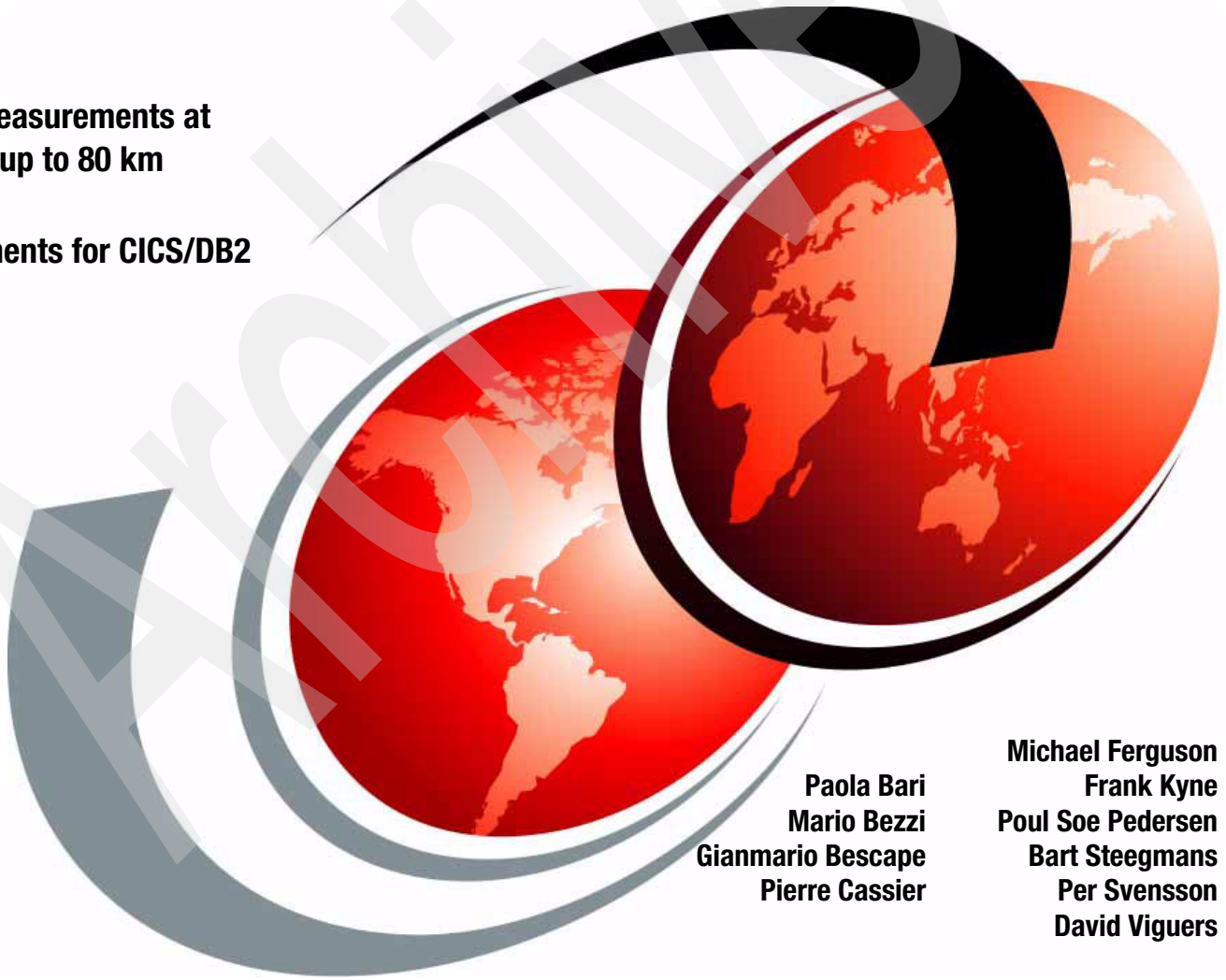
IBM

Considerations for Multisite Sysplex Data Sharing

- Understand how distance impacts data sharing
- Sample measurements at distances up to 80 km
- Measurements for CICS/DB2



Paola Bari
Mario Bezzi
Gianmario Bescape
Pierre Cassier
Michael Ferguson
Frank Kyne
Poul Soe Pedersen
Bart Steegmans
Per Svensson
David Viguers

Redbooks

ibm.com/redbooks

International Technical Support Organization

Considerations for Multisite Sysplex Data Sharing

August 2008

SG24-7263-00

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

First Edition (August 2008)

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

Contents

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

Trademarks

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

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

CICSPlex®
CICS®
DB2®
DS8000™
ESCON®
FICON®
GDPS®
HyperSwap™
IBM®
IMS™
MVS™
MVS/ESA™
OS/390®
Parallel Sysplex®
PR/SM™
Redbooks®
Redbooks (logo) ®
RMF™
S/360™
System z™
System z9®
Tivoli®
VTAM®
WebSphere®
z/Architecture®
z/OS®
z9™

Disk Magic, IntelliMagic, and the IntelliMagic logo are trademarks of IntelliMagic BV 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® Redbooks® publication provides a broad understanding of the considerations to keep in mind when you implement IBM z/OS® sysplex data sharing across geographically distributed sites. This is a relatively new technology with many complexities, so it can be challenging to understand and accurately predict how its implementation might affect the performance and capacity of your environment.

To promote that understanding, we measured and documented a number of workloads over various distances. In this book, we discuss in detail how the performance changed in each case, and *why* we experienced the results that we did. While it is not possible to accurately model the impact of distance on data sharing, by providing a discussion of our results you will be better prepared to undertake a similar project yourself.

