB. The performance objectives that you specify depend on your application and the environment. However, a response time goal (rather than a velocity goal) is usually advisable.
- ▶ The UNIX System Services components of WebSphere Application Server for z/OS need to be classified as well. Certain UNIX System Services scripts are executed during server startup. Therefore, if these are not classified in the WLM, the server startup time is increased.
- ▶ The control regions, servant regions, and enclaves all should have their own reporting classes to enable you to do performance analysis at a more granular level.
- ▶ For more information about WLM recommendations, see *System z Mean Time to Recovery Best Practices*, SG24-7816.

7.4.9 Use Fast Response Cache Accelerator

WebSphere Application Server for z/OS V7.0 can be configured to use the Fast Response Cache Accelerator (FRCA) facility of z/OS Communications Server. This high-speed cache within TCP/IP can be used to cache static and dynamic contents.

The benefits of using the FRCA are a reduced response time and a reduced CPU cost for the serving of requests, compared to the WebSphere Application Server Dynamic Cache. Tests have shown that a request served from the FRCA used approximately 8% of the processor time that the same request consumed in a Dynamic Cache environment.

7.4.10 Use WebSphere Configuration Tool for z/OS

The WebSphere Configuration Tool (WCT) has a GUI workstation interface and includes the Profile Management Tool for z/OS and the z/OS Migration Management tool. These create, migrate, or augment a WebSphere Application Server for z/OS configuration. This was previously accomplished by the Application Server Toolkit for z/OS (AST) and, previous to AST, by ISPF. Both have been discontinued with WebSphere Application Server for z/OS V7.0.



A

Important sysplex messages

This appendix provides a list of sysplex-related messages that, at the time of writing, you should be aware of if they are issued on your systems. Define these messages to your automation package. It might be possible to have automation automatically reply to the message or to take corrective action based on the message. At a minimum, inform the appropriate staff (perhaps by an SMS text or an email) whenever they are issued.

Note: This is not a comprehensive list. There might be additional messages that are particularly important in your environment. The messages in this section are those that we believe are important to all sysplex customers. Additionally, we have seen customer situations where problems arose because one or more of these messages were issued, but either not seen, not responded to, or ignored.

Automation and critical messages

Very few installations still have a traditional MVS console as their main operations interface to manage their mainframe environment. Much more common is a GUI-based automation front end that uses color coding to warn the operators of situations that require their attention.

Managing a system using a traditional console is not a realistic option for most customers. The rate at which messages are sent to the MVS console, and the rate at which they roll off the console, make it impossible to comprehend every message as it flies past. Therefore, operations that are assisted by an automation tool are a necessity.

The main drawback of any automation product is that it needs to be told which messages are important and which can be ignored. If a system issues an IXL041E message, for example, the automation package more than likely will ignore that message unless someone explicitly included that message in the list of important messages.

The challenge for automation analysts is to identify, from the thousands of pages of message manuals, and tens of thousands of messages, which are the important ones. Even more difficult, how do you keep that list up to date? After all, it is not only new products or even new releases that provide new messages. PTFs can also deliver new messages, modify existing ones, and potentially delete or replace old ones.

The messages provided in this appendix are important enough that you should know that the message was issued. It is important to stress that not all of these messages require an immediate response. Certain ones might be important simply because they impart information that something has changed in your system and you should be aware of that change to effectively manage the system or sysplex.

We also suggest that you include messages from the Health Checker in your list of important messages. While any exceptions detected by Health Checker are shown on the SDSF CK display, if someone is not monitoring that display, it is possible for exceptions to go unnoticed for a while. By including the messages in your automation, you ensure that the appropriate person is notified immediately if an exception condition is detected. You can get a list of the messages that the various health checks generate in *IBM Health Checker for z/OS User's Guide*, SA22-7994.

Finally, what about messages that were delivered as part of an APAR fix and that have not been integrated into the latest messages manual yet? You can get a list of these messages on the LookAt website, available at:

<http://www.ibm.com/systems/z/os/zos/bkserv/lookat/>

Enter the message prefix that you are interested in (IXC*, for example), select **APAR and ++DOC HOLD Docs z/OS**, and click **Go**.

