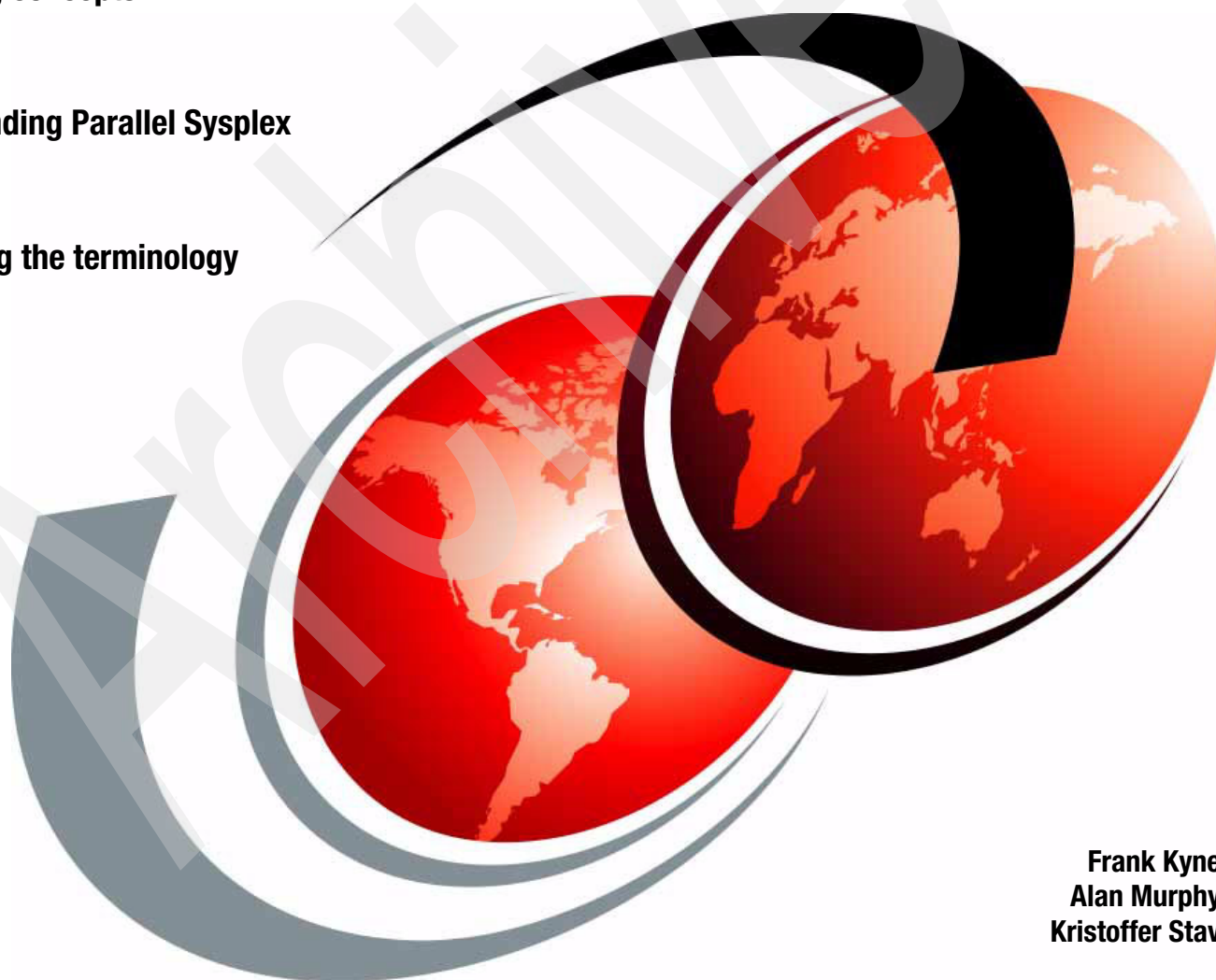


Clustering Solutions Overview: Parallel Sysplex and Other Platforms

Clustering concepts

Understanding Parallel Sysplex
clustering

Comparing the terminology



Frank Kyne
Alan Murphy
Kristoffer Stav



International Technical Support Organization

Parallel Sysplex Clustering Comparison

April 2007

Archived

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

First Edition (April 2007)

This edition applies to Version 1, Release 7, Modification 0 of z/OS (product number 5694-A01).

This document created or updated on April 9, 2007.

© Copyright International Business Machines Corporation 2007. All rights reserved.

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

Contents

Notices	v
Trademarks	vi
Preface	vii
The team that wrote this Redpaper	vii
Become a published author	viii
Comments welcome	viii
Chapter 1. Introduction	1
1.1 What is clustering	2
1.1.1 Why cluster?	2
1.1.2 Cluster types	2
Chapter 2. Clustering basics	5
2.1 Commonly used and abused terminology	6
2.1.1 Grid	6
2.1.2 Partition	6
2.1.3 Interconnect	6
2.1.4 Node / cluster member	6
2.1.5 Redundancy	6
2.1.6 Single point-of-failure	6
2.1.7 Failover	7
2.1.8 Cache	7
2.1.9 Lock	7
2.1.10 Failure	7
2.1.11 Split-brain	7
2.1.12 Active-active	7
2.1.13 Active-standby	7
2.2 Cluster architectures and storage models	8
2.2.1 Shared-nothing model	8
2.2.2 Shared-everything model	10
2.3 Cluster performance	13
2.3.1 Scalability	13
2.3.2 Interconnects	13
2.3.3 Shared configuration	14
2.3.4 Access control	14
2.4 Cluster availability	14
2.4.1 System upgrades	14
2.4.2 Failure response	14
2.4.3 Restart and recovery	15
2.5 Cluster features checklist	16
Chapter 3. z/OS Parallel Sysplex	17
3.1 Brief history of IBM System z clustering with z/OS	18
3.2 Terminology	18
3.2.1 Early clustering support on System z with z/OS	18
3.2.2 Base sysplex (1990)	19
3.2.3 Parallel Sysplex (1994)	19
3.3 Parallel Sysplex architecture	20

3.3.1 Parallel sysplex components.	21
3.4 DB2 for z/OS data sharing implementation.	25
3.4.1 DB2 for z/OS terminology	26
3.4.2 Storage model.	26
3.4.3 Performance	38
3.4.4 Availability - restart and recovery	40
3.4.5 Systems management	43
3.4.6 Application requirements	46
Chapter 4. Clustering on AIX	51
4.1 History of clustering on AIX5L.	52
4.1.1 RS/6000	52
4.1.2 pSeries	53
4.2 System p5 software.	54
4.2.1 General Parallel File System.	54
4.2.2 Cluster Service Manager.	54
4.2.3 High-Availability Cluster Multiprocessing	55
4.3 DB2 UDB clustering implementation on AIX	56
4.3.1 DB2 active-standby cluster using HADR.	57
4.4 Oracle Database 10g RAC implementation on AIX	61
4.4.1 Storage model.	61
4.4.2 Application requirements	62
4.4.3 Performance	63
4.4.4 Availability - restart and recovery	63
4.4.5 Systems management	63
Related publications	65
IBM Redbooks	65
Other publications	65
How to get IBM Redbooks	65
Help from IBM	65
Index	67

Notices

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

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

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

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

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

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

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

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

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


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

COPYRIGHT LICENSE:

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

Trademarks

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

AIX 5L™	MVS™	Sysplex Timer®
AIX®	MVS/ESA™	System p5™
CICS®	MVS/XA™	System z™
DB2 Connect™	OS/390®	System/360™
DB2 Universal Database™	Parallel Sysplex®	VTAM®
DB2®	POWER™	WebSphere®
Enterprise Storage Server®	POWER Hypervisor™	xSeries®
eServer™	POWER5™	z/Architecture®
General Parallel File System™	pSeries®	z/OS®
GDPS®	Redbooks™	z/VM®
GPFS™	Redbooks (logo)  ™	zSeries®
HACMP™	RS/6000®	z9™
IBM®	S/360™	1350™
IMS™	S/370™	
iSeries™	S/390®	

The following terms are trademarks of other companies:

SAP R/3, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

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

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

This IBM® Redpaper will help you do an informed comparison of Parallel Sysplex® against clustering implementations on other platforms. We start by describing the basic components of Parallel Sysplex data sharing, using DB2® as a sample implementation of the capabilities provided by Parallel Sysplex. Using this as a base, we then describe how each of the components are implemented on other platforms.

This Redpaper does not attempt to discuss all of the very many flavors of clustering or even a majority of them. We are concentrating on clustering solutions suitable for commercial online transaction processing that require good scalability and high or continuous availability. All of our implementation examples are of clustered database solutions because these present the greatest challenge for a cluster implementation.

We begin with a chapter that discusses the concepts around clustering such as types of clusters, storage models, and associated data integrity issues. This chapter forms the basis for the concepts and terminology used in the following chapters that discuss z/OS® parallel Sysplex and clustering on AIX®.

The team that wrote this Redpaper

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

Frank Kyne is a Certified IT Specialist at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of Parallel Sysplex. Frank is also responsible for the GDPS® product documentation. Before joining the ITSO seven years ago, Frank worked in IBM Global Services in Ireland as an MVS™ Systems Programmer.

Alan Murphy is a Senior IT Specialist working for IBM Global Services in Ireland. He has 18 years of systems programming experience on the zSeries® and S/390® platforms. He is co-author of a number of previous IBM Redbooks™ on Parallel Sysplex.

Kristoffer Stav is an IT Specialist in IBM Norway. He works with z/VM® and Linux® solutions, and WebSphere® products on IBM System z™. Before joining IBM, Kristoffer did his master thesis, "Clustering of servers, a comparison of different platform implementations", at the University of Oslo in cooperation with IBM and the Norwegian Central Securities Depository.

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

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Scott Fagen
IBM USA

Madeline Nick
IBM USA

Marcy Zimmer
IBM USA

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an e-mail to:

redbook@us.ibm.com

- Mail your comments to:

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

Introduction

It is unlikely that there are any businesses today that do not use a variety of computing platforms. One of the complexities of operating in such an environment is trying to identify the most appropriate platform on which to deploy new applications.

One of the difficulties faced by z/OS Systems Programmers, when discussing platform capabilities with their colleagues on other platforms, is that the PC and the UNIX® platforms typically use different terms for the same things (files versus data sets, disks versus DASD, and so on). Nowhere is this more evident than when it comes to discussing clustered computing solutions.

The term *Clustering* means dramatically different things to different people and spans everything from massively parallel computing environments used by research scientists to simple flivver solutions between a couple of PC file servers running Windows® or Linux.

This Redpaper addresses the following two audiences:

- ▶ People with predominantly System z experience who want to gain an understanding of some clustering approaches used in commercial computing in Unix
- ▶ People with predominantly Unix experience who want to gain an understanding of System z Parallel Sysplex concepts and understand how that relates to the Unix world.

1.1 What is clustering

A cluster is a collection of interconnected computers called nodes or cluster members, working together as a single system. A single system means that the set of nodes connected together in the cluster appear to the users as a one entity.

1.1.1 Why cluster?

First, a cluster may improve the data and application availability significantly compared to a single system. Two or more computers makes the system more available and may eliminate the single points of failures found in single-system implementations. A second reason for using a cluster is that when the workloads exceed the capacity of a single system, adding more resources fixes the problem. Putting more computers into work to handle unexpected processing demands is what clustering is all about. And this can be done dynamically in a nondisruptive manner to maintain the availability.

1.1.2 Cluster types

Clusters can be divided into the following two groups:

- ▶ High-performance (HP) clusters
- ▶ High-availability (HA) clusters

Both are briefly described in the following sections.

High-performance clusters

High-performance clusters are used in numerically intensive environments where the workloads tend to be very processor intensive. These clusters are typically used in scientific areas like weather forecasting, earthquake analysis, and in render and compilation farms, to mention a few.

High-availability clusters

High-availability clusters are typically used to host commercial applications, databases, and for handling big amounts of transactions simultaneously. Applications and database systems should in many cases be available for the users and customers, even if a node is taken out for maintenance or update or fails for some reason. With global e-commerce it is vital to have 24x7 application availability.

Systems that have very high demands for availability may be spread over multiple sites. This is called geographically dispersed clusters and basically implies that the nodes are physically scattered over several, usually two or three, sites. If an entire site is disabled, for some reason, the applications and data are still available. The nodes are connected to each other through high-speed fiber interconnects.

A *two-node cluster* is widely used to provide a redundant high-availability system.

- ▶ One approach is to let one node handle all the workloads, while the other node is acting as a spare. This configuration is called an *active/passive cluster*. If something unexpected happens to the active one, the passive node is ready to kick in. This can be done in a matter of minutes.
- ▶ On the other hand, a scenario where both nodes participate actively and share the workloads is called *active/active clustering*. This configuration is a bit more complex, and requires a careful look at how data is shared and kept consistent across the cluster members.

A myriad of different cluster solutions exists, designed to solve all kinds of different problems. As an example, the number of scientific high-performance cluster projects under Linux is really impressive. This document focuses on high-availability clusters used in commercial environments and addresses topics related to this, especially how data is shared among the nodes.

Archived

Archived

Clustering basics

A cluster is a collection of interconnected computers called nodes or cluster members, working together as a single system. A single system means that the set of nodes connected together in the cluster appear to the users as one entity. The purpose of a cluster is to improve availability over a single system; thus, users can continue to access database systems even if a node is taken offline, whether it is planned or not. Another great advantage with clustering databases is the ability to scale on demand. Adding nodes when needed can be done in a nondisruptive manner if the cluster is properly designed.

2.1 Commonly used and abused terminology

Following are the most common terms that could be used with different meanings depending on the context they are used.

2.1.1 Grid

The term *Grid* has different meanings depending on the context in which it is used.

- ▶ The Oracle® database solution Real Application Clusters (RAC) is deployed in what Oracle calls grids. In fact, the Oracle RAC grid solution is realized at the application level. Grid in this context is a collection of small coordinated computers built of standard hardware components acting as a single-large computer.
- ▶ In the scientific world, a grid is at the infrastructure level and represents a collection of homogenous, loosely coupled computers working together to utilize their processor power to solve some computational intensive problems in common, which can be regarded as a subset of high-performance clusters. In general, the nodes in a cluster work together to form a single-system image, so they are more tightly coupled than the nodes in a grid.

2.1.2 Partition

The term *Partition* has different meanings in different contexts.

- ▶ In an IBM DB2 context, partition is a logical cluster member and the data associated with it.
- ▶ In a Parallel Sysplex context, a partition is a running instance of an operating system. A more generic meaning of the term is a disk partition.

2.1.3 Interconnect

Interconnects between computer systems enable the nodes to communicate and exchange data with other nodes, clients, and storage systems over a network or point-to-point connections, without writing it to disk.

2.1.4 Node / cluster member

A *node/cluster member* is a computer system that is part of a group of systems that make up the cluster. All nodes or cluster members have some degree of interconnection and run some common services that manage cluster membership.

2.1.5 Redundancy

A failure to a single component, whether it is hardware or software, should not affect the availability to the entire cluster. *Redundancy* is achieved with having spare components available.

2.1.6 Single point-of-failure

A *single point-of-failure* is a software or hardware component that represent a loss of availability if failure occurs.

2.1.7 Failover

Failover is the process of moving services and workloads from one cluster member to another if a failure occurs.

2.1.8 Cache

A *cache* is a buffer stored in main storage (or RAM) that holds data blocks that is often or recently accessed or not yet written to disk. The intention with caches is to boost up performance since disk access is much slower than memory access.

2.1.9 Lock

Locks are entities that ensure that only one process accesses shared resources, such as files or data in a database, at a time.

2.1.10 Failure

Failure is an unexpected behavior of a hardware or software component or an operator that influences the availability of a node.

2.1.11 Split-brain

A *split-brain* occurs when a node or a set of nodes loses connect to the rest of the cluster. When they become unable to communicate with each other they are not aware of the existence of the other nodes. They both assume the other one is dead. This can lead to data integrity problems when both parties access shared resources concurrently in an uncontrolled fashion. Clusters usually have mechanisms to avoid a split-brain scenario. Read more about avoiding split-brain in section 2.4.2, “Failure response” on page 14.

2.1.12 Active-active

Active-active is a term used to denote that multiple members of the cluster run a workload concurrently. In our discussion on clustered database systems, it means that each member of the cluster actively manages some or all of the data.

In the authors’ experience, this term often causes confusion. For example, a cluster in which multiple members run different unrelated workloads with no element of data sharing may be described as active-active because all members are actively running workloads with the ability to failover workloads between cluster members. However, this model does not conform to our definition of active-active because data is not being actively shared between the members.

2.1.13 Active-standby

Active-standby is a term that denotes where one member of a cluster runs the active workload, while another member is available to take over that workload in the event of some failure on the active member.

2.2 Cluster architectures and storage models

As mentioned in section 1.1, “What is clustering” on page 2, there are many types of clusters; however, we are looking at examples of shared-database clusters in particular. This implies that data must be shared across members of a cluster in some way.

Attention: A simple active-standby cluster model does not need to actively share data across the cluster members. The standby node just needs to be able to access the disk once failover occurs. We are not discussing this model here.

In a typical online transaction processing environment, a clustered database solution must be able to cope with a vast number of concurrent Inputs/Outputs (I/Os) in a controlled and efficient way.

In this section we discuss various approaches for accessing and sharing the data. In many cases all the nodes have to access the same data, while in other cases it is appropriate to split up the data among the nodes. We discuss several approaches to make data available to the cluster members.

2.2.1 Shared-nothing model

Each node has its own disk and owns a subset of the data. A node typically has no direct access to disks owned by other nodes, so it may only read or modify its portion of the data. Figure 2-1 is a logical view of this configuration. In a shared-nothing database cluster, the databases have to be partitioned or replicated.

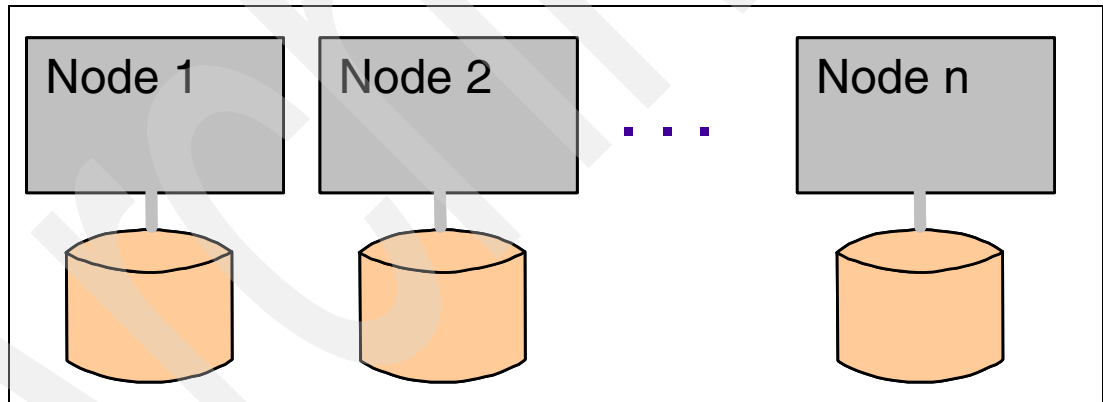


Figure 2-1 Shared-nothing model

Partitioned model

In a partitioned model, each member manages a portion of the data. Typically, if a member receives a request for data that is managed by another member of the cluster then the request is shipped to that member and the results retrieved from it. This is called function shipping. Figure 2-2 on page 9 shows a logical layout of a partitioned database.

In this environment, each member manages its own cache and local locks, so there are no cache coherency issues and a global lock manager is not required. In a transactional environment where updates happen across multiple members of the cluster, then each member must participate in a two-phase commit process to ensure transactional integrity.

Sharing data is usually performed through function shipping that minimizes the data movement. If function shipping is not enabled, your installation needs to distribute the workload to the correct nodes that have access to the data needed.

Adding nodes to a shared nothing partitioned model requires repartitioning of data. This implies that the entire cluster may have to be shut down during this operation. This is not acceptable in clusters that require continuous availability. Repartitioning is often time consuming and requires a lot of planning. Partitioned databases are built with the intent of being able to balance the workload across all nodes.

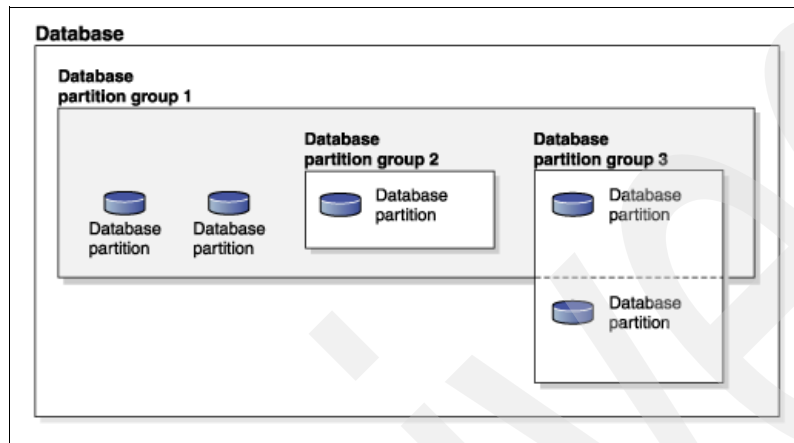


Figure 2-2 Partitioned database layout

Replicated model

Data may be replicated across the nodes. Each node manages a complete copy of the data, and client requests may be directed to any node in the cluster. This is appropriate if the majority of the database queries are read operations.

Having duplicates of data, which is frequently altered, produces a lot of inter-nodal communication in order to keep all the instances of the data up-to-date. Managing concurrent updates across multiple members with replicated data becomes a big overhead, and this model is not commonly used in a transactional environment.