The team that wrote this book

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

Paola Bari is an Advisory Programmer at the International Technical Support Organization, Poughkeepsie Center. Paola has 23 years of experience as a systems programmer in OS/390®, z/OS, and Parallel Sysplex®, including several years of experience in WebSphere® MQ and WebSphere Application Server.

Mario Bezzi is a Consulting I/T Specialist with IBM Italy. He also works half-time as a Project Leader for the International Technical Support Organization, Poughkeepsie Center. He has 24 years of experience in MVS™ system programming, both as a customer and in various technical support positions. Mario's areas of expertise include system tuning; Parallel Sysplex configuration design, management, tuning and programming; System Logger, RRS, SMF, Assembler and C language programming; and system and DASD hardware technology.

Gianmario Bescape is an Advisory Software Support Specialist with IBM GTS in Italy. He has worked with the IMS™ Technical Support group in Milan for about 20 years, and is currently a member of the EMEA IMS second level support team. Gianmario's experience includes problem resolution support and the implementation of IMS Shared Queue environments for large customers.

Pierre Cassier is a Certified IT Specialist working for IBM France in Integrated Technology Services. He has more than 21 years of systems programming experience in mainframe environments on MVS, OS/390, and z/OS platforms. His areas of expertise include z/OS, Parallel Sysplex, WLM, and performance tuning. Pierre teaches WLM and Performance Management courses.

Michael Ferguson is a Senior IT specialist in the IBM Support center in Australia. He has 21 years of experience in the z/OS software field, including Tivoli® Workload Scheduler for z/OS. His areas of expertise include Tivoli Workload Scheduler for z/OS, Parallel Sysplex, and z/OS. Michael teaches and provides services for Tivoli Workload Scheduler for z/OS and Parallel Sysplex in the Asia Pacific region.

Frank Kyne is an Executive IT Specialist at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas

of Parallel Sysplex and high availability. Before joining the ITSO 10 years ago, Frank worked in IBM Global Services in Ireland as an MVS System Programmer.

Poul Soe Pedersen has worked as a DB2® for z/OS system programmer for IBM for three years. He has 30 years of mainframe computer experience. Paul has been a system programmer for DB2 for the last 18 years, working with DB2 data sharing, Parallel Sysplex, and GDPS® systems.

Bart Steegmans is a Senior DB2 Product Support Specialist from IBM Belgium, currently working remotely for the Silicon Valley Lab in San Jose, providing technical support for DB2 for z/OS performance problems. From 2001 to 2004, he was on assignment as a Project Leader with ITSO, San Jose Center, specializing in Data Management for z/OS. Bart has more than 19 years of experience in DB2. Before joining IBM in 1997, he worked as a DB2 system administrator at a banking and insurance group. His areas of expertise include DB2 performance, database administration, and backup and recovery.

Per Svensson is a DB2 Database Administrator with IBM in Denmark. Prior to joining IBM three years ago, Per worked for Danske Bank. He has 22 years of experience in application development and database administration. For the last eight years, Per has specialized in large system database performance and DB2 sysplex data sharing.

David Viguers is a member of the IBM IMS development team at Silicon Valley Lab, working with performance, test, development, and customer support. He also develops and delivers education on IMS Parallel Sysplex. For more than 35 years, David has worked with IMS in various functions and organizations.

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

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Bob Haimowitz
International Technical Support Organization, Poughkeepsie Center

Stephen Anania
IBM USA

Chris Baker
IBM UK

Andrew W Bullinger
IBM USA

Andrea Harris
IBM USA

Claudia Ho
IBM USA

Jeff Josten
IBM USA

Jerry Kenyon
IBM USA

Lucy Krantz
IBM USA

Terri Menendez
IBM USA

Bruce Naylor
IBM USA

Carsten Rasmussen
IBM Denmark

Frank Ricchio
IBM USA

Judy Ruby-Brown
IBM USA

Akira Shibamiya
IBM USA

Bill Stillwell
IBM USA

David Surman
IBM USA

Gary Wicks
IBM Canada

Chung Wu
IBM USA

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbooks document dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will 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 Redbooks documents to be as helpful as possible. Send us your comments about this or other Redbooks documents in one of the following ways:

- Use the online **Contact us** review 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. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Introduction to multisite sysplex data sharing

In this chapter, we examine the reasons why enterprises are expressing an increasing interest in multisite sysplex data sharing. The ability to share data over more than one data center affords many possible benefits. However, because this is a relatively new technology, enterprises are still trying to determine how implementing such a configuration could impact their costs, performance, and availability.

This book was written in response to this customer interest in multisite sysplex data sharing. It will help you to understand how the implementation of multisite data sharing could affect your environment.

1.1 Why use multisite data sharing

As mentioned, many enterprises are interested in multisite sysplex data sharing. In this book, we focus on the performance aspect of multisite data sharing (also known as Multi Site Workload, or an Active/Active configuration), specifically from a sysplex perspective.

Many installations have already implemented disk and tape mirroring to a second site, but still have all the production systems running in the same site (this is known as an Active/Standby configuration, or Single Site Workload in GDPS terms).

For this reason, we did *not* consider it necessary to address the following topics in detail in this document:

- Disk remote copy.

  Most large customers have already implemented some form of remote copy. It might be PPRC or some other form of synchronous mirroring, or it might be asynchronous, like Global Mirror or XRC. However, the point is that this is a technology and a cost that installations are already familiar with. There are also existing tools that provide accurate projections of the response time impact of implementing PPRC or XRC.

- Cross-site connectivity options.

  There are many options for connecting two sites. Those options are discussed in detail in other documents; for example, the IBM Redbooks publication *IBM System z Connectivity Handbook*, SG24-5444, discusses the many options that are available.

- The “dark fiber” bandwidth that will be required to deliver acceptable performance over the required distance.