XCF messages

The following messages are issued by XCF and, at a minimum, must be highlighted and brought to the attention of the responsible system programmers:

Message	Brief description
IXC101I	Sysplex partitioning in progress for system nnnnnnn.
IXC102A	Partitioning reset and reply down messages.
IXC105I	Sysplex partitioning has completed for system nnnnnnn.

IXC244E	XCF cannot use sysplex CDS.
IXC246E	Couple data set has been experiencing I/O delays.
IXC255I	Unable to use dsname as the Pri/Alt for nnnn.
IXC256A	Removal of CDS cannot finish until the following systems acknowledge the removal.
IXC259I	I/O error for data set nnnnnn.
IXC267E	Processing without an alternate CDS.
IXC402D	Partitioning reset and reply down message.
IXC409D	Signalling connectivity lost message.
IXC417D	CONFIRM REQUEST TO REMOVE sysname FROM THE SYSPLEX.
IXC418I	System sysname is now active in sysplex plexname.
IXC426D	XCF heartbeat stopped but still sending XCF signals.
IXC427A	System has not updated status, but is sending XCF signals.
IXC430E	System has stalled XCF group members.
IXC431I/432I	Stalled XCF group member.
IXC440E	Stalled member detection.
IXC446I	System hurtsys impacted by stalled members on stallsys.
IXC458I	Signalling CTC device stopped.
IXC459I	Signalling CTC device stopped unconditionally.
IXC467I	Restarting PIPO devname.
IXC501A	Reply Y to use CF (CFRMOWNEDCFPROMPT).
IXC512I	Policy change in progress.
IXC518I	System nnn <i>not</i> using CF cfname.
IXC519E	CF damage recognized for CF cfname.
IXC522I	Rebuild for structure is being stopped.
IXC538I	Duplexing rebuild of structure was <i>not</i> initiated by MVS.
IXC552I/553E	Duplex structure not failure isolated from other instance.
IXC560A	See IXC501A.
IXC573I	Processing during a system-managed process for structure structname encountered an error.
IXC582I	Indicates that allocation size exceeds CFRM policy definitions. Adjust policy values.
IXC585E	Structure above threshold (FULLTHRESHOLD).
IXC588I	Auto alter initiated.
IXC615I	Address space terminated because it was causing sympathy sickness or defined itself as CRITICAL=YES.
IXC631I	XCF member stalled, impacting another system.
IXC633I	Deemed or confirmed impaired XCF group member.
IXC635E	Deemed or confirmed impaired XCF group member.
IXC636I	Deemed or confirmed impaired XCF group member.

IXC640E	Stalled member detection.
IXC700E	Indicates that we hit capacity of sysplex CDS (max members in a group, max groups in the plex).
IXC701I	Indicates IXCJOIN failure due to an environmental condition.
IXC800I	ARM element not restarted.

XES messages

The following messages are issued by XES (that is, they are related to the CFs) and, at a minimum, must be highlighted and brought to the attention of the responsible system programmers:

IXL010E	Notification received from CF cfname.
IXL013I	Structure connect request failed.
IXL040E/41E	Member not responding to confirmation request (usually rebuilds). Addressed by SFM in z/OS 1.12.
IXL044I	Interface control checks on CF link.
IXL045E	Rebuild delay.
IXL049E	Indicates whether SFM will take action against connector identified on IXL040/041E message.
IXL051E	CF Dump was initiated.
IXL158I	Path now not operational to CF.
IXL159E	CF hardware error detected.
IXL160E	Request time ordering needed but not available.
IXL162E	Request time ordering needed but will not be enabled.

System Logger messages

The following are the System Logger messages that we believe are particularly important to monitor for:

IXGH007E	Created in case of a Logger health check exception).
IXGH008E	Created in case of a Logger health check exception).
IXGH009E	Created in case of a Logger health check exception).
IXG054A	Logger couple data set not available.
IXG058E	Log stream connections lost due to Loggerabend.
IXG063I	System Logger has abended. Includes information from the dump title.
IXG067E	Logger was stopped by the operator, or abended and failed to restart successfully.
IXG074I/075E/076I	System Logger services disabled/constrained for GROUP: TESTIPRODUCTION.
IXG105I	Unable to start structure rebuild.
IXG114A	Offload not progressing.