Federated-database model

The federated-database model is often considered as a cluster, which it is not. The federated-database is a subset of the shared-nothing model. A DB2 federated system is a special type of distributed database management system. A federated system consists of a DB2 instance that operates as a federated server, a database that acts as the federated database, one or more data sources, and clients accessing the database and data sources. With a federated system, you can send distributed requests to multiple data sources within a single SQL statement. For example, you can join data that is located in a DB2 Universal Database™ table, an Oracle table, and to an XML tagged file in a single SQL statement. Figure 2-3 on page 10 shows the components of a federated system and a sample of the data sources you can access. In this model, the servers are considered as loosely coupled, interconnected by Local Area Networks (LANs), with no resources shared among them. The data is distributed between the servers. All parties are equally important since no data is replicated. The availability characteristics for a federated database are poor and there is no control mechanism for data integrity.

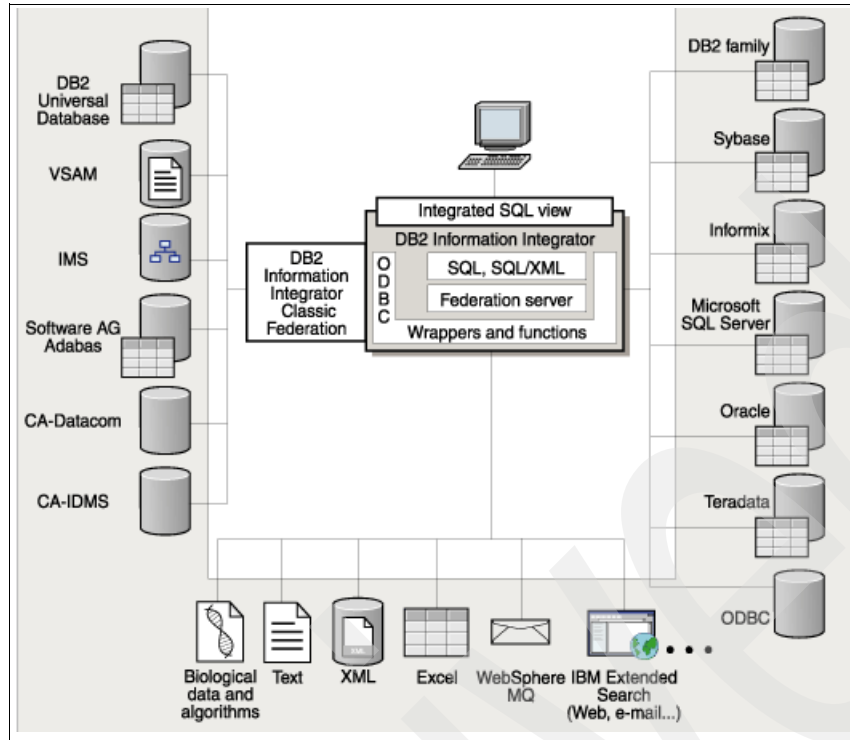


Figure 2-3 Federated database layout

2.2.2 Shared-everything model

The shared-everything model implies that data access is shared across all the members of the cluster. All cluster members share the same set of storage. They all have direct access to the shared disks. In the shared-everything model, mechanisms to synchronize data access are required to maintain data integrity. Access granularity may be at file level (the most basic) or go down to database-row level.

The shared-everything model may be divided into the following two groups: the shared-disk model and the shared-cache mode. The shared disk model shares data at the file level and relies on shared disk communication, while the shared cache model can have a more fine grain access policy, as well as at file level, including tables, pages, and rows. Managing data sharing introduces much more complexity compared to controlling access to shared files.

An important goal in a busy database cluster is to achieve the best throughput and response time as possible. Keeping the inter-node communication and disk activity as low as possible are factors that affect the performance. There is always some traffic between the nodes. They pass lock and cache information around, and in the shared nothing model that was presented in the previous subsection, they produce a lot of communication overhead due to function shipping. Ideally scaling should be linear with the number of nodes in the cluster, but it can be impacted and limited by the communication overhead. Technology can help to alleviate this limitation as you can see in the latest SMP servers. In the shared-nothing model, access to the data is constrained through a single-database instance, while in the shared-everything model data can be accessed through multiple database instances that increase both the performance and availability characteristics of the cluster. Figure 2-4 on page 11 shows the logical view for a shared-everything model.

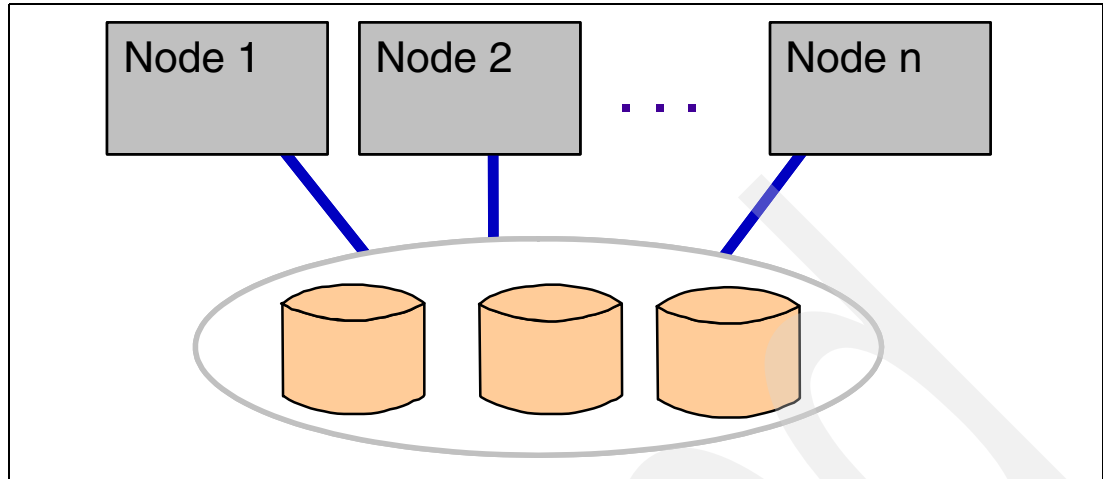


Figure 2-4 Shared-everything model

Shared-disk model

In the shared disk model, the data is logically shared among all the nodes. Each node can access the same data directly. The shared disk model is appropriate for fault tolerant systems. If a node fails for some reason, the other nodes can still access the data. Since the data may be accessed by all the nodes, this model is perfect for workload balancing as well. Workloads may be distributed to any of the nodes, based on resource utilization.

In the shared-disk model, a lock coherency control has to be enforced in order to serialize the access to shared resources. Only one node can update a piece of data at a time, which we address in the following subsection.

Lock management

In a cluster, multiple nodes read and write to shared storage. These accesses have to be coordinated in order to guarantee data integrity. The locking schemes vary from cluster to cluster, but they all provide mechanisms to serialize access to resources that are simultaneously shared by multiple nodes. Global concurrency control with global locks has to be enforced. When a node needs to access a shared resource, it tries to obtain a lock. If the resource is not accessed by an other process, that is the resource is not locked, access is granted and a lock is set. The lock is released when the operation is completed. Processes trying to access a locked resource are put in a queue until the lock is released.

There are several solutions to which resources are allocated and locked. The most basic and straight forward solution is to take ownership of the entire disk or partition. The locking is done on the operating system level. This approach is certainly not a very good solution for data intensive applications. Lock files control access to files. This is usually done by storing a file in the same directory as the locked file, which acts as a lock. The access control occurs on the application level, so this scheme is not as robust as allocating the entire partition or disk system. Locks can either be shared or exclusive. If the operation involves writing, the lock is always exclusive. Shared locks are granted for multiple nodes that make read operations concurrently.

The locks can either be maintained in a distributed or in a centralized fashion. The distributed lock scheme was the first locking scheme and is still the most widely used. In the distributed architecture each node manages a piece of the global lock database, since the lock database, as the name indicates, is distributed among all participants. They communicate directly to each other. Each resource is usually associated with a master node. The scaling characteristics are poor when the number of cluster members increases, compared to the

centralized model. The overhead when obtaining and releasing locks is high. With a centralized lock assist, the nodes communicate with each other via a dedicated lock server. The centralized approach has less overhead and it scales better. Messages are sent between single nodes and the lock manager, instead of having a mesh of communication paths within the cluster. In general, the less synchronization between the nodes, the better the cluster performance and scalability will be.

The lock manager can either be eager or lazy. An eager lock manager releases the lock when the transaction is committed, whereas a lazy lock manager releases the lock on conflict. Distributed locking managers usually use the latter. Lazy lock managers produce a lot of overhead and delay. As long as the traffic is low, this is a satisfying approach.

Shared-cache model

The shared-cache model can be regarded as a subset of the shared-disk model, since having a shared-cache model without a shared storage system is unusual. All nodes may access each others' cache. No I/O is required to obtain a needed piece of data. This approach has a significant performance gain over the shared-disk model, since accessing data from main storage (or RAM) is considerably faster than doing disk operations. High-speed interconnects are required for efficient communication between the nodes. In order to retrieve data stored in a remote cache, messages are passed between the nodes. Fetching data blocks from a cache on a remote node may take a few milliseconds, while accessing a data block from disk is typically many times slower. The data block access time for a local cache buffer is measured in microseconds. The shared cache can either be distributed or be centralized.

Cache topologies

Several cache topologies exist. We briefly discuss three of them in this section. The first three are implemented in distributed cache architectures, while the latter is centralized.

► Replicated cache

The cache on all nodes are identical. All data is replicated, so from an availability point-of-view, replicated cache is perfect. Seen from a performance point-of-view, it depends of the type of transactions. This way of organizing the caches is appropriate for reads; however, if the majority of transactions involves updates, the overhead is bad. The replicated cache approach has major problems when it comes to scaling. This produces a lot of network traffic, especially when the cluster has a high number of nodes and there are a lot of transactions.

► Partitioned cache

Partitioned cache architectures scale better than the replicated cache. Each data block is "owned" by one node. Each and every node is aware of which node the data blocks in the caches belongs to. Read and write operations involve two nodes at the most, so partitioned cache scales much better than the replicated cache.

► Near cache

Each data block has a master location, just like in the partitioned cache, but this cache model allows duplicates. A write operation updates the master as well as the local if there is any. A read fetches the data block from a remote node if it does not reside on the local node. The duplicates are always synchronized.

► Centralized cache

A centralized cache concurrency coherency control keeps track of which nodes have which data blocks stored in their local caches. Invalidation messages of data blocks are sent to nodes that have a local copy, in case of a write operation.

Lock management

The locking scheme in the shared cache model can be either centralized or distributed, exactly the same as in the shared disk model. In addition, to have access control at partition and file level, the shared cache model allows a more fine grained lock management solution. Data units that are able to be locked can be a table, a row, or a data block. This allows multiple instances of the database application to access the database concurrently.

Cache

A global cache coherency control for the local caches has to be provided. There has to be a way to keep track over the local caches to make sure they are valid. Caches introduce synchronization problems because they represent duplication of data. The cache control can, as the lock manager, either be centralized or distributed. A cache coherency control is needed for write operations, not for read operations. Every time a node makes an update of a data block, remote hosts that have a local copy of that particular data block must invalidate their copy. A message is sent to all, or some of the nodes, saying the following: "Invalidate your copy of this data block". The nodes this message is sent to depends on how the cache control is managed. If the node has no overview of which other nodes have a remote copy of the data block, a broadcast invalidation message must be sent. The overhead is high, this approach produces a lot of network traffic, particularly when the number of nodes in the cluster is large.

2.3 Cluster performance

The following sections provide some performance considerations

2.3.1 Scalability

To ensure data consistency, a way of serializing the data access between the nodes has to be enforced. This synchronization mechanism is provided by a lock manager. There are multiple ways to construct a shared-disk cluster with emphasis on how locks are passed between the parties involved.

The global concurrency control could either be distributed or centralized. We address these two different approaches: one way to construct a cluster is to use a star topology, where the cluster has a centralized lock manager. A second way is the ring, where tokens/locks are passed around. Star topology has much better scalability characteristics than the ring. In a ring topology, adding more nodes increases the internode communication.

The star topology has a centralized lock manager. All the nodes are connected directly to the star. Another approach is to connect the nodes together in a circular fashion or in a mesh. This is called a ring topology.

2.3.2 Interconnects

In a shared-everything cluster it is crucial that the delay is at the minimum and the throughput is at the maximum. Copper-based interconnects have a limitation of both distance and performance. Traditional Fast-Ethernet LAN interconnects are often used in low budget clusters made of commodity hardware components. These network components may be a bottleneck when the demands for throughput and delay are high. Interconnects based on fiber optics is suitable for clusters with demand for high performance. Fiber is also widely used between different geographical sites.

2.3.3 Shared configuration

It is highly advantageous to have a single point-of-administration. When committing changes to the cluster settings, the changes effect the whole cluster from one, single point-of-control. To have a shared configuration, the configuration files have to be stored in a place where they are accessible by all the cluster members. This may be provided in a shared all cluster, but shared configuration is hard to manage in a shared-nothing cluster.

2.3.4 Access control

As with a single point-of-administration, it is highly advantageous to have shared access control across all members of the cluster. Access control information may be held in a shared database of changes, which can be propagated across all members of the cluster.

2.4 Cluster availability

The following sections provide some availability considerations.

2.4.1 System upgrades

Some clusters have the ability to have different versions of software components in the cluster at the same time. This allows the operators to perform rolling upgrades. The cluster can be upgraded in a nondisruptive manner, seen from the users point-of-view. The nodes can be taken out one at the time and brought back online after maintenance.

2.4.2 Failure response

Failure handling can be done on several levels. If a hardware component or the operating system fails, the entire node might be out of action. Then other cluster members have to clean up the mess and take over all its workloads. If a particular service encounters an error and fails, this can be solved by the operating system locally. Failure handling should be automated, so no manual intervention is needed. Practically, the failure and recovery routines should be transparent to the clients.

A lot of factors can cause a node failure, whether it is a hardware component, the operating system, connectivity problems, database problems, or failure in an application. The overall goal is to maintain data integrity, and the core issues are how locks and caches are handled. Node failure is usually detected by lack of heartbeat signals. Any surviving node in the cluster can initiate the recovery process. The first thing to do is to make sure the failing node does not mess up the shared data. In response, a node could either shut down the entire failing node or fence it off.

Kill the failing node

Killing the failing node is a straightforward approach but not very sophisticated. Each node is connected to a network controlled power switch. The power supply is cut off by another node when it is no longer considered a reliable member of the cluster. The goal is to prevent the node from accessing the shared storage. It certainly prevents the failing node from accessing shared storage, but there are a number of side effects. If multiple services are running on the failing node, this method also halts these services, which in most cases are unacceptable. And ever worse, where two nodes are not able to communicate with each other, they might both go down. The recovery process may be complex, since you never know in which state the recently re-booted node is in, which may have an unpredictable impact on the cluster.

Attention: The killing the failing node method is not the recommended way of operating.

Fence off the failing node

A second solution to a failure response is to form an I/O barrier. Here the failing node is isolated from the shared I/O devices. This is a more graceful solution because no node is killed here; instead, the node is just excluded from the cluster and fenced off from the shared storage system. This can be done by simply disabling the ports to the I/O devices. This implies that the node itself can run diagnosis and recovery routines. Another benefit is that applications that do not require access to the shared storage can still run on the fenced-out node. A third advantage is that no extra hardware is required.

Self-healing node

If a node loses connectivity to the other nodes or to the shared storage system, it suspends or stops itself. When the failing node stops responding to heartbeat signals, the other nodes conclude that the node is dead. The other nodes then start the recovery process of cache and locks.

2.4.3 Restart and recovery

Failure recovery can be an extremely complex thing to do, particularly if the cluster implements the shared-cache model. Tasks required after a failure may include system diagnosis, release locks held by failed node, rollback and recommit transactions, rebuild cache, rebuild lock database, reconnect clients, restart failed node, and bring it back online. Two overall goals with failure recovery are that it is done as fast as possible and that data integrity is held so that the impact for the clients is as small as possible. We will in this subsection mainly discuss how lock and cache information is recovered.

The cache and lock data that are mastered by the failing node have to be recovered and rebuilt. Cache and lock information are restored from log entries committed to disk and in a centralized shared cache implementation, from the shared cache as well. Evidently, in order to recover the data, a still remaining valid copy must exist whether it is on the other nodes or on persistent storage. In some database clusters, the database is not available to clients during the recovery process. In other database systems, only the data that is rebuilt is not accessible. Each node writes to its own recovery log, however, the logs must reside on a shared disk, so all nodes have access for recovery processing.

A better solution is to withhold the incoming transactions while recovery is in progress and release them as soon as the system is restored.

Sequencing log entries

During recovery, logs are used to reconstruct lost data. When rolling back uncommitted transactions, you need to know in which sequence the log records were written to figure out in which order the operations happened. A common way of synchronizing the sequence for log entries is to use timestamps. The nodes need to have the same perception of time. The time synchronization ought to be extremely precise, especially in busy database clusters that may have to handle hundreds or thousands of operations per second. Another solution is to use sequence numbers. With this approach the nodes have to synchronize the sequence number, either in a centralized or distributed fashion.

2.5 Cluster features checklist

Table 2-1 contains a brief list of features that are normally provided by database solutions that implement each cluster model.

Table 2-1 Cluster features checklist

	Shared nothing	Shared disk	Shared cache
Cluster interconnect	YES	YES	YES
Ability to takeover disk storage from failed member (requires sharable disk)	YES	N/A	N/A
Failure detection	YES	YES	YES
Automate move of resources (network, storage) from active node to standby node	YES	N/A	N/A
Shared storage	N/A	YES	YES
Global lock manager to serialize access to shared disk	N/A	YES	YES
Ability to partition out failing nodes	N/A	YES	YES
Node restart / recovery support	N/A	YES	YES
Ability to sequence log updates across the cluster	N/A	YES	YES
Global cache/buffer manager to maintain cache coherency	N/A	N/A	YES

z/OS Parallel Sysplex

This chapter describes the basic concepts of Parallel Sysplex, specifically Cross System Coupling Facility (XCF) messaging and the data sharing using DB2 as a sample implementation.

In this chapter we discuss the following:

- ▶ Brief history of System z clustering
- ▶ Terminology of Parallel Sysplex
- ▶ Parallel Sysplex concepts
- ▶ DB2 on z/OS datasharing implementation

3.1 Brief history of IBM System z clustering with z/OS

The IBM System z has evolved from the IBM System/360™ platform introduced in 1964 all the way to the IBM System z platform available today. Throughout this period the platform has moved from 24 bit through 31 bit to the current 64 bit implementation. Binary compatibility was maintained so that programs written for the original System/360 could still run on the current platform.

Table 3-1 gives a brief time line of the major architectural changes over the past 40 years.

Table 3-1 Architectural changes

Decade	Architecture	OS	Addressing
60s	S/360™	MVT	24bit
70s	S/370™	MVS/370	24bit
80s	S/370-XA	MVS/XA™	31bit
90s	S/390-ESA	MVS/ESA™ OS/390®	31bit
00s	z/Architecture®	z/OS	64bit

From the very start of the platform, the emphasis was on providing a robust, secure, and highly-available commercial computing platform. Today, System z is used in commercial computing environments that run high-volume online and batch systems. It is used extensively in the financial services, airline, and retail industries.

3.2 Terminology

The following section takes you through the different stages of the clustering evolution on the System z platform, and introduces the different concepts and architectural changes.

3.2.1 Early clustering support on System z with z/OS

From a transactional workload point-of-view, the early clustering functionality was not capable of sharing data and relied on the idea of function shipping. This function was implemented not in the operating system but by individual transaction/database managers. Take the following for example:

- ▶ CICS® function shipping
- ▶ IMS™ Multiple Systems Coupling (MSC)

The operating system permitted shared disks but used the hardware Reserve/Release mechanism to synchronize the update to the data—with the Reserve/Release only one system could access a disk at any one time. Unlike SCSI reserve on open systems, a disk was only reserved for the period while the update was happening. It was possible to share files between systems but the update of an individual file required the whole disk to be locked for that period. This meant that shared disk access had to be carefully planned to prevent excessive queuing and deadly embrace problems.

In the early 80s, z/OS implemented a global lock manager called Global Resource Serialization (GRS) that allowed serialization at the file level, removing the need of the hardware Reserve in most cases. This was implemented by a GRS task on each system communicating with every other system via high-speed interconnects. GRS supports

serialization on any resource, not just files, and is used by many applications to provide serialization across multiple z/OS operating system images.

3.2.2 Base sysplex (1990)

In September 1990, IBM introduced the SYStems comPLEX, or sysplex, to help solve the complexity of managing multiple z/OS systems. A sysplex is a collection of z/OS systems that cooperate using hardware and software functions to process work, providing improved growth potential and higher level of availability. The sysplex architecture established the groundwork for simplified multisystem management through the Cross-System Coupling Facility (XCF) component of z/OS. XCF services allow authorized applications on one system to communicate with applications on the same system or on other systems. In a base sysplex, connectivity and communication between images is provided by high-speed interconnects called channel-to-channel (CTC) links. The couple data set, which is shared between all of the images, holds control information and provides a mechanism for monitoring the status of the images. When more than one CPC is involved, a Sysplex Timer® is required to synchronize the time on all systems.

From a functionality perspective, the base sysplex did not change the clustering storage model. z/OS still provided a shared-disk model without shared cache. What sysplex added was the following:

- ▶ A standard way to communicate between systems
- ▶ The support for quorum disks and member status
- ▶ A common time source in the cluster

3.2.3 Parallel Sysplex (1994)

The basic sysplex laid the foundations for communications between components and subsystems on the participating z/OS images, but the basic sysplex did not provide the support for cache buffer coherency required for high-performance database subsystems. In addition, the existing GRS ring implementation could not provide the required performance for sharing high-volume transactional systems across a large number of systems.