  IBM provides services offerings to help you accurately determine your inter-site bandwidth requirements.

- Software costs.

  There may be software cost advantages if you can aggregate existing CPCs that are already in multiple sites. The rules and considerations for sysplex aggregation are described in detail in the IBM Redpaper *z/OS Systems Programmers Guide to: Sysplex Aggregation*, REDP-3967.

For the measurements in this book we used PPRC to remote copy the disk, and DWDM devices to provide cross-site connectivity. However, specifics about these topics are addressed in detail in other documents. Where we touch on these topics in this book, we refer to the appropriate documentation if you need more detailed information.

Because XCF and the Coupling Facility are critical components in data sharing, we concentrate on their role in data sharing and examine how the performance they deliver and the resources they consume can be impacted by adding some distance between the systems and the Coupling Facilities they are connected to.

Because normal Coupling Facility response times are so short (typically in the order of 5-50 microseconds), the relative impact of the fixed cost of distance (10 microseconds per km) is much higher on Coupling Facilities than on disk, where typical response times are currently in the 1000-3000 microsecond range. The focus of this book is on how these ingredients contribute to changes in system performance and capacity requirements.

Acknowledgement: This book was inspired by a set of measurements done by Pierre Cassier of IBM France in 2005. We are indebted to Pierre for all his help and advice in setting up the environment and workloads for this book.

1.2 Motivations for multisite data sharing

Many installations are using single-site sysplex data sharing. For these installations, the move to data sharing is a natural progression of their workload growth or their need for ever-higher levels of application availability.

Similarly, there are many installations that have more than one data center. This situation is often the result of company mergers, where you end up with the data centers of both former companies. Or it could simply be that your company did not want to “place all its eggs in one basket”.

However, the decision to investigate multisite sysplex data sharing is usually driven by some specific corporate requirement. We discuss typical reasons in this section.

1.2.1 Software cost savings

A common reason for enterprises to consider implementing multisite sysplex data sharing is that they already have two sites within a “reasonable” distance of each other. However from a software pricing perspective, there is no cross-site aggregation of their CPCs.

Changing the configuration so that all the CPCs are in the same PricingPlex[1] can potentially result in significant software cost savings. However, the sysplex aggregation requirement that 50% of the used MVS MSUs in each CPC must be in the same sysplex means that either existing work must be redistributed across the two sites, or else systems or entire sysplexes must be merged into one of the existing sysplexes.

Given these choices, spreading an existing data sharing configuration over two sites may be viewed as an option that delivers both technical and financial advantages. And from a technical perspective, this option is more attractive than merging two completely unrelated sysplexes. In other words, when the project is over and the savings are forgotten, will the technical benefits of the project continue to be viewed as worthwhile? If so, you have enjoyed two benefits for the price of one.

1.2.2 Improved availability

If your business requires extremely high levels of availability, you have hopefully already implemented data sharing, have already addressed any singe points of failure that might have existed in your configuration, and have a tried and tested disaster recovery plan in place. So, where do you go from there?

The steps you have already completed should protect you from the outage of any single component in your data center. But what if you have a catastrophic loss, such as a local fire that damages a number of CPCs? Or, much more likely, if you have planned site maintenance that requires all power to the computer room to be discontinued for a time? In that case, you would normally invoke your disaster recovery plan.

But even a tried and tested plan carries some level of risk, and your system will be unavailable for a number of hours at a minimum. In contrast, if your applications are running in both the primary and alternate site, actively sharing the data, you may be able to continue running with little or no outage across these events. And if you are able to exploit GDPS HyperSwap™[2], you can even switch the primary disk to the alternate site with little more than a short pause in processing.

[1] The set of CPCs that are treated as a sysplex aggregate for IBM software pricing purposes.
[2] Basic HyperSwap, available on z/OS 1.9 with a supporting APAR, is intended for use only within a single site.”

In a recent survey[3], respondents highlighted the following events as concerns which could take down an entire data center:

- Earthquake
- Fire
- Flood
- Snow storms
- Hurricane
- Tsunami
- Terrorist attack

If you live in the middle of the United States, you may not be too concerned about being affected by a tsunami; however, more mundane events like snow storms or tornados can be equally disruptive. If your configuration and workload enables you to do multisite data sharing, you may be able to protect your service across these events.

1.2.3 Improved flexibility

Many enterprises have a second site for backup purposes. And these sites are typically used to run the development or test systems. But it is also typical to find that the backup site will have far more capacity installed than is being used. For example, if your production systems span four large z9™-type servers, but your development workload only requires one server, there will be three idle servers in the backup site. And every time you upgrade a server in the production site, you must expand the capacity (possibly using CBU) of one of the backup servers.

But if you were able to effectively exploit multisite data sharing, you could spread the development and production work over both sites and both sets of servers. As more work is added, it could exploit spare capacity wherever it might exist over both sites. So, instead of having a total of eight servers, three of which are idle—you might have six servers, each equipped with CBU to handle a site failure event. The larger the disparity in size between your production systems and the systems running in the backup site, the more attractive this option becomes.

1.2.4 Regulatory reasons

The final common reason to consider multisite data sharing is for regulatory compliance. Many governments and central banks around the world now understand the vital role IT services play in economic and social stability. As a consequence, laws or guidelines such as the Sarbanes-Oxley Act and Basel II have been enacted which require providers of these services to be fully active again within a short number of hours after a disaster. In fact, there is even discussion of some governments requiring their largest financial institutions to have a three-site strategy.

For sites that are close enough to be able to exploit it, multisite data sharing can help you address these government requirements.