IXG115A/272E/312E	System Logger offload problems. See “Offload and Service Task Monitoring” in <i>z/OS MVS Setting Up a Sysplex</i> , SA22-7625, for the recommended actions.
IXG221I	Insufficient structure storage available. New log stream connection attempts might fail.
IXG222I	Severe storage shortage in structure xxx.
IXG257I	Data set directory for log stream xxx is over 90% full.
IXG261E	Shortage of directory extents.
IXG262A	Critical shortage of directory extents.
IXG263E	Staging or offload data CISZ not a multiple of 4 KB.
IXG267I/268I/269I	Invalid attributes for Logger data set.
IXG271I	Logger data set service task delay.
IXG282I	Invalid VSAM Share Options corrected for new offload or staging data set.
IXG310I	Offload is not progressing.
IXG311I	Offload is not progressing.

Related publications

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

IBM Redbooks publications

For information about ordering these publications, see “How to get Redbooks publications” on page 177. Note that certain documents referenced here might be available in softcopy only.

- ▶ *Architecting High Availability Using WebSphere V6 on z/OS*, SG24-6850
- ▶ *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079
- ▶ *Parallel Sysplex Application Considerations*, SG24-6523
- ▶ *Systems Programmer's Guide to: z/OS System Logger*, SG24-6898

Other publications

These publications are also relevant as further information sources:

- ▶ *CICS Application Programming Guide*, SC34-7022
- ▶ *DB2 V9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845
- ▶ *z/OS MVS Setting Up a Sysplex*, SA22-7625

Online resources

These websites are also relevant as further information sources:

- ▶ *WP100743 Parallel Sysplex Performance: XCF Performance Considerations V3.1*
<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100743>
- ▶ *System-Managed CF Structure Duplexing*
<ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/zsw01975usen/ZSW01975USEN.PDF>

How to get Redbooks publications

You can search for, view, or download Redbooks publications, Redpapers publications, Technotes, draft publications and additional materials, as well as order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

- active log streams 71
- Activity keypoint frequency 97
- addressing directory reclaims 107
- addressing sympathy sickness problems 37
- ADRUENQ exit
 - using to avoid RESERVEs 15
- ALLOWAUTOALTER and CF storage trimming 48
- ALLOWAUTOALTER attribute of structure definition 58
- application server clustering 160
- Application Server Toolkit 169
- Auto Alter 104, 106
 - considerations for lock structures 104
 - recommended structures 59
- Automatic Restart Manager
 - restart groups 113
 - use with DB2 112
- Automatic Restart Manager, and WebSphere Application Server 167
- automation 4, 12
- automation for Couple Data Set failures 12
- Automation reminder 105
- AUTOREC option for GBPs 103
- avoiding RESERVEs on the XCF CDS volumes 15

B

- BCPii interface 34
- benefits of multiple OS Configs 7

C

- cache structure cross invalidates 66
- calculating CF CPU usage by structure 47
- CASTOUT(NO) option for DB2 restart 114
- CDS format levels 17
- CDS naming conventions recommendation 12
- CDS sizes 18
- CDS usage information
 - BPXMCDS 20
 - CFRM 21
 - LOGR 21
- CF
 - avoiding single points of failure 100
- CF access list io IODF 69
- CF CPU utilization 45
- CF CPU utilization recommendations 45
- CF Dump space 49
- CF failure isolation
 - options 41
- CF Level 16, enhancements that affect IMS 137
- CF Levels 44
- CF links 49–50
- CF maintenance 53
- CF Path Busy recommendations 50