With the Parallel Sysplex technology, high-performance data sharing through a new coupling technology (Coupling Facility) gives high-performance multisystem data sharing capability to authorized applications, such as system components and subsystems. The issue of the coupling facility by subsystems, such as IMS/DB DB2 and VSAM/RLS, ensures the integrity and consistency of data throughout the entire sysplex. The CF is implemented in a Logical Partition (LP) just like a z/OS image but it runs its own highly specialized operating system, Coupling Facility Control Code (CFCC). It is connected to all the z/OS images in a Parallel Sysplex by high-speed fibre or copper links. The design of the CF ensures that requests are queued in a logical order that is not dependent on the speed of the connected CPCs. This removes the unpredictable response time that is inherent in conventional devices such as DASD and CTC.

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

The Parallel Sysplex technology builds on the base sysplex capability and allows to increase the number of CPCs and operating system images that can directly share work. The coupling facility enables high performance, multisystem data sharing across all the systems. In addition, workloads can be dynamically balanced across systems with the help of new workload management functions.

Parallel Sysplex enhanced the clustering storage model provided by System z to a shared cache model. The following is what parallel sysplex added:

- ▶ The high-performance shared memory external to the operating system images in the cluster.
- ▶ The enhanced z/OS components exploiting the coupling facility. For example, GRS could now move to a star topology using the CF to store global lock information, avoiding a continuous communication activity to keep all the systems synchronized. This greatly improved GRS performance when many systems were active in the sysplex.
- ▶ The z/OS support for subsystems (DB2, VSAM, and IMS/DB) to exploit the shared cache and lock facilities in a standardized manner.

3.3 Parallel Sysplex architecture

Parallel sysplex provides the infrastructure for shared cache clustering as detailed in section 2.2.2, “Shared-everything model” on page 10. Parallel sysplex is provided by a combination of hardware components and software components. These components provide a very advanced infrastructure that any database or application server providers (IBM and ISV) can use in a standardized manner to enable their application to be shared across multiple systems with full data integrity and very high performance.

Figure 3-1 illustrates the Parallel Sysplex components in a two-way configuration sample. It is a logical representation of the Parallel Sysplex. There are many variations on how the logical components can be mapped to physical hardware, in fact z/OS images and CFs can run in LPARs as well as on physical CPCs.

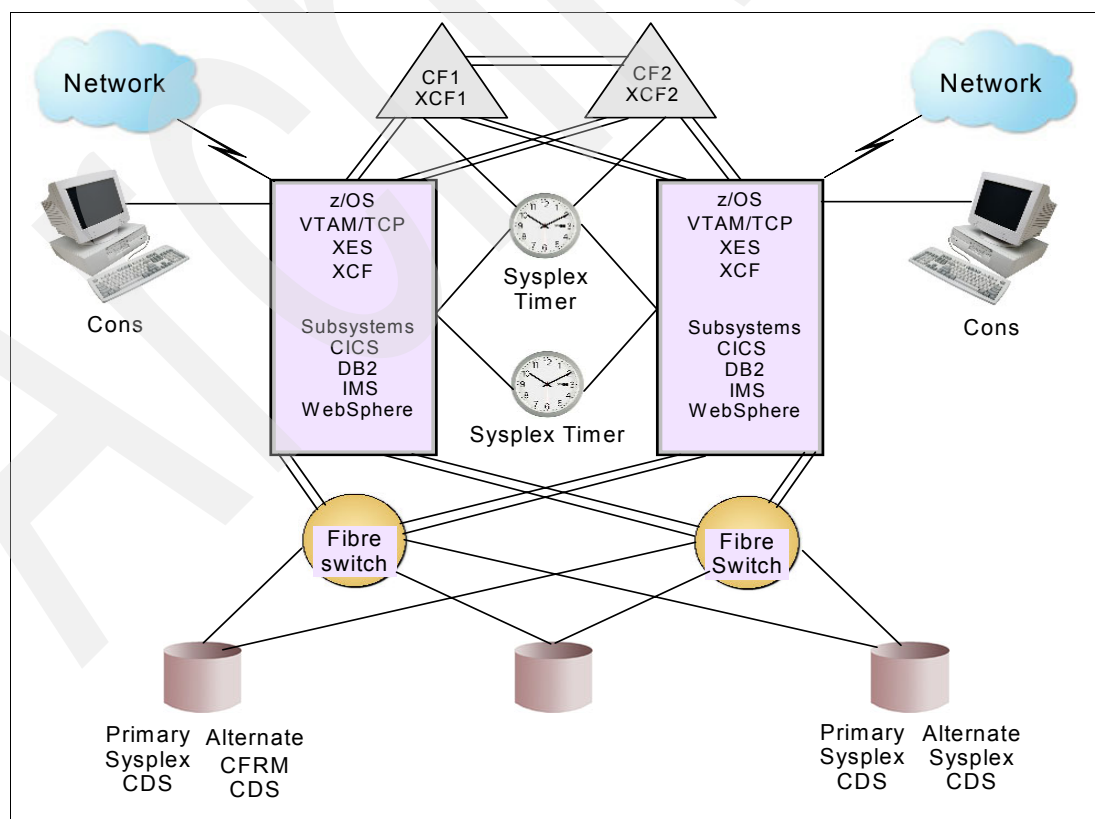


Figure 3-1 Parallel Sysplex components

The number of Coupling facilities is related to the volume and nature of the shared workload in the Parallel Sysplex. Two is the minimum recommended for availability reasons, and in the authors' experience only the largest System z customers need to go above two for capacity reasons.

3.3.1 Parallel sysplex components

This section discusses the components that make up a Parallel Sysplex.

Coupling Facility

A coupling facility (CF) is a logical partition on the System z processors. The coupling facility allows software on different systems in the sysplex to share data. The coupling facility is the hardware element that provides high-speed caching, list processing, and locking functions in a sysplex. The Coupling Facility has processor resources assigned (dedicated or shared) and a RAM assigned to run its own specialized OS called Coupling Facility Control Code (CFCC) loaded from the System z hardware. It has no disk storage attached to it. The Coupling Facility must have connectivity to all the systems sharing the data through coupling facility channels.

The CFCC interacts with the XES component on z/OS to create and maintain data in the CF memory data. In the CF, data is organized into distinct objects called structures. CF structures are used by authorized programs to implement data sharing and high-speed serialization. There are a number of different types of CF structures that can be created: cache, list, and lock, each providing a specific function to the application. Some storage in the coupling facility can also be allocated as a dedicated dump space for capturing structured information for diagnostic purposes.

Cache Structures

A cache structure supplies a mechanism called buffer invalidation to ensure consistency of cached data. The cache structure can be used as a high-speed buffer for storing shared data with common read/write access. It is used by subsystems like database managers to provide shared buffers and maintain buffer coherency across multiple systems.

Cache structures can be used in one of the following three ways:

- ▶ Directory only (IMS OSAM, IMS VSAM) Structure is only used for tracking who is using what data (cross-invalidation), no data is kept in CF.
- ▶ Store-through (IMS OSAM, VSAM/RLS) Data is written to CF and DASD at the same time. All data in CF is considered "unchanged" as the CF copy always matches the DASD copy.
- ▶ Store-in (IMS Fast Path, DB2) Updated data is written to the CF, then hardened to DASD some time later. CF will contain both "changed" and "unchanged" data.

Lock Structures

Lock structures supply shared and exclusive locking capability for serialization of shared resources down to a very small unit of data. The main users of lock structures are GRS (for sharing any type of resource) and the data sharing lock managers (IRLM and VSAM/RLS). Associated with the lock structures are XES lock services, which come into play in case of lock contention.

The user of the request can specify shared or exclusive access. Normally, shared access is associated with reads and exclusive access with a write, but it depends on the program issuing the request. As long as no one has exclusive access to the resource, shared access requests are always granted by the Coupling Facility. If there is anyone with exclusive

access, the allow/disallow decision must be made by the resource serialization manager for that resource.

List Structures

List structures are the most general purpose type of structure and as such are not easy to define in a concise manner. List structures enable authorized applications to share data that is organized in a set of lists for implementing functions such as shared work queues and shared status information.

List entries can be processed as FIFO or LIFO, and access to list entries can be serialized or unserialized. List structures provide the most flexibility and functionality of all the CF structure types.

There are many exploiters of list structures on z/OS. Following are just a few examples:

- ▶ XCF
- ▶ System Logger
- ▶ WebSphere MQ
- ▶ DB2 SCA
- ▶ JES2 checkpoint

Coupling Facility Links

Coupling Facility Links (CF links) are high speed fibre or copper interconnects used to communicate between CF LPARs and z/OS LPARs. In the case of System Managed structure duplexing, CF links are also used for CF LPAR to CF LPAR communication. There are a number of types of CF links.

- ▶ ISC3 - fibre, 200MB/sec, up to 100km - Standard fibre links used on System z
- ▶ ICB3 - copper, 1000MB/sec, 10m, z900 technology - High speed over low distance
- ▶ ICB4 - copper, 2000MB/sec, 10m, z990 technology - High speed over low distance
- ▶ ICP - microcode, 3500MB/sec - Very high speed within one CPC

Timers

The presence of the timers is due to a long standing requirement for accurate time and date information in data processing. As single operating systems have been replaced by multiple, coupled operating systems on multiple servers, this need has evolved into a requirement for both accurate and consistent clocks among these systems. This capability has been provided by the Sysplex Timer device (IBM 9037) and by its follow-on, the Server Time Protocol.

- ▶ Sysplex Timers provide an external time source to all the CPCs in a Sysplex. All systems in a sysplex must operate from the same time source. Time synchronization between all systems in a sysplex is required because there are many instances where events that happen on different CPCs need to be aggregated for processing. An example is a database restart that needs to apply log records from several systems in the correct time sequence.
- ▶ STP is a chargeable feature and is a server-wide facility implemented in the System z microcode and designed to provide the capability of time synchronization between the latest System z machines (z9™ EC, z9 BC, z990, and z890).

STP is designed for servers that are configured in a Parallel Sysplex or a standard sysplex without a Coupling Facility, as well as servers that are not in a sysplex but need to be time-synchronized. STP supports a multi-site timing network of up to 100 Km (62 miles) over fiber optic cabling. STP is a message-based protocol in which timekeeping

information is passed over data links between servers. The timekeeping information is transmitted over externally defined coupling links (ISC or ICB).

z/OS components

Following are the z/OS components that participate in the parallel sysplex solution.

XCF

Cross System coupling facility (XCF) is a software component provided as a part z/OS. XCF performs a number of core functions in a Parallel sysplex.

XCF provides the signaling services in a Parallel Sysplex. Authorized applications can call XCF services to pass messages to other systems in the sysplex. XCF uses signaling paths over high-speed fibre interconnects between systems or through CF list structures. For a detailed discussion on XCF signaling, refer to *z/OS MVS Setting Up a Sysplex*.

XCF is also responsible for maintaining the status of members of the sysplex and for notifying other members if any member enters or exits the sysplex. XCF checks for duplicate member names and that the Sysplex name is correct. XCF uses data sets called Sysplex Couple Data sets to store this status information, and XCF uses its signaling paths to notify other members of the sysplex about events. No new member can join the Sysplex without XCF signaling connectivity to the other systems in the Sysplex.

XES

Cross-system extended services (XES) uses one or more coupling facilities in a Parallel Sysplex to do the following:

- ▶ Provide high-performance data sharing across the systems in a sysplex
- ▶ Maintain the integrity and consistency of shared data
- ▶ Maintain the availability of a sysplex

XES is an extension to the XCF component. XES provides the sysplex services that applications and subsystems use to share data held in the coupling facility. These services support the sharing of cached data and common queues while maintaining data integrity and consistency. The cross-system extended services (XES) component enables applications and subsystems to take advantage of the coupling facility. In fact, any subsystem on z/OS, such as DB2, that needs to perform activity against the CF structures uses XES services. XES handles structure build, rebuild, alter, deletion, and system managed duplexing. XES also plays a critical role in lock management in a Sysplex. XES acts as the Global Lock Manager on a system for exclusive locks held by resource managers on that system.

SFM

Sysplex Failure Management (SFM) is a z/OS component that helps managing failures in a Sysplex. SFM allows you to define a sysplex-wide policy that is used to automatically manage recovery actions in the event of failures in different elements in the Parallel Sysplex.

Following are examples of the type of failures that SFM can manage:

- ▶ XCF connectivity failures between members of the sysplex
- ▶ Missing status updates on the CDS
- ▶ H/W resource reconfiguration in the event of one or more LPARs being halted.

ARM

Automatic restart management is a z/OS component that is key to automating the restarting of subsystems and applications so that they can recover work they were doing at the time of an application or system failure. They can then release resources, such as locks, that they were holding. With automatic restart management policy, you can optionally control the way

restarts are done. You can use the policy defaults that IBM provides, or you can override them.

In a sysplex environment, a program can enhance its recovery potential by registering as an element of automatic restart management. Automatic restart management can reduce the impact of an unexpected error to an element because it can be restarted automatically without operator intervention. In general, z/OS restarts an element under the following conditions:

- ▶ When the element itself fails. In this case, z/OS restarts the element on the same system.
- ▶ When the system on which the element was running unexpectedly fails or leaves the sysplex. In this case, z/OS restarts the element on another system in the sysplex. This is called a cross-system restart.

In general, your installation may use automatic restart management in two ways:

- ▶ To control the restarts of applications (such as CICS) that already use automatic restart management as part of their recovery.
- ▶ To write or modify installation applications to use automatic restart management as part of recovery.

Automatic restart management manages subsystems and restarts them appropriately after either subsystem-level or system-level failures occur.

WLM

The z/OS Workload Manager (WLM) component introduces the capability of dynamically allocating or re-distributing server resources, such as CPU, I/O, and memory across a set of workloads based on user defined goals and their resource demand within a z/OS image. Looking over the fence of the z/OS image, the Workload Manager is able to perform this function also across multiple images of z/OS, Linux, or VM operating systems sharing the System z processor. Another function of the WLM is to assist routing components and products to dynamically direct work requests associated with a multisystem workload. This is to run on a z/OS image within a Parallel Sysplex that has sufficient server resources to meet customer defined goals. For example, WLM can assist with recommendation values, the sysplex distributor capability, the VTAM® generic resource function, the CICS dynamic routing mechanism, and other routing and balancing function available in a Parallel Sysplex configuration.

Subsystems

Developers of database subsystems, transaction servers, and messaging systems can use a standard API to make their subsystems parallel sysplex enabled. This means that they do not have to write and support the underlying services to perform the basic global lock management or global buffer management. These are all provided by z/OS software and System z hardware.

Following are examples of IBM supplied subsystems that exploit Parallel Sysplex:

- ▶ IBM DB2 for z/OS
- ▶ IMS
- ▶ WebSphere MQ
- ▶ VSAM/RLS
- ▶ CICS Transaction Server
- ▶ WebSphere Application Server for z/OS

3.4 DB2 for z/OS data sharing implementation

DB2 for z/OS is the IBM relational database solution on System z. It was one of the early subsystems on System z to add support for Parallel Sysplex data sharing and has been at the forefront of Parallel Sysplex exploitation for more than 10 years.

As we will discuss in detail in the next part of this section, DB2 for z/OS implements a shared cache cluster model. Multiple DB2 subsystems running on multiple z/OS systems across multiple CPCs are presented to applications as one single database instance. Applications can access any member of the DB2 data-sharing group with full read/write access to all data.

Like any subsystem on z/OS that wants to exploit Parallel Sysplex data sharing, DB2 uses the z/OS services (XCF and XES) to interact with the CF for the lock and cache management required for a shared-data cluster.

Figure 3-2 is a logical representation of a DB2 data sharing group with two members.

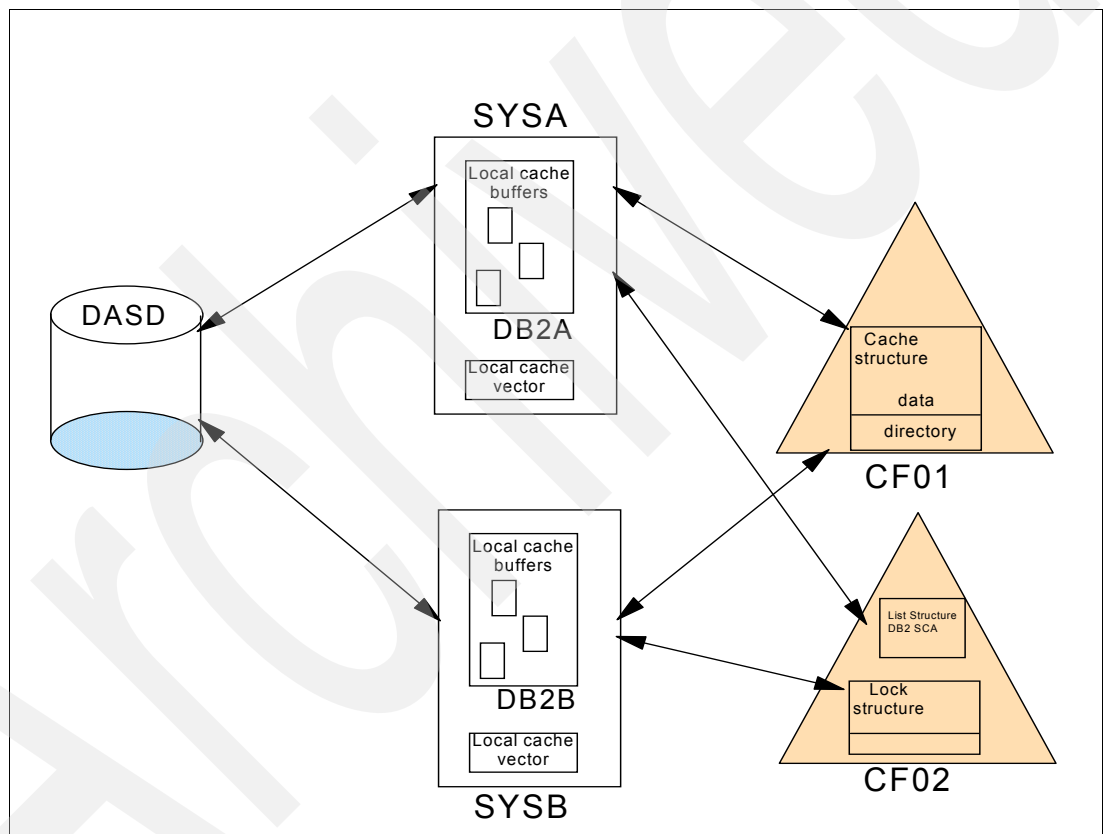


Figure 3-2 DB2 data sharing group logical view

In this example, there are two z/OS images SYSA and SYSB that could be running either as LPARs on the same CPC or in different CPCs. Each z/OS image runs a DB2 for z/OS subsystem that is configured as a member of the DB2 data sharing group. The DB2 data sharing group are relying on data in the coupling facility to share the data. This sample configuration uses two coupling facilities that represents the recommended configuration for highest availability.

3.4.1 DB2 for z/OS terminology

Data sharing group: The collection of DB2 subsystems that share system catalog and data tables.

Member: A single DB2 instance that is a part of a data sharing group. Each DB2 subsystem has its own active and archive logs and BSDS but shares a common DB2 catalog, directory, and data tables with all other members in the group.

3.4.2 Storage model

DB2 for z/OS implements a shared-cache cluster model. As discussed in the section 2.2.2, “Shared-everything model” on page 10, a shared-cache implementation must provide global lock management and global cache management. This section analyzes each of them and discusses how they are implemented in a DB2 data-sharing group. Make sure that the Shared-all model concepts are clear before continuing since the following explanations assume a good understanding of the concepts behind a shared-cache cluster; otherwise, you can refer to the section 2.2.2, “Shared-everything model” on page 10 for details.

Lock management

Each DB2 subsystem runs a lock manager called IRLM. IRLM manages locking for that DB2 subsystem. For further details and a general description of the DB2 locking and IRLM mechanism, refer to the *DB2 for z/OS Administration guide*, which gives a very good overview of DB2 locking in general.

Locking model

When the DB2 subsystem is part of a data sharing group in a parallel sysplex, IRLM in conjunction with XES and the CF lock structure participate in the global transaction locking mechanism required to maintain data integrity. Each DB2 member is connected to the same lock structure in the CF and this lock structure holds lock information for that data sharing group. This model can be described as a centralized global-locking model, which we described in section “Lock management” on page 26.

P-Locks and L-Locks

In a single DB2 system environment we are usually just concerned with transaction locking, for example locks held for DB2 resources to maintain data integrity during a transaction. In a data sharing environment there are two types of locks: the traditional transaction locks (referred to as L-locks) and physical locks (referred to as P-locks).

P-locks are only used in a data sharing environment and relate to physical objects that can be held in buffers (table space, indexspace or partitions). They are required for global cache management; for example, you may have row-level locking for a table holding a lock against row 1 of that table. If the page in which the row is held resides in the Group Buffer Pool, then the buffer management needs to protect the physical page, requiring a P-lock on that page to indicate to the global cache manager that this DB2 member has an interest in that physical page.

P-Locks are not directly related to transactions. P-locks are related to caching, not concurrency, and they use different mechanisms than the transaction locks you are familiar within DB2. They are owned by a DB2 member not by an individual work unit. Also, DB2 members in the data sharing group may negotiate changes in the P-lock mode (exclusive to shared for example) depending on the conditions at the time.