1.3 What are the considerations

It is said that “there’s no such thing as a free lunch” and this concept applies to multisite data sharing, as well. Increasing distances by kilometers or tens of kilometers will increase I/O and

[3] "How Far is Far Enough" © 2002 PreEmpt, Inc. http://www.preemptinc.com

Coupling Facility response times, potentially by significant amounts. These increases alone can affect online response times, batch elapsed times, and channel and CF link utilization.

Further, the increased response times may result in increased contention and other secondary effects. While it may be possible, with a considerable effort, to identify the impact of the *primary* effects of distance on the system, it is *not* possible to accurately model the impact of the secondary effects.

For this book, we measured a number of different workload types at various distances. In the corresponding chapters we discuss not only the results we observed, but also the tools and metrics we used, and the reason why we experienced the results we obtained. By providing this information, we hope that you can apply that understanding to your own environment.

Important: This book does not, unfortunately, provide easy answers about how a certain distance will impact your workload. Instead, the objective is partially to give you data points, so you have some idea of what to expect.

But more importantly, we want to provide you with an understanding of how distance impacts the components of your IT service, to help you apply our experiences to *your* unique environment.

The only way to know the exact impact distance will have on your environment is to actually measure it. Especially if you are contemplating data sharing over large distances, we strongly recommend that you consider renting DWDM equipment and fiber suitcases and actually trying this for yourself.

1.4 Layout of this book

As mentioned, the objective of this book is to provide information about how distance impacts performance in a data sharing environment. Therefore, Chapter 2, “Distance considerations for Parallel Sysplex technologies” on page 7, discusses the z/OS components that are impacted by increasing distances. These components play a key role, regardless of which database or transaction manager you will be using.

Chapter 3, “DB2 data sharing” on page 27, provides a high level overview of DB2 data sharing, primarily from the perspective of interactions with the Coupling Facility and DASD. This information will help you to understand the role that the Coupling Facility plays in data sharing, and how intensively the Coupling Facility is used. As a generalization, the fewer times a transaction needs to go outside the processor, to the Coupling Facility or disk, the less of an impact that distance will have on a data sharing application.

Chapter 4, “CICS/DB2 workload and metrics” on page 53, describes the hardware and software environment we used for the tests. Understanding our naming convention (for systems, database managers, and so on) will make it easier for you to follow the discussions about the results we encountered.

Chapter 5, “Our configuration” on page 89, provides the actual results of the measurements. The sample workloads we used are:

- CICS/DB2
- DB2 batch
- CICS® VSAM/RLS
- IMS Shared Message Queue

Note: This documentation will be released in a number of installments. The first installment examines CICS/DB2 measurements. Future installments will add information from CICS VSAM/RLS and IMS Shared Message Queue measurements.

For each workload, we provide the results of a base measurement at 0 distance. We then ran the same workload at 20 km, 40 km, 60 km, and 80 km. For each distance we provide not only the results, but also an explanation of *why* we saw the results we saw. We hope this explanation will help you apply our experiences to your environment.

Important: We stress again that the results we provide apply specifically to *our* environment and workload. A different mix of hardware, software, applications, or tuning options could provide very different results over the same distances.

2

Distance considerations for Parallel Sysplex technologies

Because the objective of this book is to discuss the relationship between performance and the distance between the components in a sysplex, we believe it is appropriate to start off by laying some groundwork. The distance becomes relevant any time a transaction or job requires the service of a device outside the CPU, or needs to communicate with another system in the sysplex. So, in this chapter we examine the role of the following components:

- Coupling Facility, including the different ways a transaction can be sent to the Coupling Facility, and the notification facilities provided by the Coupling Facility
- DASD, including DASD-related features that are especially relevant in a geographically distributed sysplex
- Channel-to-Channel devices (CTCs) for communication between the members of the sysplex

2.1 Impact of distance on z/OS

To understand how injecting distance into a sysplex will impact transactions and the overall system, we need to look at what happens during the life of a transaction. If we take a typical database transaction as an example, the transaction will spend some time using the CPU, some time waiting for resources held by the same or a different database manager instance, some time doing I/Os (logging and database reads and writes), some time interacting with structures in the Coupling Facility, and possibly some time communicating with peers in other members of the sysplex, either using the Coupling Facilities or CTCs.

Adding distance to the sysplex should not have much direct effect on the amount of CPU used by the transaction. And the amount of time the transaction waits for resources held by other transactions is a secondary effect of the distance. So the components that are directly impacted by distance are occasions when you have to go outside the processor, namely CF, disk, and CTC requests, so we will focus on these components before moving on to examine how these changes will affect your system and workloads.

2.2 Coupling Facility

In our day-to-day lives, the speed of light is such a huge number that we consider it to be largely irrelevant. However, if you compare the speed of light in a fibre (about 10 microseconds per kilometer round-trip) with current “good” CF response times (5 microseconds), you see that it is in fact very relevant if you are considering adding a number of kilometers between the system and the CF. In fact, we are quickly approaching the point where the time it takes the signal to travel over the CF link (either fibre or copper) is the largest component of CF response times.

Although the response time for both disks and CFs when accessed over a large distance (kilometers) is impacted by the same absolute amount by the speed of light, the *relative* impact on disk I/O response time is significantly less than the impact on CF response times, because CF response times are typically so much smaller.

For example, if you are getting 100 microsecond asynchronous CF response times today and you add 20 km between the CF and the z/OS system, the response time will increase to about 300 microseconds, which is a 200% increase. On the other hand, if your disk response times are currently 2000 microseconds and you move them 20 km from the z/OS system, we would again expect the response time to increase by 200 microseconds to roughly 2200 microseconds, which is only a 10% increase.

This means that workloads that are particularly CF-intensive are likely to see a higher relative impact as a result of increased distances than a workload that is more disk-intensive. As you will see later in this book, most sysplex data sharing implementations tend to have more interactions with the CF than with disk.