- CF Service Levels 45
- CF SHUTDOWN command 53
- CF storage capacity planning 48
- CF storage trimming 48
- CF storage utilization 48
- CF structure sizes 57
- CF subchannel utilization 50
- CF volatility 52
- CFDETAIL parameter of RMF 44
- CFRM CDS usage information 21
- CFRM MINSIZE value 107
- CFRM policies 40
- CFRM policy
 - for DB2 GBP structures 103
- CFRMOWNEDCFRPROMPT parameter 69
- CFSizer tool 30, 57, 104
- CFSizer, and DB2 SCA structure 107
- CFSTRHANGTIME function in SFM 62
- checking offload activity for log streams 71
- CI Size for Logger data sets 72
- CICS
 - affinities 83
 - obtaining unique counter values 86
 - temporary storage 84
 - affinity categories 83
 - CF data tables 88
 - CF data tables recovery 88
 - connection to DB2 93
 - connection to MQ 93
 - data tables 87
 - dynamic transaction routing 83
 - ENQ/DEQ capability 89
 - log stream recommendations 95
 - mechanisms to address affinities 84
 - named counter facility 86
 - named counter recovery 86
 - naming conventions 98
 - potential benefits from Parallel Sysplex 82
 - Queue Owning Region 84
 - region cloning 98
 - temporary storage 84
 - temporary storage server address space 85
 - Temporary Storage structure availability 85
 - use of DB2 group name 120
- CICS session balancing 90
- CICS-maintained data tables 88
- CICSplex Systems Manager
 - queue algorithm 91
- CLASSLEN values 32
- CLASST parameter in DB2 106
- CLASST recommended values 106
- CLEANUP keyword 36
- coexistence across multiple releases 6
- commits in DB2 117
- CONDBAT DB2 parameter 118

- CONFIGxx member in parmlib 3
- CONNFAIL function in SFM 39
- converting RESERVEs 127
- converting RESERVEs to ENQs 15
- Couple Data Set
 - adding a new one 11
 - ARM 19
 - BPXMCDS 20
 - CFRM 21
 - format levels 17
 - ARM 17
 - BPXMCDS 17
 - CFRM 17
 - LOGR 17
 - SFM 17
 - Sysplex 17
 - WLM 17
 - formatting 24
 - LOGR 21
 - naming convention 12
 - recommendation 11
 - separation 13
 - SFM 22
 - spare 12
 - Sysplex 35
 - sysplex 18
 - WLM 18, 22
- Couple Data Sets 11
 - best practices for DR 68
 - naming convention 15
 - placement 14
- Coupling Facility Data Tables 88
- CPU cost
 - of using System-Managed Duplexing 41
- CQS structure checkpoints 134
- CTC recommendations 26
- CURRENTDATA bind option in DB2 116

D

- DASD performance considerations for IMS 139
- DB2
 - ALLOCATE command 109
 - archive log media recommendation 122
 - Auto Alter recommendations 104
 - batch jobs 110, 120
 - commit considerations 117
 - considerations for affinities 120
 - CURRENTDATA recommendations 116
 - directory reclaims 107
 - DSNZPARM, changing without restarting DB2 123
 - dynamic workload balancing 121
 - failure scenarios 112
 - GBP checkpoint 109
 - GBP checkpoint frequency 108
 - GBP duplexing options 103
 - GBP INITSIZE recommendations 107
 - GBP recovery considerations 103
 - GBP structure full condition 105
 - GBP structure sizes 107
 - GBP structures and duplexing 102

- GENERIC LUNAME parameter 121
- Group Attach Name 120
- group attach name 120
- incremental image copy considerations 118
- ISOLATION recommendations 116
- Lock and SCA isolation requirements 101
- Lock and SCA use of System-Managed Duplexing 102
- lock avoidance recommendations 117
- lock contention 114
- Lock structure recovery considerations 101
- Locking Protocol 2 117
- LOCKSIZE recommendations 116
- other sources of information 100
- release coexistence 122
- restart considerations 112, 118, 122
- Restart Light option 112
- restart, considerations for indoubt units of recovery 113
- restart, use of CASTOUT(NO) 114
- running multiple DB2s in the same z/OS system 120
- SCA structure and Auto Alter 104
- SCA structure size 107
- statistics recording 110
- structure availability 100
- sysplex query parallelism feature 121
- system checkpoint frequency 118
- toleration service 122
- use of VTAM Generic Resources 121
- WLM service class considerations 119
- workload balancing 119, 121
- DB2 GBPs use of duplexing 66
- DB2 Recommendations 99
- DB2CONN definition in CICS 93
- DBRC recommendations 127
- DCFD 43
- DCFD recommendations 44
- DEADLOCK recommended setting 130
- DEFAULT transport class 33
- deployment manager in WebSphere Application Server 159
- DFSMSdss 15
- disaster recovery considerations 15
- disaster recovery considerations for sysplex 68
- DSNZPARM values, changing dynamically 123
- DUMPSPACE 49
- DUPLEXCF16 42
- DUPLEXMODE(COND) keyword in Logger 70
- DWQT parameter in DB2 106
- Dynamic CF Dispatching 44
- Dynamic Virtual IP Address, use with WebSphere Application Server 159
- dynamic workload balancing 121