Lock avoidance

In a data sharing environment locks need to be propagated beyond the local IRLM but this is done only where there is an inter DB2 read-write interest. DB2 provides a number of optimizations to reduce the number of locks that need to be propagated to the CF, such as the following:

- ▶ Explicit hierarchical locking makes certain optimizations possible. When no inter-DB2 read/write interest exists in an object, it is possible to avoid processing certain locks beyond the local IRLM.
- ▶ If a single member with update interest and multiple members with read-only interest exist, DB2 propagates fewer locks than when all members have update interest in the same page set.
- ▶ All locks (except L-locks) that are held beyond the local IRLM are owned by a member, not by an individual work unit. This reduces lock propagation by requiring that only the most restrictive lock mode for an object on a given member be propagated to XES and the coupling facility. A new lock that is equal to, or less restrictive than, the lock currently being held is not propagated.
- ▶ When the lock structure is allocated in a coupling facility of CFLEVEL=2 or higher, IRLM can release many locks with just one request to XES. This can occur, for example, after a transaction commits and has two or more locks that need to be unlocked in XES. It also can occur at DB2 shutdown or abnormal termination when the member has two or more locks that need to be unlocked.

DB2 lock structure

The DB2/IRLM CF lock structure contains both lock entries and record data entries. The lock entries are held in a part of the CF lock structure called the lock table. These lock entries indicate whether inter-DB2 read/write interest exists on a particular resource. Hashing algorithms hash resource names to a slot in this table.

Information in the record data entries (IRLM calls them Record List Entries) is kept purely for recovery. In case of failure, IRLM asks the CF for a list of all RLEs with IRLM connection token. Each of these is then changed to a Retained Lock Entry. If an IRLM disappears, the other members of the data sharing group do the same processing on the disappearing IRLM's behalf—this ensures that no one else can update any items that were locked by the failing IRLM at the time it failed.

Deadlock detection

In a data sharing environment, deadlocks can occur between transactions on different members. The term global deadlock refers to the situation where two or more members are involved in the deadlock. Local deadlock refers to the situation where all of the deadlocked transactions reside on a single member.

DB2 has a global deadlock detection process to handle these situations. It requires the participation of all IRLM members in the data sharing group. Each IRLM member has detailed information about the locks that are held and are being waited for by the transactions on its associated DB2 member. However, to provide global detection and timeout services, each IRLM is informed of all requests that are waiting globally so that the IRLM can provide information about its own blockers. That IRLM also provides information about its own waiters. The IRLM members use XCF messages to exchange information so that each member has this global information.

To coordinate the exchange of information, one IRLM member assumes the role of the global deadlock manager. As IRLM members join or leave the group, the global deadlock manager might change.

Example scenario

Figure 3-3 shows the sequence of locking actions required in a DB2 data sharing group with two members.

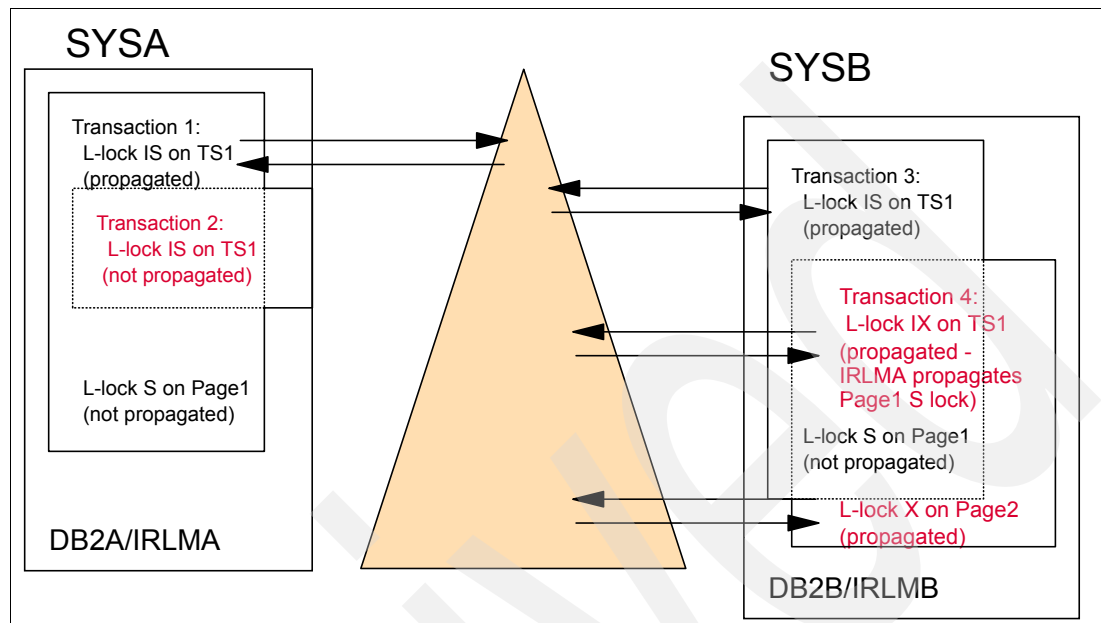


Figure 3-3 DB2 locking scenario

You can observe the following in Figure 3-3:

1. The L-lock on tablespace 1 (TS1), which is associated with transaction 2, is not propagated because DB2A already holds a lock of an equal restriction on that object. (The L-lock on TS1, which is associated with transaction 3, is propagated because that lock is from another member.)
2. The child L-lock on page 1, which is associated with transaction 1, is not propagated at the time that it is requested because its parent lock is IS on both members: no inter-DB2 read/write interest exists on the parent.
3. When Transaction 4 upgrades its lock to IX on TS1, its X lock on Page 2 must be propagated because inter-DB2 read/write interest now exists on the parent. Also, the child lock on Page 1, which is associated with transaction 1, must be propagated.

The child lock is propagated under an IRLM SRB not under the transaction's TCB. This propagation is counted in the statistics report as an asynchronous propagation.

Cache management

As discussed in Chapter 2, "Clustering basics" on page 5, database systems normally use data buffering to achieve the high levels of performance required for transactional systems.

A DB2 subsystem uses Buffer Pools (BPs) for local caching. DB2 must have at least one buffer pool for the 4K page size, the 8K page size, the 16K page size, and the 32K page size. You can define multiple buffer pools for any particular page size. It is normal to have separate buffer pools for system catalog, user tablespaces, and user indexspaces.

In a data sharing environment, every DB2 buffer pool that will be used by data to be shared must map to a cache structure in the CF called a Group Bufferpool (GBP). Applications can access data from any DB2 subsystem in the data sharing group. Many subsystems can

potentially read and write the same data. DB2 uses special data sharing locking and caching mechanisms to ensure data consistency.

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

When there is inter-DB2 read/write interest in a particular table space, index, or partition, it is dependent on the group buffer pool, or GBP-dependent (group buffer pool dependent). For example, each DB2 has a buffer pool named BP0. For data sharing, you must define a group buffer pool (GBP0) in the coupling facility that maps to buffer pool BP0. GBP0 is used for caching the DB2 catalog and directory tablespaces and indexes, and any other tablespaces, indexes, or partitions that use buffer pool 0.

Although a single group buffer pool cannot reside in more than one coupling facility (unless it is duplexed), you can put group buffer pools in more than one coupling facility.

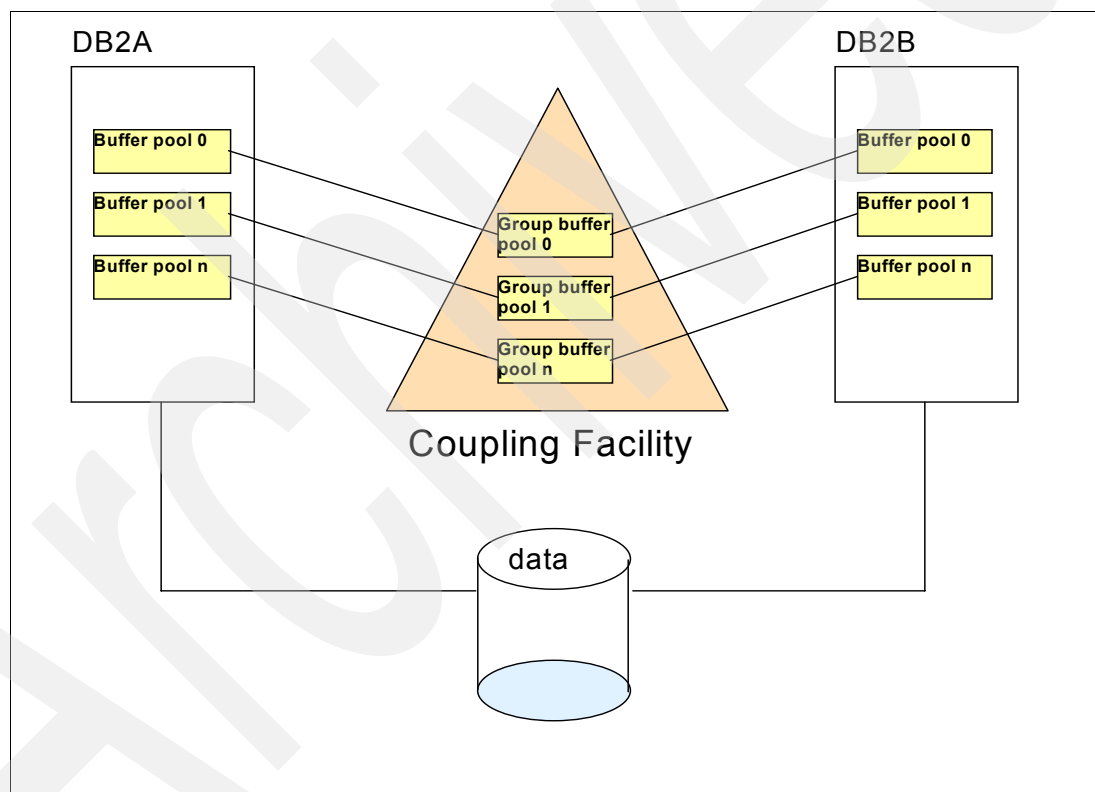


Figure 3-4 Buffer pools mapping to Group Buffer pools

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

DB2 allows an installation to decide whether the cache structure is to be used to cache all data, updated data, or is just to be used for cross invalidation purposes. This effects how much data is held in the GBP and when the GBP is allocated. If you choose to cache changed data only, the GBP cache structure will only be allocated the first time DB2 detects

whether there is an inter-DB2 read/write interest for any DB2 object (row, page and so on) that is associated with that particular buffer pool.

Cross invalidation

DB2 uses a local vector cache in the HSA to check if a local buffer is still valid.

Cache coherency

This section explains how cache coherency is maintained in a DB2 data sharing group. We work through a typical scenario, for example an application requesting a row of data in a two systems environment.

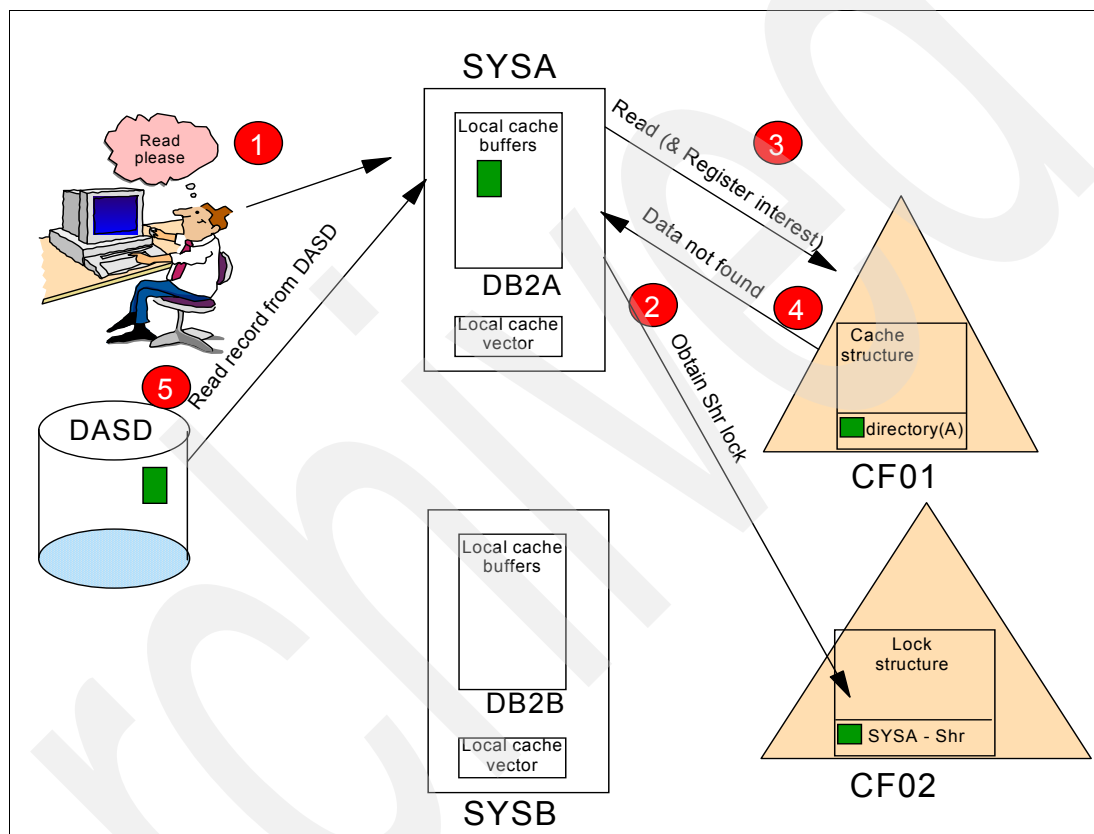


Figure 3-5 Sample scenario - reading a DB2 row

DB2 has a number of optimizations that allow it to check whether any other DB2 member in the data sharing group has an R/O or R/W interest in the page set where the row resides. In this case we assume that there is a P-lock already held on the page set where the requested row resides. Because of this, DB2A will process this read as follows:

1. The application requests a read to DB2 member DB2A running on system SYSA.
2. IRLM obtains a read lock, and this is propagated to the lock structure in one of the CFs.
3. DB2A issues a Read and Register interest to the GBP structure on CF01. This is to see if the page already resides in the cache structure and to register the fact that DB2A now has a local copy of that page R/O. That interest information is held as a GBP directory entry.
4. DB2A is informed that there is no copy of that page in the GBP.
5. DB2A reads the physical page from disk.

At this stage we now have a copy of 4K page containing the row held in local buffers on DB2A. There is an entry in the GBP cache structure indicating that DB2A has a local copy of that page and that there is an entry in the lock structure indicating a transaction lock on the row we have read (assuming row level locking).

As described in Figure 3-6, let us see what happens when the application connected to DB2A wants to update the row that was read in the previous step.

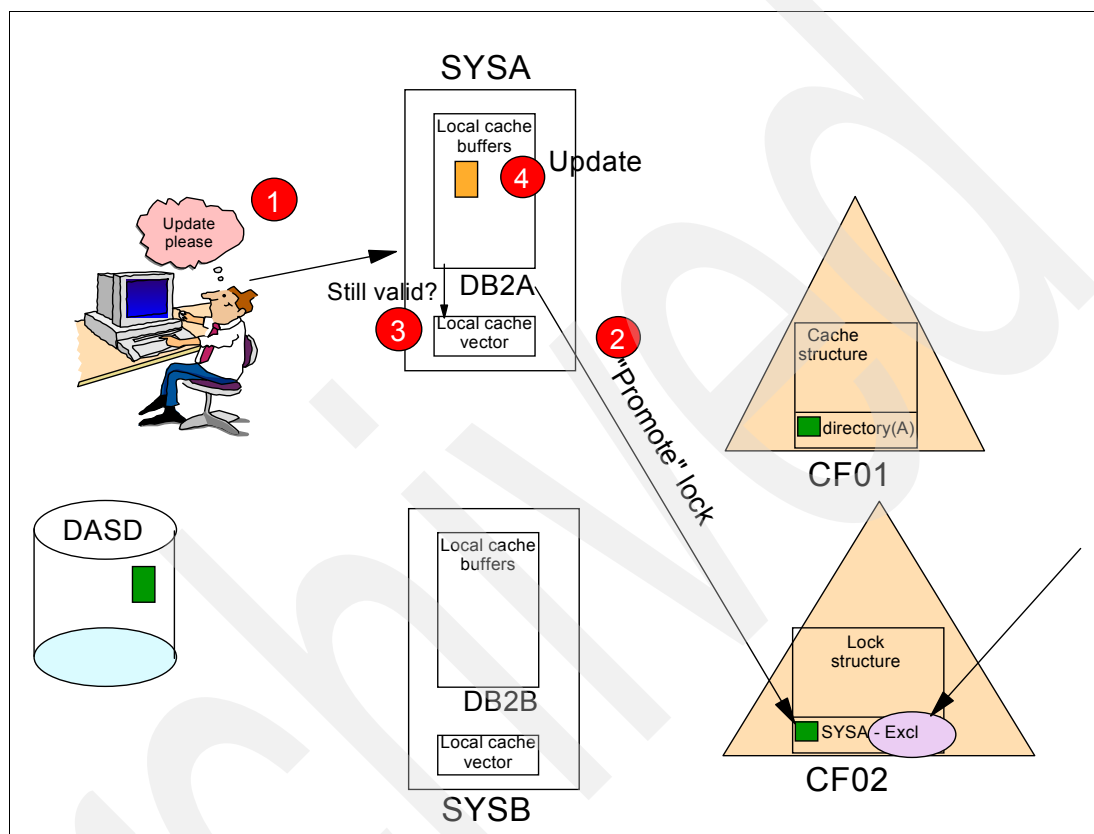


Figure 3-6 Sample scenario - updating a DB2 row

1. The application issues an update request to DB2A on the row previously read.
2. DB2A promotes the share lock to exclusive.
3. DB2A checks its local cache vector table to make sure that the copy of the page in its local buffer is still valid. Why does it need to do this? Well it could be possible that another DB2 member has updated a row on that same 4K page. In which case an updated copy of the 4K buffer would reside in the GBP cache structure and the CF would have already sent a cross-invalidate signal that would set the status of that 4k buffer as invalid in the local cache vector on SYSA. Let us assume in this case that the buffer is still valid.
4. DB2A can now update the row in 4K page buffer.

At this point the application has not yet issued a commit, so the locks are still held and the update exists in a local cache buffer.

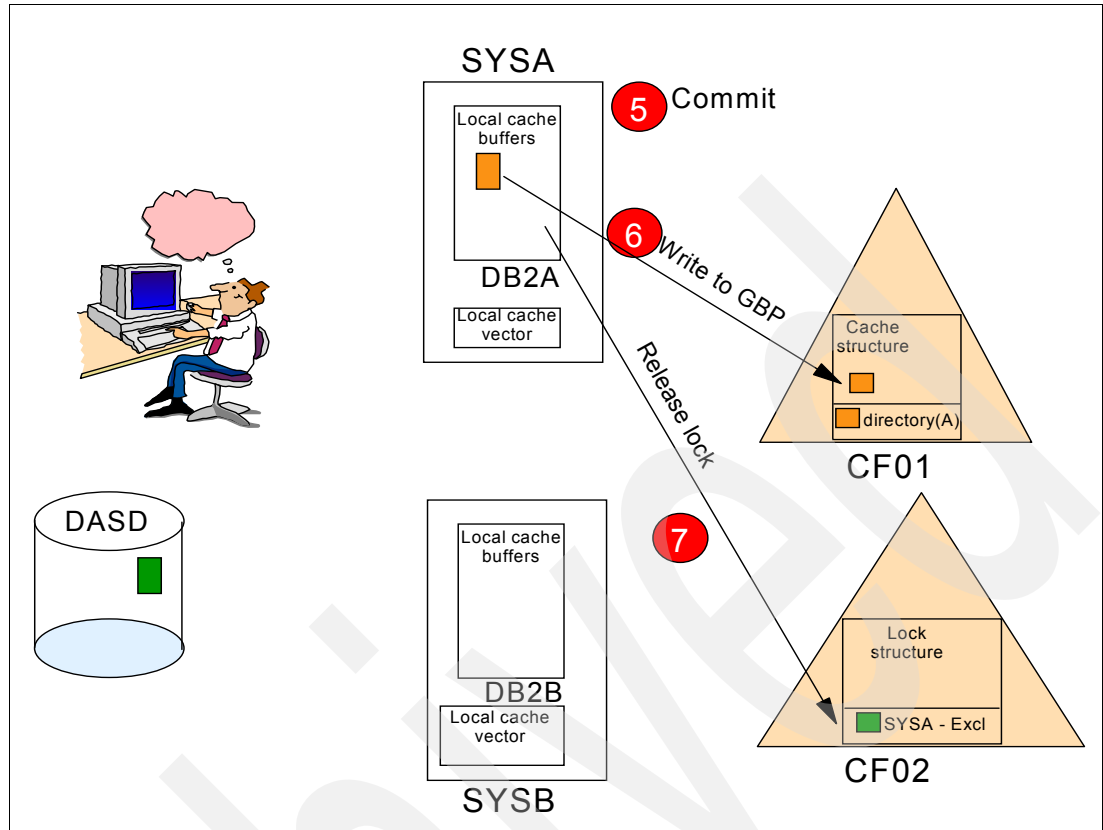


Figure 3-7 Sample scenario - committing the updates

As illustrated in Figure 3-7, the application now commits the update.

1. DB2A receives the commit unit or work containing the update. DB2A performs normal commit processing. Update is hardened to active log for roll-forward recovery in the event of DB2A failure. Updated 4k page is not written to disk for performance reasons. Update to disk will occur at some stage in the future as part of buffer cleanup.
2. As part of the commit processing, DB2A now writes the updated 4k page to the GBP cache structure. This is done so that it is available to other members of the DB2 data sharing group. This step is required for recovery processing in the event of DB2A failure before the updated buffer is written to disk.
3. The transaction (L-lock) is released.