And most of the CF accesses are synchronous to the execution of the transaction (meaning that the transaction must wait for that request to complete), whereas many of the disk accesses are asynchronous. Therefore, any change (positive or negative) in CF response times has a more profound effect on transaction response times than an equivalent change in disk response times.

From this background, next we take a closer look at the various interactions between the CF and the z/OS systems in the sysplex.

2.2.1 Synchronous and asynchronous CF requests

Requests to a CF can be issued by z/OS in one of two manners: synchronously or asynchronously. When a request is issued *synchronously*, the XCF address space sends the request to the CF over a CF link, then spins on the CP (using up z/OS CPU time), waiting for the response. This delivers the best possible response time for this request, because the response from the CF is processed immediately after it is received back by z/OS. As a result, it is possible to deliver CF response times as low as 5 microseconds for certain types of CF requests when using current CF technology and ICB or IC links.

The other way a CF request can be issued is *asynchronously*. In this case, when the request is sent to the CF, control is returned to the operating system, which will then typically dispatch some other piece of work (a CICS transaction, for example). When the response subsequently arrives back from the CF, it is probable that some other piece of work is running, meaning that the response will not be processed until that other piece of work is completed and the XCF address space is dispatched to handle the returned CF request. While no time was spent spinning on the CP, CPU time is still consumed as a result of all the task switching that takes place.

If the configuration (CF, plus links, plus distance) can deliver response times that are less than the amount of CPU time that would be used to send a request asynchronously, it is more efficient to send a request synchronously, and spin waiting for its completion. However, as the response time for the request increases, there is a point where it becomes more efficient (in terms of used z/OS CPU time) to send the request asynchronously instead.

This is shown in Figure 2-1, where you can see that the CPU time for a synchronous request increases linearly as the response time (for any reason) increases. In contrast, the CPU time for an asynchronous request is the same, regardless of the actual response time being delivered by the CF.

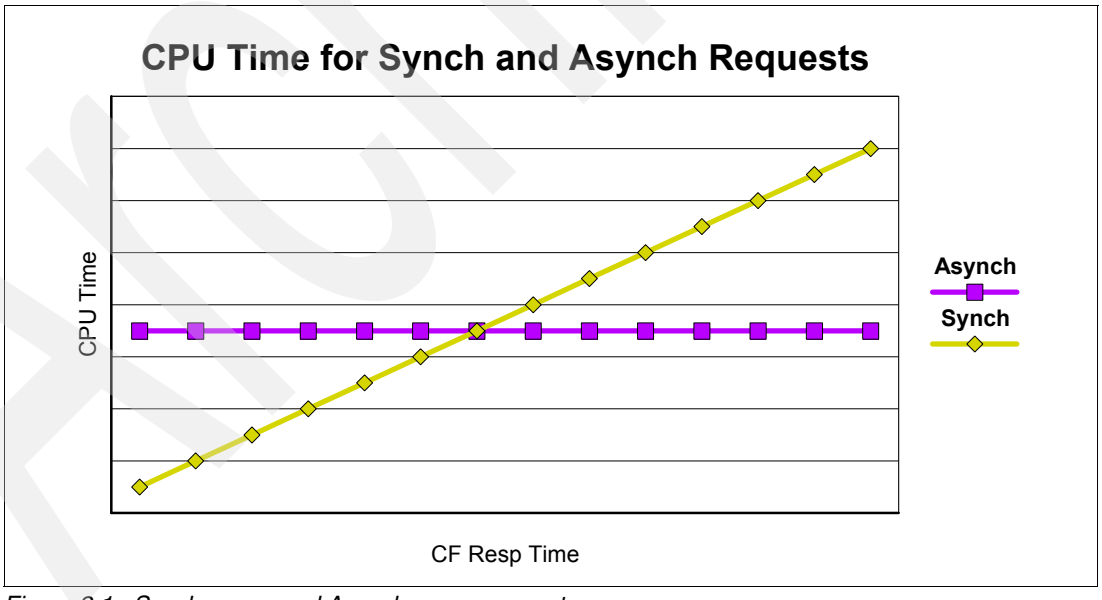


Figure 2-1 Synchronous and Asynchronous requests

Sending requests asynchronously will always result in longer CF response times than an equivalent synchronous request, all other things being equal. However, from an overall operating system perspective, in terms of throughput, it is more efficient to convert long-running CF requests to be asynchronous than to let them spin, consuming large amounts of CPU time without doing any real work.

As you can imagine, the precise point at which it becomes more efficient to treat the request asynchronously will vary, depending on the speed of the z/OS processor. A very fast processor can do more work in the same amount of time than a slower processor. So, the faster the processor, the lower the response time at which it becomes more efficient to send the request asynchronously.

For this reason, z/OS 1.2 introduced the so-called “Heuristic Synch/Asynch Conversion Algorithm”. Prior to this, there were some hard-coded rules in XES about when requests should be converted to be asynchronous. However, the heuristic algorithm is a far more effective and efficient way to control this processing. There is an excellent discussion of the heuristic algorithm in the section titled “How z/OS performs synchronous to asynchronous conversion of Coupling Facility requests” in the z/OS 1.9 or later level of *z/OS MVS Setting up a Sysplex*, SA22-7625.[1]

Because it is likely that some other unit of work is running when the response to an asynchronous request is received back from the CF, it is difficult to accurately predict how much longer the response time will be when a request is issued asynchronously rather than synchronously. We typically see increases from as low as 30 microseconds, to as high as hundreds of microseconds. However, for a given configuration, the increase is usually reasonably consistent.