E

- ENQMODEL definitions in CICS 89
- ERBRMF04 member 44
- exclusion lists for XCF structures 27
- expected structure response times 55

F

- Failure Detection Interval 35, 38
 - relation to spin loop recovery processing 35
- failure-isolated CFs 101
- failure-isolation for CFs 41
- false contention 64
 - decreasing 114
 - identifying in RMF 115
- false contention threshold 114
- Fast Response Cache Accelerator 168
- FDBR recommendations for IMS recovery 131
- FULLTHRESHOLD and Auto Alter 106
- FULLTHRESHOLD, specifying threshold for structure full monitoring 58
- funnel log streams 71

G

- GBP checkpoint 108
- GBP dependency 109
- GBPOOLT parameter in DB2 106
- GBPOOLT, relationship to FULLTHRESHOLD value 106
- GDPS considerations 10, 13, 28, 33, 39
- GRECP status 103
- GROUP attribute, in log stream definition 71
- GRS ENQ/RESERVE/DEQ monitor 15
- GRS Reserve Conversion RNL 15
- GRS RNLs 15
- GRS_CONVERT_RESERVES health check 16

H

- Health Checker for z/OS 10
- how many CFs to have 40
- how many sysplexes to have 67
- HWSHWS00 PPT entry 138
- HyperPAV 15

I

- impact of distance between CFs
 - on System-Managed Duplexing 41
- Important messages 72
- important messages 12, 36–38, 58–59, 62, 72–73, 105, 132, 134, 172
- IMS
 - addressing false scheduling 137
 - application recommendations 133
 - BMP restart considerations 132
 - connectivity considerations 138
 - considerations relating to DASD performance 139
 - data sharing 126
 - database cache structures 128
 - duplexing database data sets 127
 - Fast Path DEDBs 128
 - Fast Path structure recommendations 129
 - FDBR recommendations 131
 - introduction 126
 - miscellaneous database-related recommendations 131
 - OSAM structure recommendations 128

- role of DBRC 126
- role of IRLM 126
- shared message queues 133
- shared queue recommendations 135
- System Logger recommendations for shared queues 136
- using HALDBs to assist database availability 128
- using MADS to protect data set availability 127
- VSAM structure recommendations 128

- Infiniband links 50
- INITSIZE CFRM policy keyword 57
- inter-transaction affinity 83
- inter-transaction affinity in CICS 83
- IOSSPOF system service 4
- IOSSPOFD batch job 4
- IOSSPOFD program 13
- IPL frequency 5
- IRLM parameters recommendations 130
- IRLMNM value 130
- ISC links 51
- ISOLATETIME parameter in SFM 35, 37
- ISOLATION bind keyword 116
- IXC267E message 12

J

- JVM heap use 161

L

- lock avoidance in DB2 117
- lock contention 64
- Locking Protocol 2 117
- Log defer interval 96
- Logger CDS
 - considerations for disaster recovery 14
- Logger data sets 14
- Logger staging data set size 76
- LOGR CDS 14, 21
- long-running CF requests 65

M

- MAINTMODE function for CF 53
- MEMSTALLTIME 37
- MEMSTALLTIME option in SFM 37
- MEMSTALLTIME value recommendations 37
- migrating applications to WebSphere Application Server for z/OS 162
- mirroring Couple Data Sets 14
- mobile deployment manager
 - prerequisites 159
- MQ Shared Queue
 - requirements 142
- MQ Shared Queues
 - benefits 142
- MQCONN resource definition in CICS 93
- MSGBASED CFRM processing 16
- multi-site sysplex CDS placement considerations 13
- multi-site sysplex considerations 13, 26

N

Named counters in CICS and batch 86

O

offload data set attributes 72

P

partitioned DB2 tablespaces 115
path busy events 49
PCLOSEN DB2 parameter 109
PCLOSET DB2 parameter 109
peer recovery 75
PR/SM time slice 44
PREFLIST, using to control structure placement 60
primary and alternate Couple Data Sets 11
proactively 3
procedure for formatting new Couple Data Sets 11
procedure for shutting down a CF 53
protecting log data 70