At this point there is an updated copy of the 4k page held in local buffer on DB2A. There is a copy held in the GBP cache structure. All transaction locks are released, but there is still a P-lock held on the 4K page by DB2A. If nothing happens that requires this 4k page, then it will be written to disk after a certain interval.

Let us assume, though, that an application connected to DB2B does a read requests for that same row, as you can see in Figure 3-8.

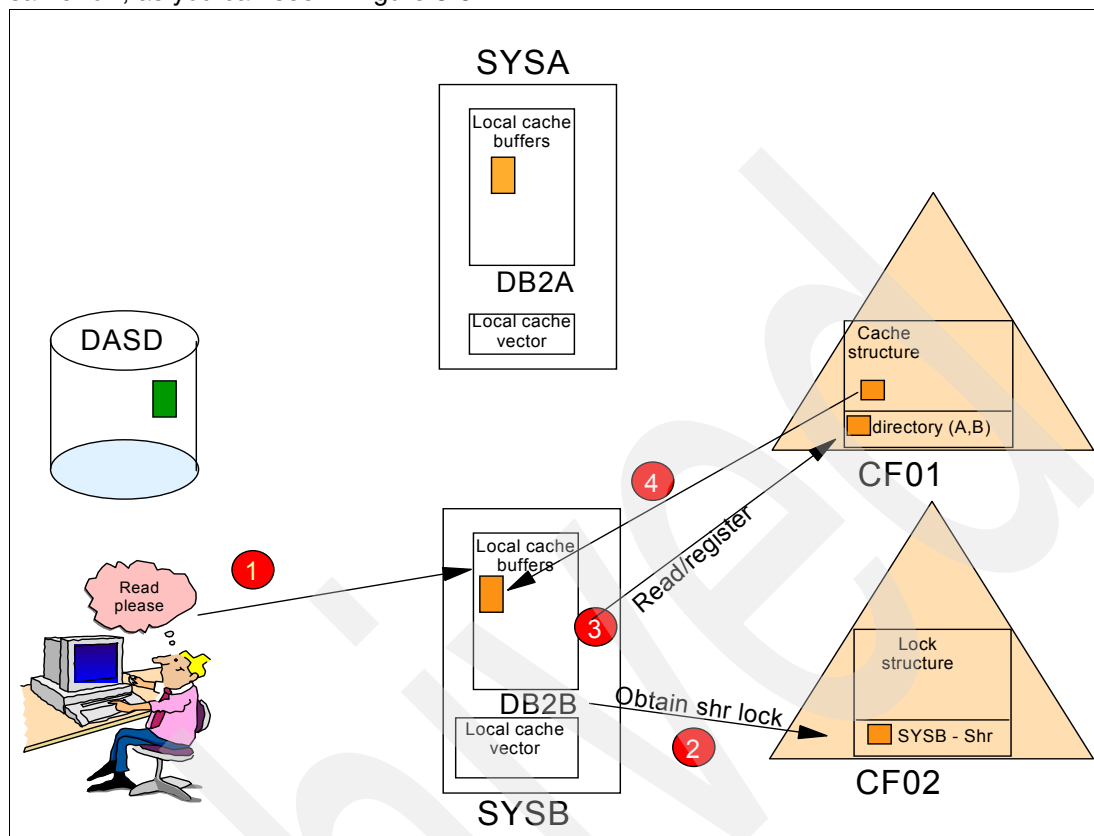


Figure 3-8 Sample scenario - reading a DB2 row

1. The application makes a request to DB2B on system SYSB to read a row on the same 4k page that was updated by DB2A in the previous steps.
2. IRLM obtains a read lock, and this is propagated to the lock structure in one of the CFs.
3. DB2B issues a Read and Register interest to the GBP structure on CF01. This is to see if the page already resides in the cache structure and to register the fact that DB2B now will have a local copy of that page R/O. That interest information is held as a GBP directory entry.
4. The 4k page is found and the CF passes it back to DB2B where it is placed in the local buffer. DB2B can now read the row and return it to the application.
5. The application commits and releases the shared lock.

At this stage the 4K page is held in the local buffer on DB2A. The local buffer on DB2B and the GBP in CF01. The 4K page is registered to both DB2A and DB2B through the directory entries in the GBP cache structure. This means the CF is aware that there are local copies of this 4k page held locally in DB2A and DB2B. This is a important for the following steps.

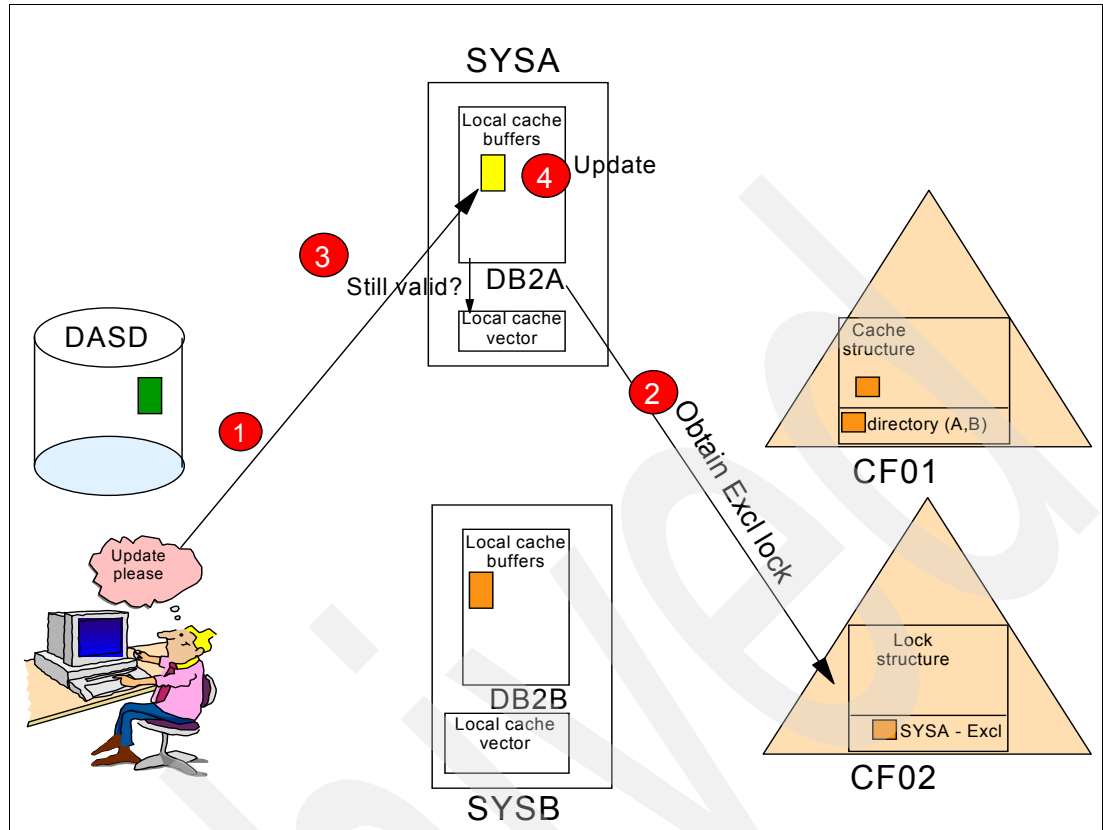


Figure 3-9 Sample scenario - updating a DB2 row

1. The application sends requests to DB2A to update a row on the same 4k page that was previously updated.
2. IRLM for DB2A obtains a read lock, and this is propagated to the lock structure in one of the CFs.
3. The 4k page already exists in the local cache on DB2A. Before DB2A can update it, it must verify that its copy is still up to date. This is done by checking its local cache vector in HSA. This confirms that the local copy of this buffer is still valid.
4. DB2A can now update the row contained on our 4K page.

At this point the update is not committed. The 4k pages with the update are only held in local cache on DB2A. The application now issues a commit of the unit of work.

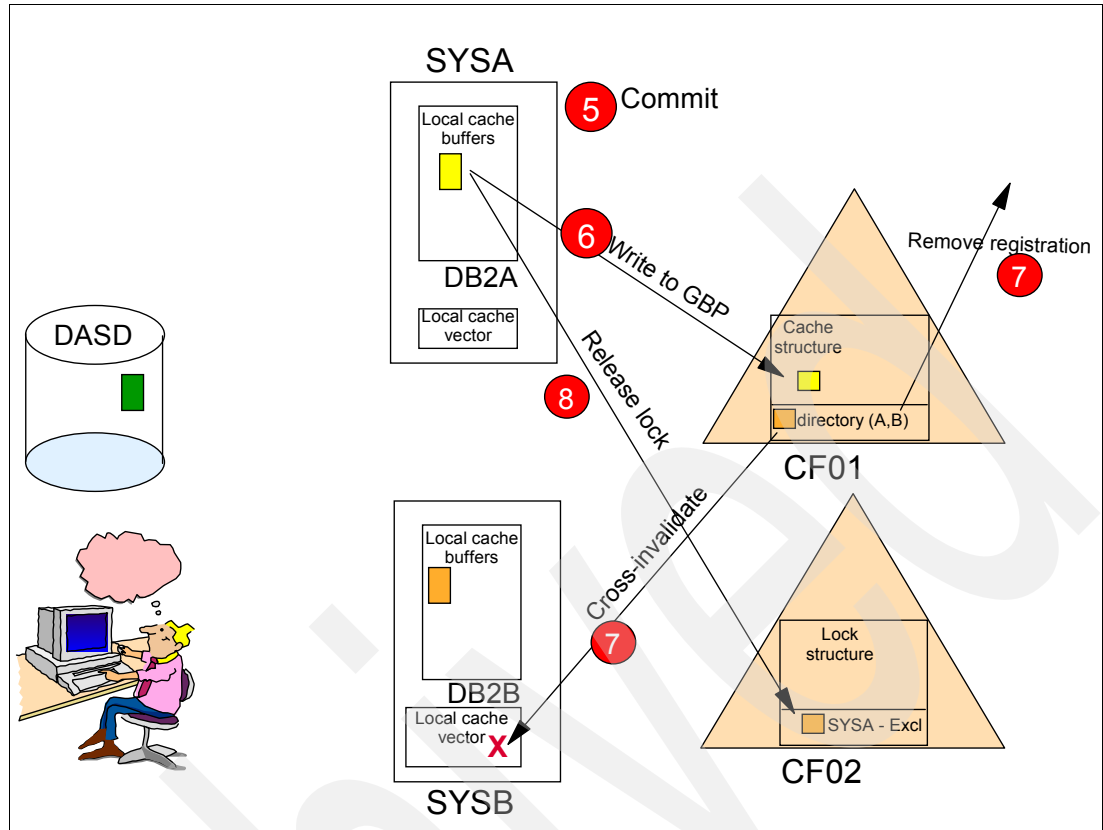


Figure 3-10 Sample scenario - committing a DB2 row

1. The application issues a commit.
2. As part of the commit processing, DB2A writes the updated buffer to the GBP cache structure.
3. The CF sees that another system has a copy of that buffer in its local cache and sends a signal over the CF link that causes the local cache vector on SYSB to mark its local copy of the buffer as invalid. The cache directory is updated to reflect the fact that DB2B no longer has a valid copy of the buffer in the local cache
This process is called cross-invalidation. It is performed by the CFCC in combination with the CF link microcode. There is no interrupt to DB2B, so this process is extremely fast with no processing overhead on z/OS.
4. The Lock on the row is released.

At this point, DB2A has a valid copy of the updated buffer. The updated buffer is also in the GBP cache structure. The old copy of the buffer is still DB2B's local cache, but it was being flagged as invalid in the local cache vector on SYSB.

Figure 3-11 on page 36 and the following explanation shows what happens when DB2B tries to read its invalid buffer as an application requests a row that resides on the same page set.

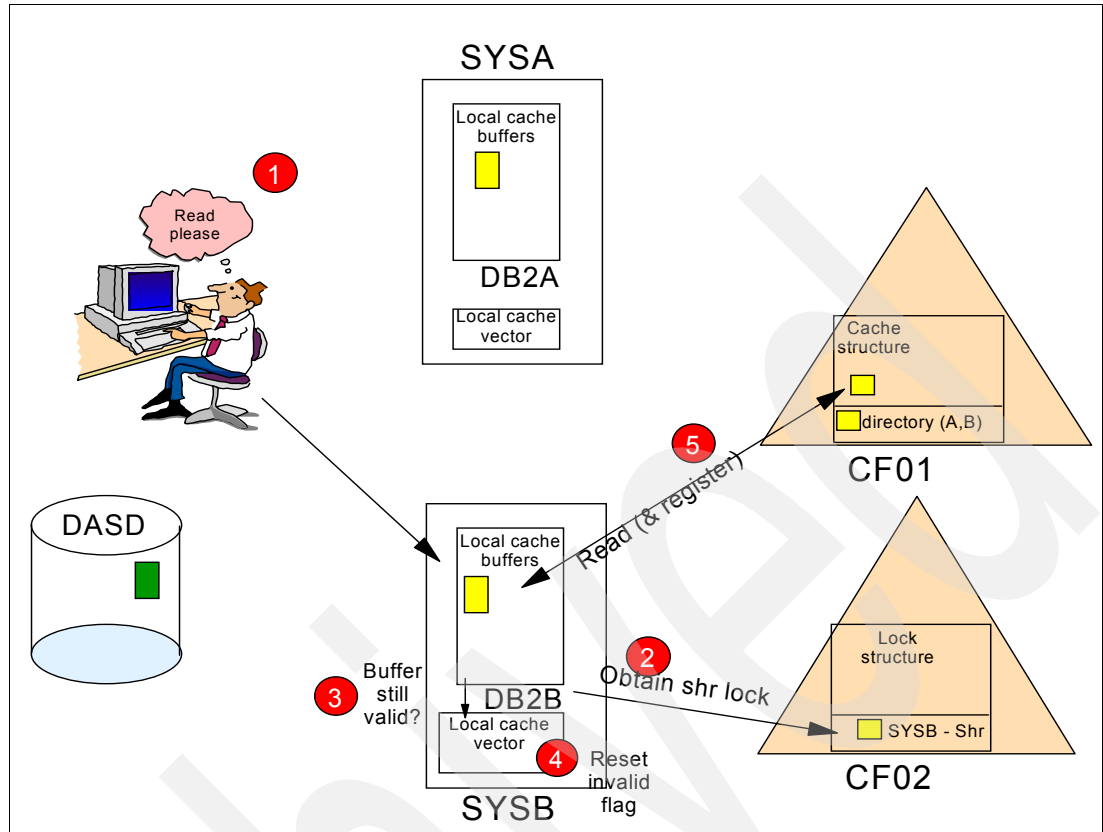


Figure 3-11 Sample scenario - reading a DB2 row from an invalidated buffer

1. An application issues a read for a row on the buffer to DB2B.
2. DB2B obtains a transaction lock on the row that gets propagated to the CF lock structure.
3. The DB2B buffer manager knows it has a copy of the buffer in its local cache, but before using this copy, it checks its local cache vector to see if this copy of the buffer is valid. If found invalid, DB2 knows that the copy from the buffer in the GBP cache structure is necessary.
4. DB2B resets the invalid flag in its local cache vector.
5. DB2B issues a Read and Register to the GBP cache structure. The CF returns the buffer and updates the cache directory to indicate that DB2B now has a copy of the buffer in its local cache.

At this point there is an up-to-date copy of the buffer in the local cache in DB2A, the local cache in DB2B, and the GBP cache structure in the CF. The copy of the buffer on disk is not up-to-date, but the update was logged by DB2A in its logs so that it can be reapplied in the event of a failure.

The updated buffer will be written to disk at some later stage, though with some additional steps compared to a non-data sharing environment. In data sharing, one of the DB2 members performs a castout. The castout process writes the changed buffers to disk.

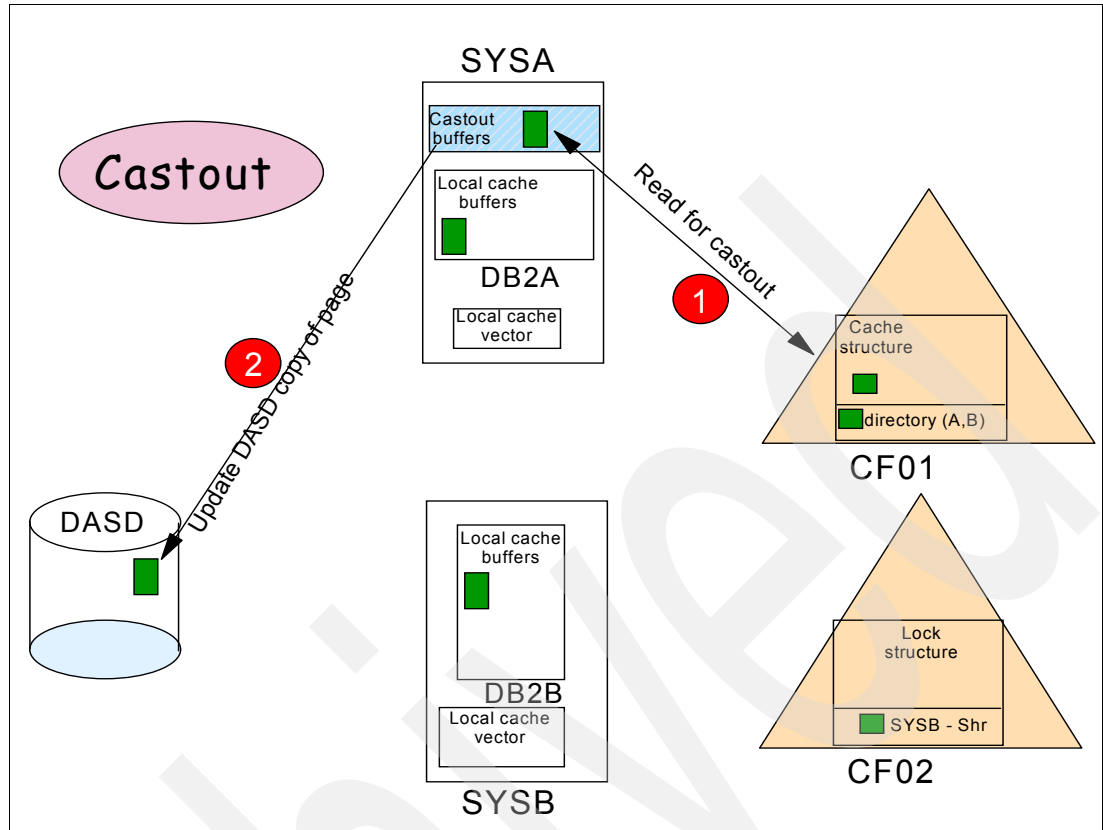


Figure 3-12 Sample scenario - castout process

Figure 3-12 shows a sample of the castout process where one of the DB2 members will be the castout owner for a buffer.

1. DB2A reads the buffer from the GBP cache structure into a castout buffer in DB2A.
2. DB2A writes the updated copy to DASD.

At this point, an updated copy of the buffer resides everywhere, local cache, GBP cache, and DASD. The buffer is not deleted from GBP yet and is still available for any member to read it.

When the cache structure reaches the full percentage threshold, the reclaim processing occurs. Reclaim processing is invoked to free up GBP cache structure space by removing buffers that are marked as unchanged. The CF uses a LRU algorithm to decide which pages should be freed up.

The reclaim processing works in different ways depending on which space needs to be freed up, data space or directory space in the structure. If data space needs to be freed up, the buffer is removed, but directory entries will be maintained to indicate which DB2 members have a copy of the page in their local cache. This still provides performance benefits even if the buffer itself no longer resides in the GBP. If directory space needs to be freed, both the directory entry and the data entry are removed. In Figure 3-13 on page 38, we assume reclaim processing is being run to free up directory space in the GBP cache structure that contains our buffer. We also assume our buffer is one of the ones chosen to be reclaimed.

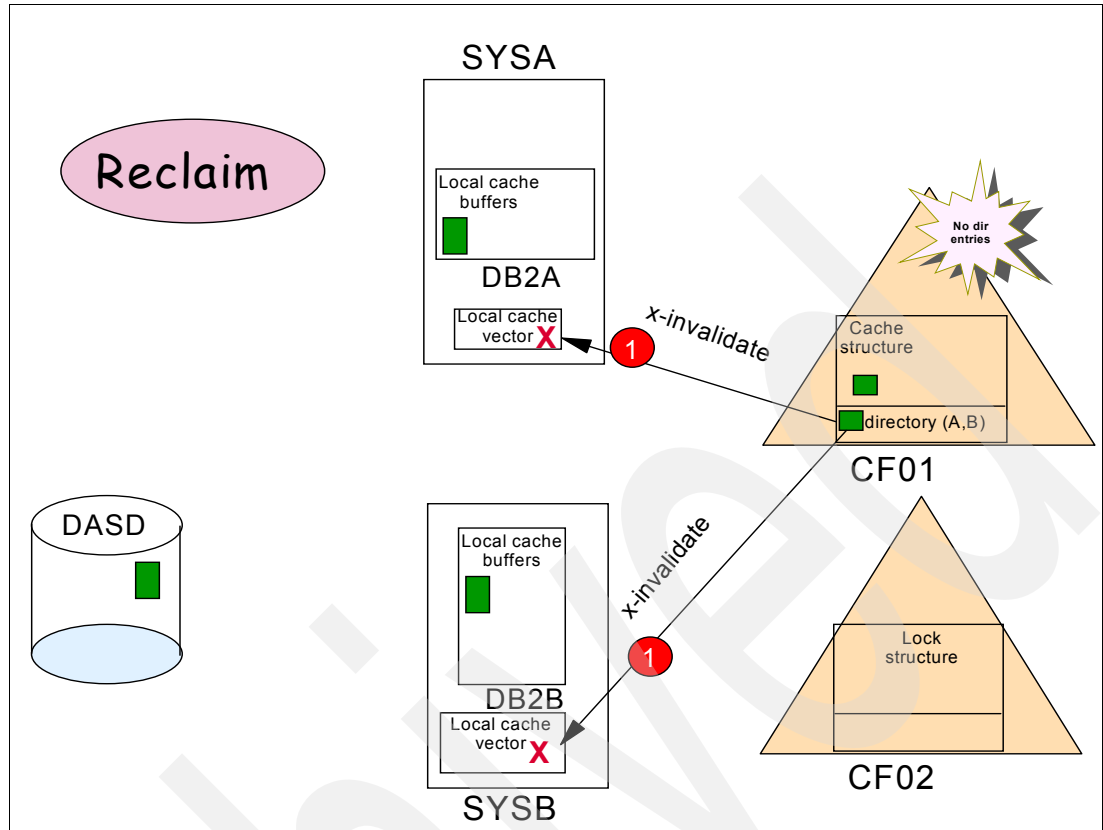


Figure 3-13 Sample scenario - reclaim process

Reclaim processing is initiated to free up directory entries. Directory entries for buffer are deleted and cross invalidate signals are sent to all systems that have the buffer in their local cache. This is done so that the buffer copies in the local cache are not used again.