You can find the impact for your configuration by looking in an RMF™ CF report and comparing the synchronous and asynchronous response times in the Subchannel Activity section of the report. In the example in Figure 2-2, the average synchronous response time was about 18 microseconds, and the average asynchronous response time was between 82 and 88 microseconds. This means that for this configuration with this mix of CF requests, the average additional time added as a result of sending the request asynchronously was about 65 microseconds. To put this in perspective, typical disk response times at the time of writing are in the region of 500 to 5000 microseconds.

Note that the `CHANGED` field in the RMF report represents the number of synchronous requests that were converted to asynchronous because there was no available subchannel; it is *not* the number that got converted by the XES heuristic algorithm. The number of requests that were converted as a result of the XES heuristic algorithm is not currently reported directly anywhere.

```
                                    C O U P L I N G   F A C I L I T Y   A C T I V I T Y
                                                                                                                      PAGE  13
      z/OS V1R9               SYSPLEX TEST               DATE 10/16/2007           INTERVAL 015.00.000
                              RPT VERSION V1R9 RMF       TIME 11.15.00             CYCLE 01.000 SECONDS

---------------------------------------------------------------------------------------------------------------------------------
 COUPLING FACILITY NAME = AAICF2
---------------------------------------------------------------------------------------------------------------------------------
                                                   SUBCHANNEL  ACTIVITY
---------------------------------------------------------------------------------------------------------------------------------
          # REQ                            ----------- REQUESTS -----------   ----------------- DELAYED REQUESTS -------------
 SYSTEM   TOTAL    -- CF LINKS --  PTH              #    -SERVICE TIME(MIC)-             #     % OF   ------ AVG TIME(MIC) ------
 NAME     AVG/SEC TYPE  GEN  USE  BUSY             REQ      AVG     STD_DEV              REQ    REQ    /DEL     STD_DEV      /ALL

 AA03       2492K CBP     3    3     0   SYNC       2272K   18.0      15.8  LIST/CACHE  204    0.0   7992       6071       1.9
          2768.4  SUBCH  21   21         ASYNC    210702    88.6     353.8  LOCK          3    0.0    9.0        3.0       0.0
                                         CHANGED     204  INCLUDED IN ASYNC TOTAL       207    0.0
                                         UNSUCC        0     0.0       0.0
 BB04       1582K ICP     2    2     0   SYNC       1486K   18.2      18.7  LIST/CACHE    0    0.0    0.0        0.0       0.0
          1758.2  SUBCH  21   14         ASYNC     97578    82.4     100.3  LOCK          0    0.0    0.0        0.0       0.0
                                         CHANGED       0  INCLUDED IN ASYNC TOTAL         0    0.0
                                         UNSUCC        0     0.0       0.0
```

Figure 2-2 Sample RMF CF Subchannel Activity report

[1] The thresholds implemented by the heuristic algorithm were subsequently adjusted by APAR OA21635.

To explain how this conversion of requests relates to and impacts the requestor, we need to take a step back. When an exploiter such as DB2 sends a request to the CF, that is actually a two-step process. For example, DB2 will issue an IXLCACHE request that gets sent to XES. On that request, DB2 can indicate whether it would *like* the request to be sent to the CF synchronously. Using the heuristic algorithm, XES then decides how the request will *actually* be sent.

The software that issued the request to the CF may or may not be aware if a request gets converted from synchronous to asynchronous—it depends on the options specified on the IXLxxxxx call. In Chapter 6, “CICS/DB2 measurements” on page 95, where we present the results of our measurements, we discuss how (or if) the exploiter is impacted by this conversion of its requests.

This discussion of synchronous and asynchronous requests is especially relevant in a multisite sysplex configuration for the following reasons:

- It is very likely that the majority of requests sent to a remote CF will be converted to asynchronous requests. As discussed previously, issuing a request asynchronously results in a further elongation of response times, with all the resulting consequences.
- On the other hand, the trend is that the threshold for converting requests to being asynchronous is getting lower with each generation of faster processors. The relative thresholds for various IBM processor models are shown in Figure 2-3. You see the downward trend with each new processor generation.