R

real contention 64
 addressing 115
real contention threshold 114
REALLOCATE command 53, 60
REBUILDPERCENT on structure definition 60
recon data sets 127
recovering damaged GBP 108
Redbooks Web site 177
 Contact us xi
reducing DB2 data sharing overhead 118
Register Name List 65
remote mirroring
 considerations for Couple Data Sets 14
REQ REJECT 28
RESERVEs 15
retained locks, in DB2 118
RETLWAIT DB2 parameter 118
RMF CF reporting 44
RMF CF Subchannel Activity report 49–50, 56
RMF Coupling Facility Structure Activity report 61
RMF Coupling Facility Usage Summary report 47, 66
RMF interval, synchronization with DB2 statistics 110
RMF Monitor III 44
RMF Spreadsheet Reporter 31, 46, 56
RMF structure detail report 115
RMF XCF Path Statistics report 26, 30
RMF XCF Usage by System report 28–29
role of automation in identifying single points of failure 3
RRS
 recommendations for use with IMS 137

S

SCA structure size 107
servant regions 161
SFM CDS 22
SFM policy 35

shared engines
 comparison to dedicated engines 43
shared engines in production CFs 42
SHAREOPTIONS for Logger data sets 72
SHUTDOWN command for CF 53
single points of failure 2–4, 26
 for critical DB2 data sets 122
 identifying with IOSSPOF service 122
SIZE CFRM policy keyword 57
SMF Type 88 records 76
SMS ACS routines 15
SMS striping for Logger staging data sets 73
spare Couple Data Sets 11
 placement 13
spin loop recovery 35–36
SPINRCVY value 35
SPINTIME value 35
SSUMLIMIT 38
SSUMLIMIT function in SFM 38
staging 76
STATIME DB2 parameter 110
STOP DB2 command 114
structure ALTER compared to structure REBUILD 104
Structure Full Monitoring 57, 104
 limitations 58
structure INITSIZE 106
structure SIZE 106
SVC dump recommendations 119
sympathy sickness 37
SYNCVAL Db2 parameter 110
Sysplex CDS 35
Sysplex Distributor, use with WebSphere Application Server 159
Sysplex Failure Management 37
Sysplex query parallelism 121
sysplex separation 6
sysplex testing 68
system CLEANUP value 36
System Logger 70, 75
 GROUP attribute in log stream definition 71
 offload data set sizes 72
 recommendations for IMS 136
 relation to CF volatility status 70
 staging data set attributes 72
System Logger health checks 73
System Status Detection Partitioning Protocol 34
system weights 39
System z Application Assist Processor 164
System z Integrated Information Processor 164
System-Managed Duplexing 41, 102, 131
 and DB2 structures 100
 for CICS named counter structure 86
 recommendation to only use with dedicated engines 42
 use with CICS temporary storage structures 85
System-Managed Duplexing performance 41

T

Temporary Storage pools in CICS 84
test and production sysplexes 67

- Tivoli System Automation 12
- TRACKMOD parameter in DB2 118
- Transaction-system affinity 83
- transport class 26
- transport classes 32

U

- Uncommitted Read option in DB2 116
- UNIX file system recommendations 166
- URCHKTH DB2 parameter 117
- URLGWTH DB2 parameter 117
- use of dedicated engines for production CFs 42
- use of exclusion lists 60
- use of shared engines for CFs 43
- use of staging sets 72
- user-maintained data tables 88
- USS file system sharing
 - considerations for WebSphere Application Server 167

V

- VDWQT parameter in DB2 106
- VTAM Generic Resources 121

W

- WebSphere Application
 - Server Network Deployment example 158
- WebSphere Application Server
 - Addressing Mode 166
 - application server clustering 160
 - application server design 161
 - Application Server Toolkit 169
 - cell description 158
 - colocation with EIS resource managers 163
 - daemon 158
 - deployment manager 159
 - file system recommendations 160
 - High Availability Manager 166
 - implementing maintenance nondisruptively 160
 - introduction 157
 - node description 158
 - real memory requirements 162
 - recommended configuration 158
 - Rollout Update option 162
 - software maintenance 160
 - support for z/OS Automatic Restart Manager 167
 - System Logger recommendations 165
 - use of symbolic links to control maintenance 160
 - using WebSphere MQ as the JMS provider 163
 - WLM recommendations 168
- WebSphere MQ
 - use with WebSphere Application Server 163
- WLM
 - recommendations for WebSphere Application Server 168
- WLM and WebSphere Application Server servant regions 161
- WLM CDS 22