3.4.3 Performance

Although the first priority for any shared data implementation is maintaining data integrity, performance is also a high priority to be of practical use. In the past, performance issues were the main reason for limiting the shared database implementations because the overhead of continuous DASD updates cannot be tolerated in some online applications where usually the transaction rate is very high, as well as the update ratio.

Parallel Sysplex was designed from the very beginning to provide an extremely high-performance shared data infrastructure. The combination of specialized hardware, microcode, and tight integration with the operating system is unique in the industry.

The main reasons why Parallel Sysplex can solve the performance issues in a data sharing environment are as follows:

- ▶ **Coupling facility** - The CFCC is not interrupt driven. It runs in a continuous pooling loop and provides a centralized shared storage. This means that there is a very little increase in communication overhead as you increase the number of nodes. Nodes talk to the coupling facility not to each other, and the star topology dramatically reduces the cost of adding nodes.
- ▶ **CF links** - The CF link HW is specialized to provide a very high-performance link to the CF using a very specific communication protocol.

- ▶ z/OS integration - A big advantage of a proprietary system is that the HW and the operating system can be tightly integrated.
- ▶ Locking optimizations - DB2 is designed to reduce the cost of data sharing enablement as much as possible. Various optimization techniques and algorithms have been implemented to enable you to take full advantage of your Parallel Sysplex environment. However, the data sharing enablement overhead is sensitive to the system and application environment used. Several design and configuration options enable you to further reduce the cost.

The path length of acquiring a lock from the IRLM compared to acquiring a latch is considerably higher. The elapsed time of an interaction with the CF is even longer than a request to the IRLM even though most of the processing is performed synchronously, assuming the CF structure response time is adequate. Propagating a lock to the CF is more expensive than latching a page. Every lock you can avoid improves the overall resource consumption in a data sharing environment. The impact of lock avoidance is therefore more important than in a non-data-sharing environment.

Every time you acquire a lock, you could have contention on the object. This contention can be due to the granularity of the lock modes of z/OS XES. Resolving this contention can be a lengthy process that might have to be repeated if the lock released is acquired again. A way of reducing the overhead is by hanging on to that lock for a longer period of time, but that could be contradicting your concurrency requirement.

- Maximize lock avoidance - The number of child locks that need to be propagated to the CF affects the cost of global locking. If the L-lock on the parent is IX, even S locks on children must be propagated to the CF if there is inter-DB2 R/W interest. Processing a large number of qualifying rows could cause a considerable overhead. Lock avoidance reduces the impact of child lock propagation by reducing the number of S locks. Lock avoidance, if successful, eliminates calls to the IRLM to request a page lock or row lock.

The efficiency of lock avoidance depends on three parameters:

- The CURRENTDATA value
- The ISOLATION level of your plan and package
- The highest committed global log record sequence number (GCLSN)
- Maximize thread reuse - With DB2 V8, the need for binding plans with RELEASE(DEALLOCATE) to avoid XES contention was significantly reduced. In DB2 Version 8, IX level tablespace intent locks, and page set P-locks, are registered in the XES as being an S lock. The net effect is a significant reduction of the number of invocations of the contention exit because of the avoidance of XES contention. You might decide to continue to use RELEASE(DEALLOCATE), but be aware of the potential side effects.

There may be a trade-off in concurrency and EDM pool requirements. Using RELEASE(DEALLOCATE) can increase EDM pool storage requirements as you hold on to your plans and packages for a longer period of time. This is especially the case if one plan is used for a large number of packages. For example, using just one plan for all transactions in one CICS results in a large number of packages using the same thread. Unplanned lock escalation has a dramatic impact, because with RELEASE(DEALLOCATE) the escalation will stay until thread deallocation, which might take a long time and could prohibit application concurrency. Problems can arise if you have processes that require exclusive access, such as vendor utilities or batch processes. They can break into long-running SQL threads through drain or by issuing a -STOP DB... AT COMMIT.

- Use uncommitted read isolation - UR isolation has the lowest locking cost, since it does not acquire parent or child L-locks. It only acquires a mass delete lock. If your application can manage the potential inconsistency or if your data is guaranteed to be consistent, you should exploit UR isolation to reduce the cost of locking. UR isolation is available at the package/plan level and also at the statement level.
- Use row locking with caution - Row locking can have a considerable impact in a data sharing environment. The cost of row locking can be significantly higher than page locking if DB2 cannot use lock avoidance efficiently and you process a number of rows per page.

Independent of the cost, the degree of concurrency might not immediately benefit from row locking in a data sharing environment. The physical consistency on a page is guaranteed through the use of a page P-lock, the data sharing latch. Even if you use row locking, you cannot simultaneously update two distinct rows in the same page, since there is no such thing as a row P-lock. It might be better for the overall throughput if you use the MAXROWS 1 option in combination with page locking, which is the equivalent of a row lock. MAXROWS N does not have to be MAXROWS 1. Simply reducing the number of rows per page can be sufficient to avoid the peak of the contention curve.

Consider the MAXROWS option if the number of rows in the table is limited and to help the situation if you experience a high degree of contention on a page without it.

3.4.4 Availability - restart and recovery

The strength of a good design is to be found not when things are going well but when things fail. Failure handling in a shared database cluster requires some planning because of the multiplicity of failure scenarios and elements. The most important recovery issue is being able to deal with lock remastering and partial writes (where a log update was done at commit but the updated buffer is still held in local cache).

In this section, we discuss how DB2 for z/OS handles various failure scenarios.

Failure of one member

There were a number of enhancements to DB2 restart processing in recent years. Following is an overview of those enhancements.

Normal restart

During a normal DB2 restart, with no special options, DB2 uses the log and BSDS to determine what to recover. For systems that do not share data, the restart process consists of the following four phases:

- ▶ Phase 1: Log initialization
- ▶ Phase 2: Current status rebuild
- ▶ Phase 3: Forward log recovery
- ▶ Phase 4: Backward log recovery

Restart Light

Restart Light allows you to restart the DB2 member on another LPAR purely for the purpose of freeing the retained locks, after which the member is terminated. It was introduced by DB2 Version 7. This type of restart is useful in data sharing if you have an LPAR failure and the DB2 member that was running in that LPAR has retained locks in the Coupling Facility. The *light* restart requires a much smaller amount of storage than is usually required to restart a DB2 member.

This means that you can restart the member on a different LPAR that would normally not be able to handle that DB2 member. The purpose of the restart is to free the retained locks. Once the locks are freed, DB2 automatically terminates the member. You can bring that member up later on the original LPAR once it is available again.

Postponed recovery

We recommend that you design your applications with frequent commit points to minimize the amount of backout processing required at restart. For example, if you have a long running unit of work that has not completed processing when one of your DB2 systems terminates, it could take a long time to restart that DB2 system. This is because DB2 has to back out all the uncommitted changes for that unit of work.

Starting with DB2 Version 6, you can limit the amount of backward log processing that occurs at restart, thus preventing long running URs from delaying the availability of a DB2 system.

- ▶ The LIMIT BACKOUT ZPARM on installation panel DSNTIPL controls whether you want to postpone some backward log processing.
- ▶ The BACKOUT DURATION ZPARM on the same panel acts as a multiplier for your checkpoint interval.
- ▶ If the LIMIT BACKOUT ZPARM is set to YES or AUTO, then at restart DB2 limits the number of log records processed by any unit of work to the value of BACKOUT DURATION multiplied by the checkpoint frequency.
- ▶ If you specify AUTO for LIMIT BACKOUT, then DB2 postpones the recovery for units of work that meet the BACKOUT DURATION criteria. Then DB2 automatically recovers those units of work once the restart completes.
- ▶ If you specify YES for LIMIT BACKOUT, then DB2 will not attempt to recover those units of work that qualify for postponed recovery until you issue the RECOVER POSTPONED command.

Automatic restart

If you are running DB2 in a Parallel Sysplex, you can have the z/OS Automatic Restart Manager (ARM) automatically restart DB2 or IRLM after a failure. When DB2 or IRLM stops abnormally, z/OS does the following:

1. Determines whether z/OS failed too.
2. Determines where DB2 or IRLM should be restarted.
3. Restarts DB2 or IRLM.

You control how automatic restart works by using automatic restart policies available with z/OS Automatic Restart Manager (ARM).

Conditional restart recommendations

You only perform a conditional restart if there are no other options available that will successfully restart your DB2 system. If you are running DB2 in a Parallel Sysplex note that conditional restarts are member specific. There is no concept of a group-wide conditional restart. Therefore there is a high likelihood that you will have data integrity problems after issuing the conditional restart, unless you have system affinities that eliminate the possibility that data could be shared across members.

Failure of multiple members

Group restart requires scanning the logs of each member to rebuild the SCA or retained lock information. It is recommended that you have an alternate coupling facility on which these vital structures can be automatically rebuilt in the event of a coupling facility failure. The

automatic rebuild that occurs during a coupling facility failure does not require the log scans that group restart does.

During group restart, all restarting members update the SCA or lock structure from information contained in their logs. If you do not issue a START DB2 command for all members of the group, the started members perform group restart on behalf of the non-starting members by reading their logs. When the members are synchronized after forward log recovery, backward log recovery proceeds in parallel for the started members.

The phases of group restart are generally the same as in a non-data-sharing environment, with the addition of function for group restart. The phases of group restart vary based on whether the SCA, lock structure, or both are lost, and whether information is needed from the logs of inactive members.

The following is the summary of group restart phases when SCA structure is lost:

- ▶ Initialization
- ▶ CSR (rebuild SCA)
- ▶ Peer CSR (rebuild SCA)
- ▶ Forward-log recovery (rebuild locks)
- ▶ Backward-log recovery

The following is the summary of group restart phases when the lock structure is lost:

- ▶ Initialization
- ▶ CSR (reacquire page set P-locks)
- ▶ Peer CSR (rebuild page set P-locks)
- ▶ Forward-log recovery (rebuild locks) or Peer forward recovery (rebuild locks)
- ▶ Backward-log recovery Failure of connectivity between systems

Failure of CF

Although only one coupling facility is required to implement a data sharing environment, it is recommended to have more than one CF, thus eliminating the single point-of-failure.

To manage data sharing, DB2 uses three different kinds of CF structures:

- ▶ Lock structure — controls locking
- ▶ SCA structure — contains information about all DB2 members and exceptions
- ▶ GBP structure — maintains data consistency between members

It is also a good practice to have dedicated CF (at least one of them) running in another central processor complex (CPC) where the DB2 is running to avoid a double failure. Additionally, the Lock and SCA structures should not be placed on a CF which is within the same machine where a DB2 member of the data sharing group is executing. Should a CPC failure occur, the whole DB2 data sharing group will come down and a group restart will be required. With two CFs, it is possible to separate Lock and SCA structures from the others.

The following sections describe what happens to a DB2 CF structure, in the case of a CF failure.

SCA and Lock structures

The SCA and Lock structures can be rebuilt, if the structure is lost or fails. All of the data, both for the SCA and Lock structures are kept in memory and the rebuild is very quick. It takes only a few seconds. During the rebuild the structures are quiesced.

Group Buffer Pool Structure

Since DB2 Version 5, GBP structure rebuild is possible as well. But, the rebuilding process is different from SCA and Lock structure rebuild. For GBP structure rebuild, it is not guaranteed that the data to be rebuilt is still available in memory.

Depending on the situation, the changed pages may reside either in the GBP, the virtual BP, or even on DASD. In the event of a CF loss, structure error, or 100% connectivity loss, a process called damage assessment is initiated. If you are not using GBP duplexing, this means, that objects that are GBP-dependent are placed in Group Buffer Pool Recovery Pending (GRECP) status. From this status, they have to be recovered.

Since DB2 Version 5, this recovery is managed by DB2. Automatic recovery only works if AUTOREC(YES) is specified for the GBP. This is the default. The rebuild usually takes minutes to recover from the log, depending on the number of affected objects. During this time period, applications do not abend, they are only delayed.

Even though the rebuild function is an enhancement that helps to increase availability, there may be an impact on applications because of the a fore mentioned delay.

GBP duplexing

The purpose of GBP duplexing is to achieve continuous availability all the time, even in the unlikely event of a failure. Duplexing results in two allocations of the same group buffer pool, in other words, two instances of the GBP are allocated, each one in a different CF. The backup copy is called the secondary, or new structure. The instance used by DB2 is called the primary, or old structure.

If the primary GBP is lost, then DB2 automatically reverts to utilizing the secondary GBP. The secondary GBP structure is a hot standby copy of the primary GBP structure, and is ready to take over during problems associated with the primary GBP. When a GBP structure is switched from primary to secondary, it is almost transparent to the applications, creating a momentary pause. When duplexing is established for a certain GBP structure, the damage assessment process is not performed against DB2 objects using the GBP structure. So, the status of DB2 objects that reside in that structure do not change to GRECP.

Duplexing does not cause an overhead, as all the activities related to page registration, cross invalidation, read, and castout functions are performed by the primary GBP only. In the case of an asynchronous write to the secondary GBP, followed by a synchronous write to the primary GBP, DB2 always checks to ensure that the asynchronous write has completed successfully before the synchronous write is allowed to complete.

3.4.5 Systems management

Building on the scalability and availability features of the DB2 subsystem, the data sharing architecture provides significant configuration and systems management benefits. You can add members to your data sharing group and systems to your Parallel Sysplex, in a wide variety of configurations. If you expect gradual business growth, you can add small increments of capacity and be confident in the ability of the new configuration to scale. Or you can add larger increments, for example, after a merger or acquisition.

You can move DB2 members and transaction managers around within the images in your Parallel Sysplex as long as your z/OS parameter libraries are shared. Figure 3-14 illustrates that the resources of the Parallel Sysplex can be modified without affecting the data sharing, application, transaction manager, or network layers.

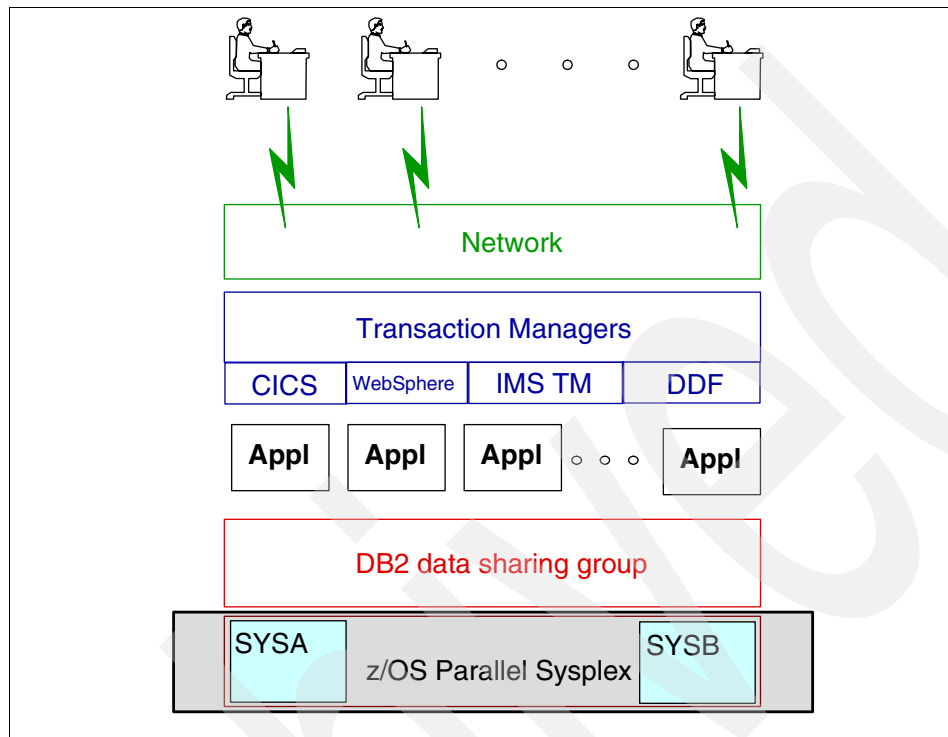


Figure 3-14 Flexible configurations for growth

Similarly, there is flexibility in the DB2 data sharing group layer, as illustrated in Figure 3-15 on page 45. DB2 members can be started on any z/OS image that has access to the coupling facility and can be moved to take advantage of changes in the Parallel Sysplex infrastructure. Changes in the data sharing group layer do not necessitate changes in the application, transaction manager, or network layers. Of course, if the DB2 changes offer performance or throughput advantages, it is beneficial to make adjustments in the application, transaction manager, and network layers to exploit these advantages.

Workload balancing techniques allow incremental changes in the Parallel Sysplex or data sharing layers to be exploited by the other layers to optimize the resources in your environment (Figure 3-15).

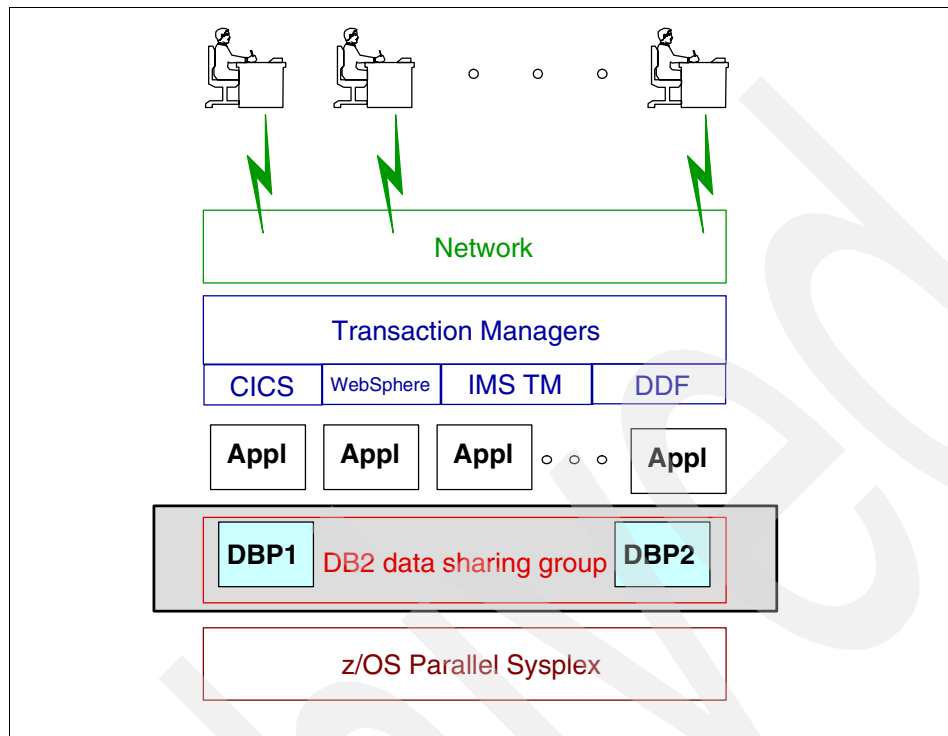


Figure 3-15 DB2 flexibility

Among the systems management advantages of data sharing is the ability to support operational and informational — or decision support — data and applications within the same data sharing group.

Often operational and decision support systems evolve separately, either due to capacity constraints or to concerns with managing the data to different service levels and with different access and security characteristics.

See Figure 3-16 for a diagram of data sharing implemented in this environment.

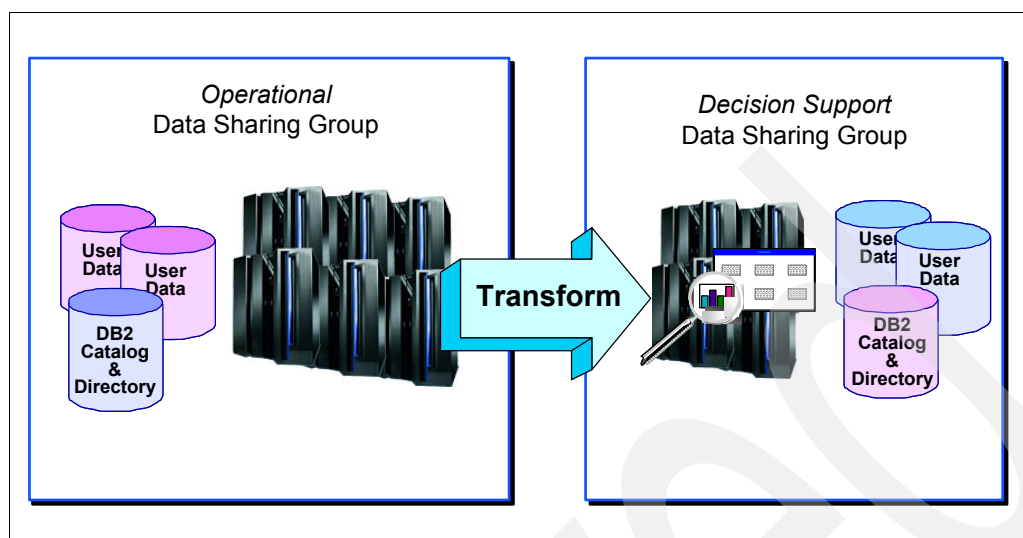


Figure 3-16 Separate operational and decision support groups

Data sharing allows you the option to bring the decision support data and applications back into the operational sphere. This allows greater flexibility for managing capacity within your environment and enables the occasional requirements to join operational and decision support rows to be supported within a single production environment. You no longer have to maintain separate systems and rely on distributed database access when combining operational and decision support data. See Figure 3-17.