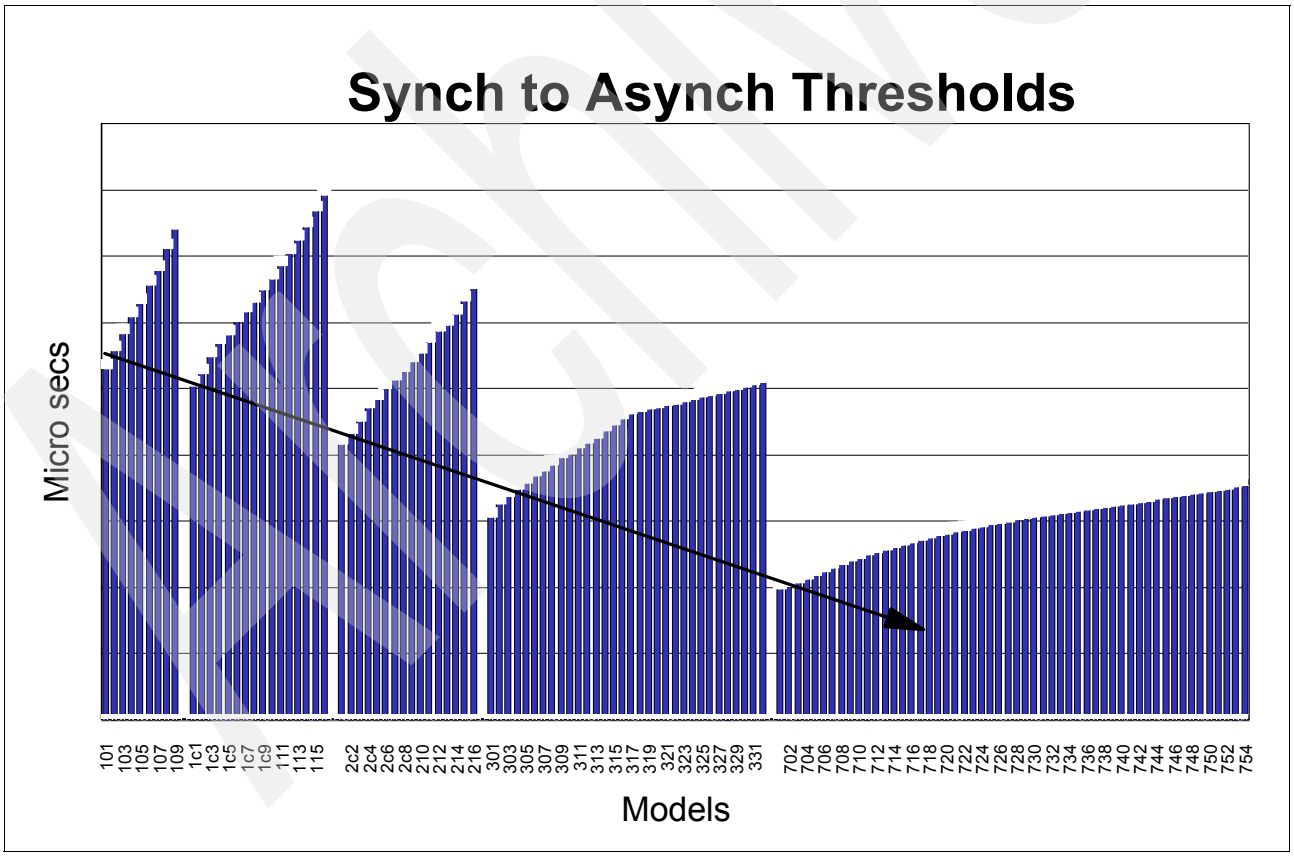


Figure 2-3 Thresholds for synch to asynch conversion

Extrapolating this trend into the foreseeable future, it is likely that nearly all CF requests, especially those issued to a CF connected using ISC links, will be asynchronous, regardless of the distance between the z/OS system and the CF.

For example, in one configuration we looked at with a z990 and a z9, both locally connected to the same CF using ISC links, we saw that 85.6% of requests from the z9 were asynchronous, compared to 53% of requests to the same CF from the z990.

The point is that the cost (in response time and secondary effects) of CF requests being issued asynchronously rather than synchronously is something that you are eventually going to encounter regardless of whether you have a single-site or a multisite sysplex.

Therefore, from a *longer-term* perspective, it would be more accurate to simply consider the additional response time increase caused by distance (10 microseconds per km) when projecting CF response times in a multisite sysplex scenario, and ignore the impact of requests being processed asynchronously rather than synchronously.

2.2.2 CF link and subchannel utilization

A very important aspect of the higher CF response times that are typical of a multisite sysplex is the increased utilization this causes for CF links and subchannels. Figure 2-4 shows the major components of the connections between z/OS and a CF.