- WLM considerations for XCF exploiters 29
- WLM Scheduling Environments 110, 120
- WLM service class, relation to XCF buffer usage 38
- WLM service classes 119
- Workload balancing 119

X

- XCF 10
 - alternate couple data set requirements 11
 - automation for CDS failures 12
 - best practices recommendations 10
 - CLASSLEN values 32
 - cleanup interval 36
 - Couple Data Set naming convention 12
 - Couple Data Set placement 13
 - Couple Data Sets 11
 - Couple Data Sets as a Single Point of Failure 11
 - management of spare Couple Data Sets 11
 - message sizes 32
 - message transfer time 30
 - message transfer times 30
 - role in a sysplex 10
 - signalling buffers 28
 - signalling path types comparison 26
 - signalling paths 25–26
 - signalling paths types 25
 - signalling structures 30, 34
 - spare Couple Data Sets 11
 - stalled members 38
 - transport class connectivity 26
 - use of alternate Couple Data Sets 11
 - use of CTCs for signalling paths 26
- XCF exploiters 25
- XCF health check 103
- XCF health checks 100
- XCF message buffers 37
- XCF resiliency 10
- XCF_CDS_MAXSYSTEM health check 24
- XCF_CDS_SEPARATION health check 15, 27–28
- XCF_CDS_SPOF health check 11, 13
- XCF_CF_ALLOCATION_PERMITTED health check 54
- XCF_CF_CONNECTIVITY health check 49
- XCF_CF_MEMORY_UTILIZATION health check 49
- XCF_CF_PROCESSORS health check 43, 57, 60
- XCF_CF_STR_AVAILABILITY health check 60
- XCF_CF_STR_DUPLEX health check 67
- XCF_CF_STR_NONVOLATILE health check 53
- XCF_CF_STR_PREFLIST health check 60
- XCF_CF_SYSPLEX_CONNECTIVITY health check 40
- XCF_CFRM_MSGBASED health check 16
- XCF_CLEANUP_VALUE health check 36–37
- XCF_DEFAULT_MAXMSG health check 29
- XCF_FDI health check 36
- XCF_MAXMSG_NUMBUF_RATIO health check 29
- XCF_SFM_ACTIVE health check 37
- XCF_SFM_CFSTRHANGTIME health check 62
- XCF_SFM_CONNFAIL health check 28, 39
- XCF_SFM_SSUMLIMIT health check 39
- XCF_SFM_SUM_ACTION health check 37
- XCF_SIG_STR_SIZE health check 30

XCF_SYSPLEX_CDS_CAPACITY health check 19
XCF_SYSSTATDET_PARTITIONING health check 35
XCF_TCLASS_CLASSLEN health check 33
XCF_TCLASS_CONNECTIVITY health check 34
XCF_TCLASS_HAS_UNDESIG health check 33
XCFAS address space 15
XES heuristic algorithm 43, 56
XES Subchannel Tuning 50
XES subchannel tuning 50
XES, role in a sysplex 10

Z

z/OS Health Checker 78



System z Parallel Sysplex Best Practices



Redbooks®

Optimization of sysplex performance and availability

Hints and tips for major subsystems

Considerations for z/OS

This IBM Redbooks publication pulls together diverse information regarding the best way to design, implement, and manage a Parallel Sysplex to deliver the levels of performance and availability required by your organization.

This book should be of interest to system programmers, availability managers, and database administrators who are interested in verifying that your systems conform to IBM best practices for a Parallel Sysplex environment. In addition to z/OS and the sysplex hardware configuration, this book also covers the major IBM subsystems:

- ▶ CICS
- ▶ DB2
- ▶ IMS
- ▶ MQ
- ▶ WebSphere Application Server

To get the best value from this book, readers should have hands-on experience with Parallel Sysplex and have working knowledge of how your systems are set up and why they were set up in that manner.

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

SG24-7817-00

ISBN 0738434671