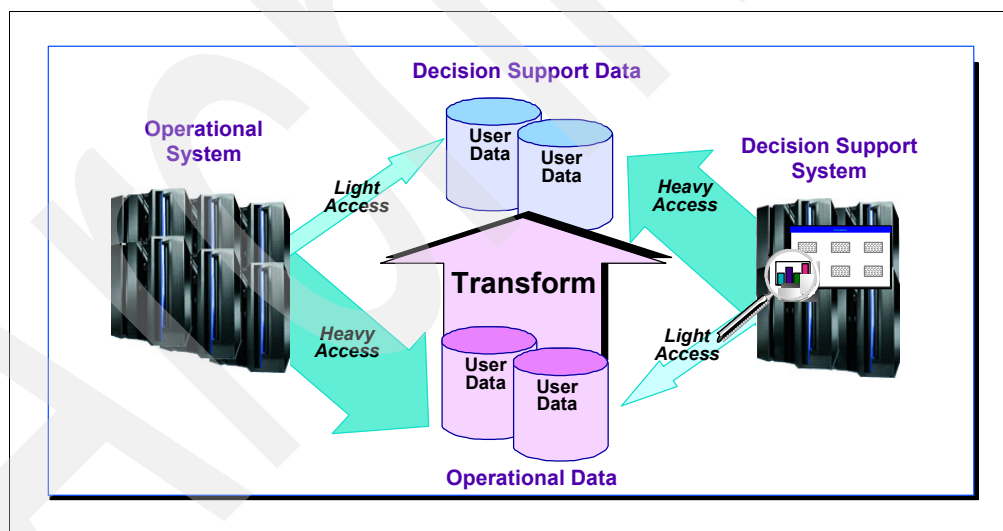


Figure 3-17 Combined operational and decision support data in one group

For further discussion about configuration flexibility and systems management, see Chapter 1 of the *DB2 for z/OS Version 8 Data Sharing: Planning and Administration*.

3.4.6 Application requirements

Access to a DB2 data sharing group can vary depending upon the needs of the application or the user that requires access. In this section we explore various methods for accessing DB2

data, and provide an overview of techniques you can use to redirect work to another member in the case of failure of one of the members. We discuss the following:

- ▶ Access directed to a single member
- ▶ Access directed to the group
- ▶ Using DB2 Connect™ to access remote data sharing groups
- ▶ Using group attach to connect to a data sharing group

Applications and users can communicate with a DB2 data sharing group by using one of two protocols: TCP/IP or SNA protocol. Each of these protocols supports multiple methods to access data-sharing groups.

- ▶ A TCP/IP requester can use one of the following three methods to access a data sharing group:
 - Group access
 - Member-specific access
 - Single-member access
- ▶ An SNA requester can use one of the following three methods to access a data sharing group:
 - Member-specific access
 - Group-generic access
 - Single-member access

Although the types of accesses sound the same for each protocol, there are differences in how they behave. Since many of the recent network computing enhancements to DB2 are only supported when using TCP/IP connections, we will only discuss the TCP/IP protocol for the remainder of this section.

Access directed to a single member

Single-member access is used to connect to a specific member of a data sharing group. All subsequent requests connect to the same member. If you have applications that must run on a specific member because of affinities to other resources such as VSAM files or other non-DB2 databases, you can use the single-member access method to connect to that member. Single-member access is just like accessing a DB2 subsystem that does not share data.

We recommend that you do not use single-member access because it does not allow for workload balancing across the data sharing group and does not allow for failover processing since the application cannot connect to another member if the specific member is unavailable.

Access directed to the group

Group access is supported by either dynamic virtual IP address (VIPA) network addressing or domain server (DNS) network addressing. The application requesting the connection uses the dynamic VIPA or DNS name to connect to any member of the group. That member then provides the requester with a list of available members. Subsequent requests can connect to any of the available members.

Dynamic VIPA network addressing gives you the ability to assign a specific, virtual IP address to a data sharing group and to each member of the group. This address is independent of any specific TCP/IP stack within the Parallel Sysplex. Even if a member is moved to another z/OS system, as in the case of a failure or maintenance, the member remains accessible and retains the same virtual IP address. Dynamic VIPA is supported in OS/390 V2R8 and above. Sysplex Distributor is the strategic IBM solution for connection workload balancing in a Parallel Sysplex and is built on dynamic VIPA. When a TCP connection request arrives, the

Sysplex Distributor routing stack consults Workload Manager (WLM) to find the relative available capacities on the nodes (OS/390 or z/OS) hosting the stack and application. The routing node also consults Service Policy Agent for network performance and defines policies that might affect the distribution decision.

With all available relevant information, including precisely which target stacks have server applications ready for work, the routing stack selects a target stack and forwards the request to that target stack for processing. The routing stack also remembers the full connection that connection is sent on to the same target stack for processing.

When the connection ends, the target stack notifies the routing stack that the connection ended, so the routing stack can discard the connection routing entry for the ended connection. The next connection from the same client is distributed independently of the earlier one.

We recommend that you use dynamic VIPA and Sysplex Distributor to configure your Parallel Sysplex for the best distribution of workload and to ensure continuous connectivity in the event of failure of one of the members.

Using DB2 Connect to access remote data-sharing groups

DB2 Connect Enterprise Edition connects LAN-based systems and their desktop applications to your company's mainframe and minicomputer host databases. DB2 Connect can be configured to access a single member or the entire group. Configuring DB2 Connect to access a single member requires disabling its sysplex support, which makes DB2 Connect dependent on that member being operational.

We recommend that you configure DB2 Connect for group access. DB2 Connect takes advantage of dynamic VIPA and Sysplex Distributor to balance the distributed workload across the members of the group. You need to configure DB2 Connect to access remote data-sharing groups. Use the Configuration Assistant to update the database directories that DB2 Connect uses to manage database connection information.

Using group attach to connect to a data-sharing group

When you submit a job on a z/OS system, you specify a DB2 subsystem name. That name can be either an actual subsystem name or a group attachment name. The group attachment name is defined on installation panel DSNTIPK and can be used by batch programs, the call attachment facility (CAF), the RRS attachment facility (RRSAF), IMS, CICS Transaction Server, and DB2 utilities. The group attachment name should not be the same as any of the member names in the group.

When one of the above listed requesters attempts to connect to the DB2 subsystem listed in the JCL, DB2 attempts to find a qualifying subsystem name.

- ▶ If that subsystem is found and it is started, DB2 attaches to that subsystem.
- ▶ If the subsystem is not found, then DB2 uses the group attachment name and connects to the first started DB2 member that is defined to this z/OS system, according to the order in which the subsystems were initialized at IPL time. DB2 will always connect to the first started subsystem in the list. There will be no attempt at load balancing.
- ▶ If the subsystem is found but it is not started and the connection request did not specify NOGROUP, then DB2 uses the group attachment name and connects to the first started DB2 member that is defined to this z/OS system, according to the order in which the subsystems were initialized at IPL time. DB2 will always connect to the first started subsystem in the list. There will be no attempt at load balancing.

Group attach recommendations

We recommend that you use the group attachment name for jobs that are not sensitive to a particular member. Specify the group attachment name for CICS or IMS applications, but note that CICS and IMS applications must be aware of the particular member to which they are attached. This is so they can resolve indoubt units of recovery in the event of a failure.

IMS Version 7, or later, allows IMS dependent regions to use the DB2 group attachment name to direct jobs to a data-sharing group. CICS Transaction Server Version 2.2 or later allows you to use the CICS RESYNCMEMBER=YES option to handle indoubt units of work.

Archived

Clustering on AIX

This chapter describes the basics of high-availability clustering on AIX and discusses a number of approaches that are commonly used. It then goes on to look in some details at a couple of commercial database clustering solutions, DB2 UDB DPF and Oracle RAC 10g.

In this chapter we discuss the following:

- ▶ Brief history of AIX clustering
- ▶ Terminology
- ▶ IBM HACMP™ clustering software
- ▶ DB2 for UDB on AIX with high availability and disaster recovery (HADR)
- ▶ Oracle 10g Grid RAC

4.1 History of clustering on AIX5L

AIX5L is the IBM UNIX operating system. AIX was introduced in 1986 and since then it has spanned multiple hardware and provided additional features and enhancements. The following section provides details for each of the major steps in its evolution.

Unlike z/OS, which is mostly found in commercial computing environments running online and running batch workloads, AIX is used across a much wider range of computing environments. As such, it is not surprising that there are a number of products available for AIX that address different clustering requirements.

High-availability clusters in AIX are traditionally of the active standby kind. An application running on one machine needs to be quickly restarted on a backup machine.

4.1.1 RS/6000

IBM RS/6000® servers brought 64-bit technology to the market with the introduction of the RS/6000 Enterprise Server Model S70 and AIX Version 4.3. Together, they provided the key elements of the 64-bit computing environment, both in hardware and software. This is the origin of the RS/6000 Enterprise Server. Along with the S70, the RS/6000 Enterprise Server Model S70 Advanced was introduced with the capability to be attached to the RS/6000 SP as an SP-attached server. The system is then ideally suited, for example, to handle large database transactions while allowing the other SP nodes to act as application servers.

Introduced in September of 1999, the RS/6000 Enterprise Server Model S80 uses up to 24 microprocessors built with the IBM innovative copper chip technology to meet the rigorous demands of mission-critical enterprise applications, such as Enterprise Resource Planning (ERP), which are rapidly evolving to Web serving. In addition to superior Web serving and ERP capabilities, the S80 excels with server consolidation, Supply Chain Management (SCM), Customer Relationship Management (CRM), On-line Transaction Processing (OLTP), and business intelligence (BI).

As business has grown, the number of RS/6000 Enterprise Servers sold to businesses increased. This promoted a demand for a solution that manages a number of RS/6000 Enterprise Servers, similar to the manageability that Parallel Systems Support Programs for AIX (PSSP) provides to RS/6000 SP systems. New to the PSSP 3.2 release, you can now manage RS/6000 Enterprise Servers like SP-attached servers in an RS/6000 SP system in a configuration called RS/6000 Clustered Enterprise Servers (CES). The CES system contains only RS/6000 Enterprise Servers managed from a single point-of-control, called a control workstation (CWS).

IBM RS/6000 Enterprise Servers

Designed for a broad range of applications serving medium to large businesses, IBM RS/6000 Enterprise Servers come in symmetric multiprocessor (SMP) models that are well suited for mission-critical commercial, large e-business, or ERP environments. The RS/6000 Enterprise Servers running the IBM AIX operating system provide complete 64-bit computing.

IBM RS/6000 SP-attached server

The joining of the IBM RS/6000 Enterprise Servers to the RS/6000 SP satisfies the need many SP environments have for large, powerful, and memory-rich processors for their database servers and SAP® R/3 applications and generally provides a single point-of-management for the entire system. The joining of these technologies was accomplished in the fall of 1998 with the introduction of the first SP-attached server. Since then, SP-attached servers have proven to be immensely popular and extremely stable for

commercial environments. The unique characteristics of each technology are still maintained and the differences, in some cases, are still accentuated.

IBM RS/6000 Clustered Enterprise Servers

In addition to the Enterprise Server's unsurpassed power, flexibility, and reliability, you can now utilize the PSSP's manageability. The CES system is a cluster of RS/6000 Enterprise Servers (S70, S70 Advanced, or S80), each running the PSSP software, connected to one control workstation (CWS) running PSSP 3.2 and connected to the SP Ethernet without any SP frame in the system. You are no longer required to have at least one SP frame and node in order to use the PSSP software. A maximum of sixteen Enterprise Servers are supported in one CES system. CES systems, however, are not supported with any type of SP Switch.

4.1.2 pSeries

POWER5™ is the IBM second generation dual-core microprocessor chip. The POWER5 chip features single and multi-threaded more than the previous machines at equivalent frequencies. In the following section, we describe some of the POWER5 features that are peculiar to the clustering environment.

POWER Hypervisor

The POWER™ Hypervisor is the foundation for virtualization on a System p5™ server. It enables the hardware to be divided into multiple partitions and ensures strong isolation between them.

Always active on POWER5-based servers, the POWER Hypervisor™ is responsible for dispatching the logical partition workload across the physical processors. The POWER Hypervisor also enforces partition security and can provide inter-partition communication that enables the Virtual I/O Server's virtual SCSI and virtual Ethernet function.

LPAR and shared-processor partitions

A Logical Partition (LPAR) is not constrained to physical processor boundaries, and it may be allocated processor resources from a shared processor pool. An LPAR that utilizes processor resources from the shared processor pool is known as a Micro-Partition LPAR. The percentage of a physical processor that is allocated is known as processor entitlement. Processor entitlement may range from ten percent of a physical processor up to the maximum installed processor capacity of the IBM System p5. Additional processor entitlement may be allocated in increments of one percent of a physical processor.

Dynamic reconfiguration

It is possible to dynamically move system resources, physical processors, virtual processors, memory, and I/O slots, between partitions without rebooting. This is known as dynamic reconfiguration (DR) or dynamic LPAR (DLPAR).

Virtual LAN

A function of the POWER Hypervisor, Virtual LAN allows secure communication between logical partitions without the need for a physical I/O adapter. The ability to securely share Ethernet bandwidth across multiple partitions increases hardware utilization.

Virtual I/O

Virtual I/O provides the capability for a single physical I/O adapter and disk to be used by multiple logical partitions of the same server, allowing consolidation of I/O resources and minimizing the number of I/O adapters required.

4.2 System p5 software

IBM offers a complete portfolio of software to improve the reliability, performance, and manageability of IT environments that include IBM System p5, eServer™ p5 and pSeries® or iSeries™ servers running AIX 5L™ or Linux and eServer xSeries® servers running Linux. The following lists the available features that are peculiar to support the clustering environment.

4.2.1 General Parallel File System

General Parallel File System™ (GPFS™) for AIX is the IBM clustered file system for AIX. GPFS is also available on Linux. On AIX, GPFS is implemented as a standard AIX Virtual File System, which means that applications using standard AIX VFS calls (such as JFS calls) run on top of GPFS without modifications.

GPFS supports direct attach concurrent access to a file system from multiple systems. There are many network file systems (like NFS) but these rely on a communication protocol (like TCP/IP) to transport the data blocks from a central server that owns the storage to clients that request the storage. Network file systems have performance and availability limitations.

GPFS allows multiple machines running AIX to directly attach to file systems on shared storage like a SAN. GPFS services on each node coordinate file access and manage file serialization to ensure data integrity. GPFS allows multiple systems to update the same file concurrently as long as two systems do not need to update the same data block within the file. GPFS uses byte range lock management to achieve this.

4.2.2 Cluster Service Manager

Cluster Systems Management (CSM) for AIX and Linux is designed for simple, low-cost management of distributed and clustered IBM pSeries and xSeries servers in technical and commercial computing environments. CSM, included with the IBM Cluster 1600 and Cluster 1350™, dramatically simplifies administration of a cluster by providing management from a single point-of-control. CSM is available for managing homogeneous clusters of xSeries servers running Linux, pSeries servers running AIX, or heterogeneous clusters which include both.

CSM is a systems management tool that helps manage the efficiency of the configuration, reducing the cost and the complexity of the management through the following capabilities:

- ▶ CSM improves administrator efficiency by controlling multiple, heterogeneous machines from a single point, enabling fast responses and consistent policies, and by updating and monitoring by a small staff.
- ▶ CSM provides monitoring with automated responses and notification to ensure problems are recognized and addressed quickly with minimal distraction.
- ▶ CSM simplifies management with predefined functions for common tasks and predefined monitors for key system metrics. Complex functions and monitors can be defined for easy, single-command or point-and-click use.
- ▶ Management by group eliminates both the need to address each machine individually and the risk of accidental oversight or omission. CSM can even build groups automatically based on machine characteristics.
- ▶ CSM improves security and efficiency by giving administrators access to the commands they need without having to give every administrator high authorization. Security configuration is performed automatically.

In addition to providing all the key functions for administration and maintenance of typical distributed systems, CSM is designed to deliver the parallel execution required to manage clustered computing environments effectively. The managed nodes in the cluster may be pSeries or iSeries machines running AIX or Linux, xSeries machines running Linux, or a combination of these platforms and operating systems. CSM provides the following system management capabilities:

- ▶ Installation and setup commands to create and configure the cluster easily.
- ▶ Security configuration across the cluster is performed automatically, eliminating the need to manually distribute encryption keys and so on.
- ▶ Create node groups within the cluster and manage machines by group. Node groups can be defined manually or dynamically (for example, if dynamic node groups are defined for Linux nodes and AIX nodes, then new nodes added to the cluster are automatically placed in the appropriate group and configured and managed accordingly).
- ▶ Remote command execution across multiple nodes (or node groups) in the cluster. CSM tracks and displays the status of each command on each machine.
- ▶ A configuration file manager to distribute and synchronize files across nodes or node groups in the cluster.
- ▶ Comprehensive monitoring of system parameters and status with event display and automated responses (for example, a file system filling up can send an event to the CSM console and automatically notify an administrator and trigger a script to remove all temporary files).
- ▶ Software diagnostic tools minimize down time by systematically analyzing software components and servers to find the root cause of problems quickly.
- ▶ Scalable, remote hardware control and console access allows administrators to manage systems from a remote location. This enables centralized management of systems in different locations, increasing administrative efficiency.
- ▶ Fully scriptable command line interface supports user-defined commands to automate complex tasks.
- ▶ Administrators can be authorized to perform specific tasks that exceed their overall authorization level, enabling them to do their jobs effectively without compromising security.
- ▶ Web-based interface provides control and management of a CSM cluster including the following:
 - Updating and removing nodes from the cluster
 - Creating and deleting node groups
 - Distributed command execution including user-defined commands
 - Visual monitoring capabilities
 - Hardware control capabilities
- ▶ CSM is flexible and fully customizable to fit into and augment your existing administrative processes. Users can add their own power methods, console methods, MAC methods, post-install scripts, CFM pre/post scripts, sensors, conditions, responses and probes.

4.2.3 High-Availability Cluster Multiprocessing

IBM High Availability Cluster Multiprocessing for AIX 5L V5.3 (HACMP V5.3) is designed to provide high availability for critical business applications. For over a decade, HACMP has provided reliable monitoring, failure detection, and automated recovery of business application environments to backup resources. And the HACMP/XD (Extended Distance) component extends failover to backup resources at remote sites to ensure reliable service delivery even if a disaster disables the primary site. HACMP/XD provides remote failover for Enterprise Storage Server® (ESS) and SAN Volume Controller Metro-Mirror (formerly Peer-to-Peer Remote Copy PPRC) peers and unlimited distance failover for IP-connected peers using proven AIX 5L mirroring technology. When HACMP detects a failure, it can quickly and automatically take action to restore the application, restarting it on a redundant system if necessary. If the backup system is at a remote site, HACMP can also switch to a

mirror (replicated) copy of the application data automatically and reverse the mirroring process.

HACMP can be configured to react to hundreds of system events including problems that are not severe enough to interrupt proper system operation (such as process failure or exhaustion of system resources). HACMP monitors, detects, and reacts to such conditions, maintaining service availability during random, unexpected software problems. An HACMP cluster can contain up to 32 servers, allowing environments to scale-out growth with high reliability.

HACMP can also virtually eliminate planned outages by transferring users, applications, and data to backup systems during scheduled maintenance. HACMP clusters can be configured to meet complex and varied application availability and recovery needs.

4.3 DB2 UDB clustering implementation on AIX

DB2 Universal Database for Linux, Windows, and UNIX (DB2 UDB for LUW) is the IBM relational database product for open systems and Windows platforms. DB2 UDB comes in various flavors from Express Edition to Enterprise Server Edition. We discuss DB2 UDB Enterprise Server Edition because that is the version that provides the features required for a clustered environment.

With the DB2 Universal Database^(TM) (DB2 UDB) Enterprise Server Edition (ESE), installation can create a partitioned instance using the Database Partitioning Feature (DPF) to improve performance. A database partition is part of a database that consists of its own data, indexes, configuration files, and transaction logs. A partitioned database is a database with two or more partitions. Tables can then be located in one or more database partitions. Processors associated with each database partition are used to satisfy table requests. Data retrieval and update requests are decomposed automatically into sub-requests and executed in parallel among the applicable database partitions.

The most common approach we see in the AIX customers running online and batch workloads is DB2 running on SMP machines with an active standby cluster. Traditionally this model used HACMP to restart the DB2 instance on the standby node in the event of failure. With DB2 UDB V8.2, IBM introduced a new hot standby feature, high availability and disaster recovery (HADR), that can be used in local clusters or as part of a disaster recovery solution. HADR is a database replication feature that provides a high-availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the standby. In the event of a partial or complete site failure, the standby database can quickly take over for the primary database.

The automatic client reroute feature can be used with HADR to enable client applications to recover from a loss of communication with the server and to continue working with minimal interruption. You can use automatic client reroute with HADR to have client applications connect to the new primary database after a takeover operation.

A partial site failure can be caused by hardware, network, or software (DB2 or operating system) failure. Without HADR, the database management system (DBMS) server or the machine where the database resides has to be rebooted. The length of time it takes to restart the database and the machine where it resides is unpredictable. It can take several minutes before the database is brought back to a consistent state and made available. With HADR, the standby database can take over in seconds. Further, you can redirect the clients that were using the old primary database to the standby database (new primary database) by using automatic client reroute or retry logic in the application.

A complete site failure can occur when a disaster, such as a fire, causes the entire site to be destroyed. Since HADR uses TCP/IP for communication between the primary and standby databases, the databases can be situated in different locations. If a disaster occurs at the primary site, data availability is maintained by having the remote standby database take over as the primary database with full DB2 functionality. After a takeover operation occurs, you can bring the original primary database back up and return it to its status of primary database. This is known as failback.

With HADR, you can choose the level of protection you want from potential loss of data by specifying one of three synchronization modes: synchronous, near synchronous, or asynchronous.

HADR allows the standby database to take over as the primary database with full DB2 functionality. It is also possible for the original, primary database to be brought back up and returned to its status of primary database.

When a failure occurs on the primary database, you can initiate a takeover operation on the standby database, which then becomes the new primary. Since the standby database is already online, failover can be accomplished very quickly resulting in minimal down time.

Once the failed old primary is repaired, it can rejoin the HADR pair as a standby database if the two copies of the database can be made consistent. After the original primary database is reintegrated into the HADR pair as the standby database, a failback operation can be performed so that the original primary database is once again the primary database.

4.3.1 DB2 active-standby cluster using HADR

The following are main considerations for this type of solution:

Storage model

A DB2 configuration with one active node and one standby node implements a shared-nothing storage model. All requests are made to the active node. The standby node runs a standby copy of DB2 and receives log updates from the active node, which it applies to its copy of the database. Shared disk is not required because the standby copy of the database has its own complete copy of the database on its own disk. DB2 HADR can be run in synchronous, near-synchronous, or asynchronous mode.

Lock management

In this mode all lock management is done on the active copy of the database. There are no additional issues above those that exist in a non clustered environment.

Cache management

In this mode there is no distributed cache and no requirement for a global cache manager. DB2 uses its normal buffer management on the active and standby nodes. There are issues however that need to be understood that relate to the HADR mode. HADR can operate in synchronous, near synchronous, or asynchronous modes.

DB2 HADR in synchronous mode

In synchronous mode, any log update committed to the logs on the primary instance AND must be transmitted to the standby instance and be applied to its logs and acknowledged before the primary instance returns to the application. This mode provides the greatest protection against transaction loss, and using it results in the longest transaction response time among the three modes. In this mode, log writes are considered successful only when logs are written to log files on the primary database and when the primary database receives

acknowledgement from the standby database that the logs have also been written to log files on the standby database. The log data is guaranteed to be stored at both sites.

If the standby database crashes before it can replay the log records, the next time it starts it can retrieve and replay them from its local log files. If the primary database fails, a failover to the standby database guarantees that any transaction that was committed on the primary database was also committed on the standby database. After the failover operation, when the client reconnects to the new primary database, there may be transactions committed on the new primary database that were never reported as committed to the original primary. This occurs when the primary database fails before it processes an acknowledgement message from the standby database. Client applications should consider querying the database to determine whether any such transactions exist.

If the primary database loses its connection to the standby database, the databases are no longer considered to be in peer state and transactions will not be held back waiting for acknowledgement from the standby database. If the failover operation is performed when the databases are disconnected, there is no guarantee that all of the transactions committed on the primary database will appear on the standby database.

If the primary database fails when the databases are in peer state, it can rejoin the HADR pair as a standby database after a failover operation. Because a transaction is not considered to be committed until the primary database receives acknowledgement from the standby database that the logs have also been written to log files on the standby database, the log sequence on the primary is the same as the log sequence on the standby database. The original primary database (now a standby database) just needs to catch up by replaying the new log records generated on the new primary database since the failover operation.

If the primary database is not in peer state when it fails, its log sequence might be different from the log sequence on the standby database. If a failover operation has to be performed, the log sequence on the primary and standby databases might be different because the standby database starts its own log sequence after the failover. Because some operations cannot be undone (for example, dropping a table), it is not possible to revert the primary database to the point in time when the new log sequence was created. If the log sequences are different, an error message is returned when you issue the START HADR command with the AS STANDBY option on the original primary. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

DB2 HADR in near-synchronous mode

While this mode has a shorter transaction response time than synchronous mode, it also provides slightly less protection against transaction loss.

In this mode, log writes are considered successful only when the log records are written to the log files on the primary database and when the primary database receives acknowledgement from the standby system that the logs were written to the main memory on the standby system. Loss of data occurs only if both sites fail simultaneously and if the target site has not transferred, to nonvolatile storage, all of the log data that it received.

If the standby database crashes before it can copy the log records from memory to disk, the log records are lost on the standby database. Usually, the standby database can get the missing log records from the primary database when the standby database restarts. However, if a failure on the primary database or the network makes retrieval impossible and a failover is required, the log records never appear on the standby database, and transactions associated with these log records never appear on the standby database.

If transactions are lost, the new primary database is not identical to the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up-to-date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database cannot rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records (because both the primary and standby databases failed), the log sequences on the primary and standby databases are different and attempts to restart the original primary database as a standby database without first performing a restore operation will fail. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

DB2 HADR in asynchronous mode

This mode has the highest chance of transaction loss if the primary system fails. It also has the shortest transaction response time among the three modes.

In this mode, log writes are considered successful only when the log records are written to the log files on the primary database and are delivered to the TCP layer of the primary system's host machine. Because the primary system does not wait for acknowledgement from the standby system, transactions might be considered committed when they are still on their way to the standby.

A failure on the primary database host machine, in the network, or on the standby database can cause log files in transit to be lost. If the primary database is available, the missing log files can be resent to the standby database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log files, both the log files and the associated transactions will never reach the standby database. Permanent loss of transactions is caused by lost log files and a failure on the primary database.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up-to-date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database cannot rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater possibility of log records being lost if a failover occurs in asynchronous mode, there is also a greater possibility that the primary database cannot rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

Application requirements

There are no application requirements over and above those that apply in a non clustered environment. DB2 client applications can use the DB2 automatic client reroute feature to connect to the standby DB2 instance should the primary fail. No application changes are required to exploit this feature.

Transactional requirements - two phase commit across members

In this model there are no issues because all connections are made to one server.

Performance

The single-active node model is undoubtedly the most efficient model from a performance point-of-view. It avoids all the performance issues of global lock management and global cache management that exist with the shared all models. A large SMP multiprocessor is always more efficient than multiple nodes with smaller processors in an online environment.

Availability - restart and recovery

Any failure of the active node, whether its is connectivity failure, operating system failure, or DB2 instance failure results in DB2 being unavailable to client applications until the standby DB2 instance takes over. The standby DB2 does not detect failures in the active DB2. It needs some automation product to provide this function. In the AIX environment, HACMP can provide this function.

The main difference between the DB2 HADR approach and the conventional active standby cluster approach is that there is already a DB2 instance started. The cluster service does not need to vary over storage and restart a DB2 instance that will need to do log recovery. This means that failover can be very fast. IBM quotes test scenarios where failover occurs in just 10 seconds. In this case, the cluster manager needs to be able to detect the failure on the primary node.

If HADR is running in synchronous mode (which is feasible in a local cluster connected by high-speed interconnects) then there is no lost of committed transactions. If HADR is running asynchronously then there will be some data loss—the amount depends on the speed of the link between the two nodes and the type of failure scenario. In either case in-flight transactions are lost and client applications will need to reissue the transactions. Whether this is visible to users depends on the logic in the client application. A client application can be written to automatically retry transactions based on the type of error received.

Systems management

In an active-standby configuration with HACMP, there is the need to have some process in place to ensure that the configurations are synchronized. Non logged DB2 changes are not replicated to the standby database, as well as any changes to the DB2 configuration files on the active instance are not automatically applied on the standby instance. This requires strict change management processes.

DB2 HADR requires that the file systems, or raw storage being used, must be configured identically on both primary and standby instances. DB2 HADR replicates DDL, so if you issue a DEFINE TABLESPACE command on the primary database it is replicated to the standby database. To make this work, DB2 on the standby instance must have the same device names or path names available as on the primary instance.

DB2 or operating system upgrade methodology

DB2 HADR requires that both active and standby nodes are running on the same operating system and machine architecture. The level of DB2 must be identical except for a short period allowed during a rolling upgrade. IBM recommends to keep also the operating system levels in sync.

4.4 Oracle Database 10g RAC implementation on AIX

Oracle Database 10g is the current version of the database product. Oracle Database 10g can be deployed on multiple platforms, including IBM AIX, Linux, Microsoft® Windows, and Sun™ Solaris™.

Real Application Cluster (RAC) was introduced in 2001 with the release of Oracle 9i and replaced Oracle Parallel Server (OPS). OPS was first used on clusters of DEC VAX machines running the VMS operating system in 1989. With RAC it is possible to implement a shared disk, shared cache, active-active cluster.

Oracle Database 10g RAC is shipped with all the cluster software necessary, and there is no need for additional cluster software on the operating system level. No special designed hardware is required. RAC can be implemented on low-priced commodity hardware components.

Figure 4-1 is a logical illustration of an Oracle RAC configuration.

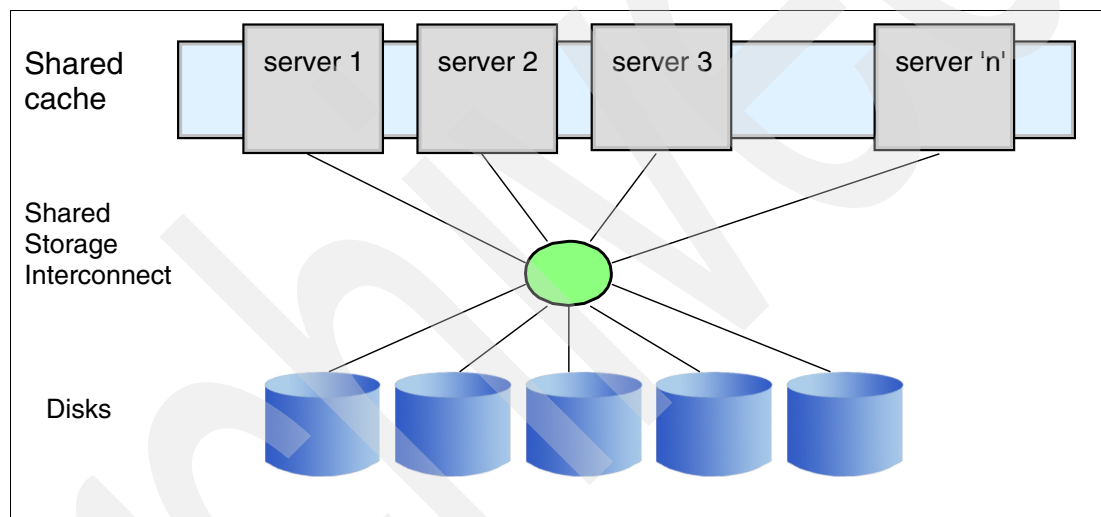


Figure 4-1 Logical view of Oracle RAC

RAC requires shared disk for all Oracle components except the Oracle binaries. On AIX this can be implemented using Raw disk devices or GPFS. HACMP is no longer required in an Oracle 10g RAC configuration. Oracle Cluster Services (OCS) introduced with 10g provides the cluster functionality.

4.4.1 Storage model

Oracle RAC implements a shared cache model. All nodes in an Oracle RAC cluster share database files and present a single-system view of the database to client applications. Oracle provides the global lock management and global cache management. Oracle refers to this functionality as Cache Fusion.

We will look at the global lock and cache management in some detail.

Lock management

Oracle uses a combination of multiversion concurrency control (MVCC) and row level locking for managing serialization to database resources. This is quite unlike the centralized lock

management model used by DB2. If you are unfamiliar with MVCC, read the Oracle Database concepts guide to familiarize yourself with this model.

MVCC does not serialize access to a resource (a row for example) using a centralized lock manager. MVCC uses multiple versions of a piece of data to provide a requestor with a consistent view of the data. The MVCC mantra is “readers do not block writers and writers do not block readers”. A detailed discussion is beyond the scope of this publication; therefore, anyone who wants to immerse themselves in the lock managers versus multiversion control and pessimistic concurrency versus optimistic concurrency debates can find many sources of information on the Web.

Oracle Database achieves MVCC by storing rollback segments that can be applied to produce a consistent read (CR) version of a data block. Versioning is based on the System change Number (SCN) at the start of a SQL statement or transaction. For example if a transaction issues a query it gets a version of all the data that was committed before the query began. Any other data updates or inserts that are part of other uncommitted transactions or were committed after the query began are not displayed. Oracle’s default setting is query level isolation.

In Oracle the locking information for a row is held as part of that row. Read only queries do not hold a lock, only updates or read for updates place a lock on a row.

RAC locking

In a RAC environment, Oracle must also provide a further locking mechanism because data blocks may be distributed amongst several Oracle instances. A data block may contain several rows and while each row contains lock information that is valid for each row, there needs to be a way to identify read/write interest at a block level between instances.

Note: In System z literature, this is the equivalent issue that DB2 on z/OS has when it uses P-Locks to register read/write interest.

GCS manages this by assigning a resource mode and resource role. The resource mode can be Null, shared, or exclusive. The resource role can be Local or Global.

Cache management

Before RAC and a facility called Cache Fusion, when multiple Oracle instances wanted to update data that was held in an other node’s cache, that cache had to be flushed to disk before the data was updated. This operation is known as ping, and is a bottleneck in terms of performance since it required I/O operations. This is eliminated with RAC because of Cache Fusion. Cache Fusion reduces disk activity by making the data stored in every cache on every node available to every other node. If a node needs data stored in another node’s cache, the data is passed over the network, not via disk. This has a big performance gain. RAC implements the shared-cache model, in a ring topology.

4.4.2 Application requirements

There are no application requirements over and above those that apply in a non clustered environment.

Transactional requirements - two phase commit across members

Oracle maintains data consistency in a multiuser environment by using a multiversion consistency model and various types of locks and transactions. Data concurrency and data integrity are provided by using the locking mechanism. Since the locking mechanism is

related to the transaction control, some attention is required during the application definition and then Oracle can manage the locking function.

4.4.3 Performance

Oracle claims that the overall system overhead is between ten and fifteen percent.

4.4.4 Availability - restart and recovery

By having multiple instances running on different machines redundancy is provided. Every Oracle instance has full access to the shared database. Even if one Oracle instance is taken out, the database will still be available through the other nodes. It does not matter which of the instances clients connect to, because they will be able to access the same data simultaneously.

Transactions that were not completed are rolled back and have to be submitted once more. The recovery procedures do not require manual intervention. There is no need for making restart scripts, the way it is done in traditional UNIX clusters because RAC takes care of all recovery in a controlled and efficient way.

When a node failure is detected, the Global Resource Directory (GRD) is frozen. This implies that the entire database is unavailable. One of the still healthy nodes takes role as the recovering instance and initiates analyzing of the dead node's log. The log is accessible because it is stored on a shared storage device. GRD is rebuilt and uncommitted transactions are either rolled back or committed. If other nodes have a duplicate of some of the data blocks owned by the failed node, these are copied to the new mastering node. When all necessary data is acquired, the GRD is unfrozen. All data blocks, except those that were mastered by the failing node, can be accessed again. This implies that the database is partially available. Recovery continues until all data blocks becomes available.

If a node fails, the recovery procedure does not include automatic restart.

4.4.5 Systems management

The RAC clustering implementation has the objective to present itself as a single system to the user, providing transparency to the data access on all disk devices and remote cache. This single system view is also used from a management point-of-view for installation, configuration, and monitoring tasks.

Archived

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

For information about ordering these publications, see “How to get IBM Redbooks” on page 65. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IBM System z Strengths and Values*, SG24-7333
- ▶ *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322

Other publications

These publications are also relevant as further information sources:

- ▶ *z/OS V1R8.0 MVS Setting Up a Sysplex*, SA22-7625
- ▶ *DB2 UDB for z/OS V8 Data Sharing: Planning and Administration*, SC18-7417-04

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

A

Active-Active 7
Active-Standby 7
application requirement 46, 59
Automatic Restart Management (ARM) 23

B

BACKOUT DURATION
 criterion 41
 ZPARM 41
Buffer Pool 28
business intelligence (BI) 52

C

cache management 25, 28, 57, 61
cache model 10
call attachment facility (CAF) 48
central processor complex (CPC) 19
CES system 52
channel-to-channel (CTC) 19
client application 56
cluster architecture 8
cluster availability 14
cluster member 2, 5–6
cluster performance 12–13
Cluster Systems Management (CSM) 54
cluster type 2
Clustered Enterprise Servers (CES) 52
configurations for growth 44
consistent read (CR) 62
couple data 19
Coupling Facility
 Link 22
 standard sysplex 22
Coupling Facility (CF) 17
Coupling Facility Control Code 19
Customer Relationship Management (CRM) 52

D

data block 7, 54
 invalidation messages 12
 local copy 13
 remote copy 13
data integrity 7, 9, 20, 54
 control mechanism 9
 multiple systems 20
database management system (DBMS) 9, 56
Database Partitioning Feature (DPF) 51
DB2 instance 9, 56
DB2 member 26
DB2 object 30
 inter-DB2 read/write interest 30

DB2 subsystem 26
 availability features 43
DB2 UDB
 DPF 56
 Enterprise Server Edition 56
 V8.2 IBM 56
decision support 45
Dynamic reconfiguration (DR) 53
dynamic VIPA 47

E

Enterprise Server Edition (ESE) 56
Enterprise Storage Server (ESS) 55

F

failover 7, 55
failover operation 58
 original primary database 59
 standby database 58
function shipping 8, 18

G

GBP cache
 structure 29, 31
 structure space 37
GBP structure 30
General Parallel File System (GPFS) 54
Global Resource Directory (GRD) 63
Global Resource Serialization (GRS) 18
Group Buffer Pool (GBP) 26

H

HADR pair 57
high availability (HA) 2
high performance (HP) 2

I

IBM offers (I/O) 53
IBM RS/6000
 Enterprise Server 52
 SP-attached server 52
IMS application 49
IMS OSAM 21
IRLM member 27

L

local buffer 30
local cache 12, 34
 buffer 12, 31
 buffer copies 38
 vector 31, 34

- vector table 31
- local IRLM 27
 - certain locks 27
- lock avoidance 27
- lock manager 8, 21, 62
- lock structure 21
 - main users 21
- log record 15, 22, 41, 58–59
- Logical Partition
 - secure communication 53
- Logical Partition (LP) 19, 53

M

- multiple member 7, 27, 41
- Multiple Systems Coupling (MSC) 18
- multisystem data 19

N

- non-data-sharing environment 39

O

- On-line Transaction Processing (OLTP) 52
- Oracle Cluster Services (OCS) 61
- Oracle Database
 - 10g 61
 - 10g RAC 61
 - 10g RAC implementation 61
 - concepts guide 62
- Oracle Parallel Server (OPS) 61
- Oracle RAC
 - 10g 51
 - cluster share database file 61
 - configuration 61
 - grid solution 6
- original primary database 57

P

- Parallel Sysplex 17
 - architecture 20
 - basic concepts 17
 - component 20, 23
 - concept 17
 - configuration 24
 - context 6
 - core functions 23
 - datasharing 25
 - different elements 23
 - environment 39
 - exploitation 25
 - incremental changes 45
 - infrastructure 44
 - note 41
 - shared workload 21
 - technology 19
 - z/OS image 24
 - z/OS images 19
- primary database 56, 58–59
 - backup image 58

- log files 58
- missing log records 58
- TABLESPACE command 60
- primary GBP 43
 - only.in 43
 - structure 43
- PSSP software 53

R

- Real Application Cluster (RAC) 61
- Redbooks Web site 65
 - Contact us viii
- remote data 47–48
- RS/6000 Enterprise Server
 - Model S70 52
 - Model S70 Advanced 52
 - Model S80 52

S

- Sample scenario 30
- SCA structure 42
- secondary GBP 43
- shared cache model
 - locking scheme 13
- Single point
 - whole cluster 14
- single point 2
 - multiple, heterogeneous machines 54
- SP frame 53
 - SP Ethernet 53
- SP-attached server 52
- standby database 56
 - acknowledgement message 58
 - log sequence 58
 - takeover operation 57
- storage model 8, 26, 57
- Supply Chain Management (SCM) 52
- Support Programs (SP) 52
- Sysplex Failure Management (SFM) 23
- System change Number (SCN) 62
- System z
 - clustering 17
 - customer 21
 - Early clustering support 18
 - early subsystems 25
 - experience 1
 - hardware 21, 24
 - machine 22
 - microcode 22
 - Parallel Sysplex concept 1
 - platform 18
 - processor 21, 24
 - relational database solution 25
- systems management 43

U

- uncommitted read (UR) 40

V

Virtual File System (VFS) 54

Z

z/OS component 20

z/OS image 19

z/OS system 19

Archived

Archived



Clustering Solutions Overview: Parallel Sysplex and Other



Clustering concepts

Understanding Parallel Sysplex clustering

Comparing the terminology

This IBM Redpaper will help you do an informed comparison of Parallel Sysplex® against clustering implementations on other platforms. We start by describing the basic components of Parallel Sysplex data sharing, using DB2® as a sample implementation of the capabilities provided by Parallel Sysplex. Using this as a base, we then describe how each of the components are implemented on other platforms.

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