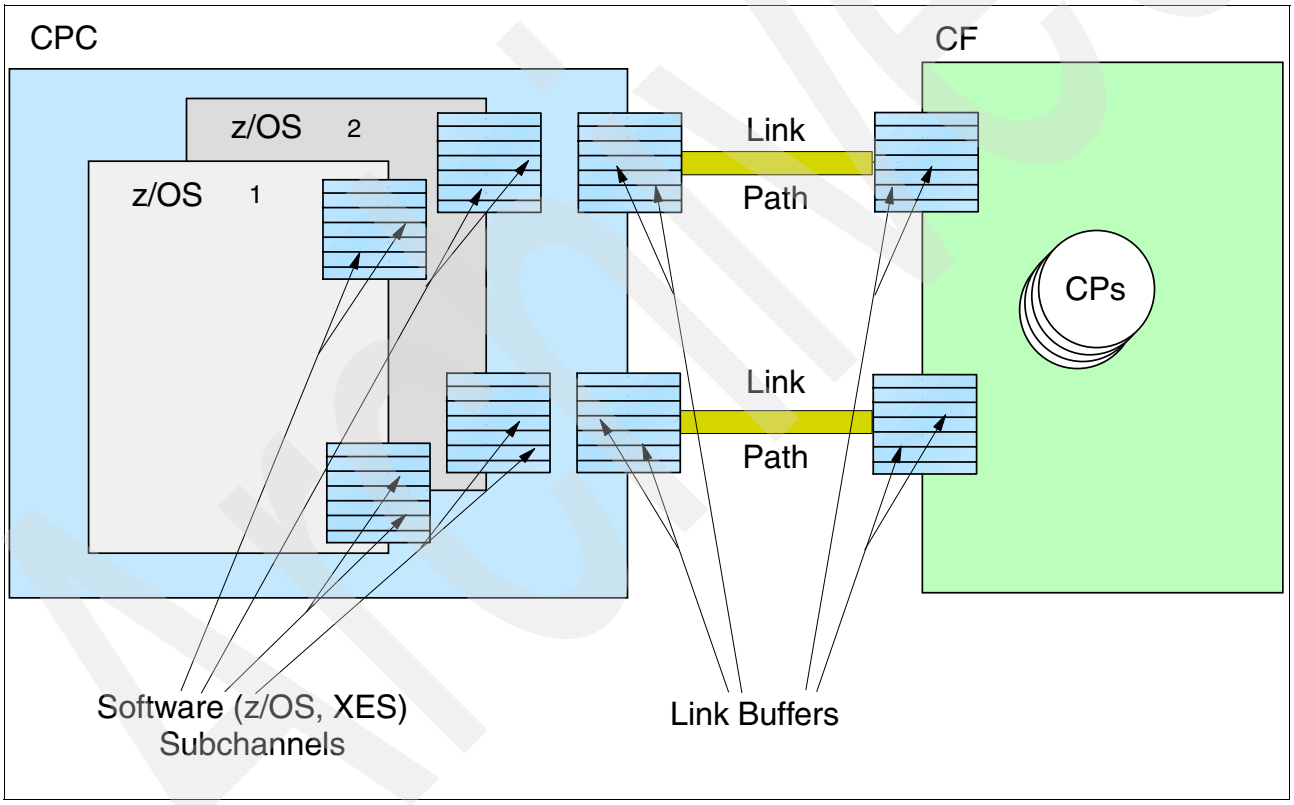


Figure 2-4 CF links, link buffers, and subchannels

At the hardware level, we have two CF links (the actual copper or fiber cables that connect the two processors). Each CF link configured in Peer mode has seven buffers at each end of the link, which are called *link buffers*. And in each z/OS system that is sharing the two links, there is one subchannel for each link buffer. So, in the configuration in Figure 2-4, there are two links, a total of 14 link buffers in the z/OS processor, and 14 subchannels in *each* of the two sharing z/OS LPARs.

When z/OS sends a request to the CF, it first selects an available subchannel. It then finds an available link buffer and the request is sent to the CF. Both the link buffer and the subchannel

remain “busy” until the XCF address space processes the response from the CF. So, the longer the CF response time, the higher is the utilization of both the subchannels in z/OS and the link buffers in the hardware.

If you have a CF that is currently located right beside the z/OS processor, you might be getting synchronous response times of 20 microseconds, meaning that both the subchannel and the link buffer are busy for 20 microseconds per request. If you then move that CF 20 km away from the z/OS system, the response time will increase by about 200 microseconds (10 microseconds per km).

Additionally, because the request is now likely to be issued asynchronously, there will be a further elongation due to the asynchronous processing, perhaps another 50 microseconds. Therefore, the response time (and thus the subchannel and link buffer utilization) has increased from 20 microseconds per request up to 270 microseconds per request, which is a significant increase.

Depending on your link and subchannel utilization in the single site configuration, you may require additional links (which provide more link buffers and subchannels) in order to keep your subchannel utilization below the IBM-recommended threshold of 30% when you move the CF away from the connected z/OS systems.

2.2.3 Coupling Facility list notification services

In a clustered environment, two of the basic services you will typically require include:

- The ability to easily, quickly, and efficiently move messages from one cluster member to another.
- The ability to have a shared queue of work items.

In a Parallel Sysplex, both of these capabilities are delivered using list structures in the Coupling Facility. However, you may wonder how the systems in the sysplex know that there is something in the CF that they would be interested in.

One mechanism would be for all connected systems to check with the CF every so often, to see if there is anything there waiting for them. However, if you check very often (to ensure a responsive system), then you increase the overhead. And if you reduce the overhead by checking less frequently, the system becomes less responsive.

The more effective alternative, as implemented by Parallel Sysplex, is for systems that have an interest in a queue within a structure to register their interest with the CF. Then, if an item (which could be a message or a work item) is placed on a queue that a system has expressed interest in, the CF will send a hardware signal to the HSA of the CPC the interested system resides on. This signal turns on an indicator in HSA that informs the associated system that something has arrived in the structure that it has an interest in.

Using this mechanism, the CF is able to update HSA in the target processor *without* generating an interrupt on the target system. In fact, it is possible that the target system is not even dispatched by PR/SM™ at the time that HSA is updated. Then, in order to find out if the CF contains any work for it, the target system only needs to check the setting of the indicator in HSA on the processor it is executing on, which is a process that takes virtually no time. This mechanism is called List Notification Services, and is a standard part of the CF list structure implementation.

The reason that this process is relevant to a discussion about distance is that it will take longer for the signal to reach a processor that is a considerable distance (kilometers) from the CF than one that is right beside the CF. Similarly, it will take longer for the retrieve request

from that remote system to get back to the CF. As a result, in extreme cases, you may find that systems that are closer to the CF containing a shared work structure will get a larger share of the work than those that are remote. There is really not much you can do to affect this, because it is simply a by-product of the distance between the CF and the connected system. However, this is something you should be aware of.

2.2.4 Coupling Facility cross-invalidation services

If you ask most database administrators what is the first thing they do to improve performance, the answer is likely to be “provide more database buffers”. In a large production environment, it is simply not possible to provide production-level response times without copious numbers of local buffers.

However, consider the case where you have a number of database managers updating a shared database. Assume DBM1 reads a page from database X. Then DBM2 reads the same page. As long as no one updates that page, there is no problem. But what happens if DBM1 makes an update? We now have an updated version of the page in DBM1, and an outdated version in DBM2, with the risk that a program in DBM2 might start using data that is not current.

One way to address this situation would be for the updating database manager to send a signal to every other database manager in the group, informing them that it has just updated that page. However, the likelihood is that most of the other managers will *not* have a copy of that page. So we have just consumed resources and wasted time passing information to database managers that have no interest in that information. And, as the number of managers in the data sharing group increases, the overhead will increase exponentially.

The alternative, and the mechanism implemented in Parallel Sysplex, is to use CF cross-invalidation services. In this case, whenever a database manager is about to load some data into a local buffer, it sends what is known as a Read and Register request to a cache structure in the CF, as shown in Figure 2-5 on page 15. This request lets the CF know that this system is going to have a copy of the data item in its local buffers. The request uniquely identifies the piece of data and also identifies a bit in HSA that will be associated with the buffer that will contain the data.

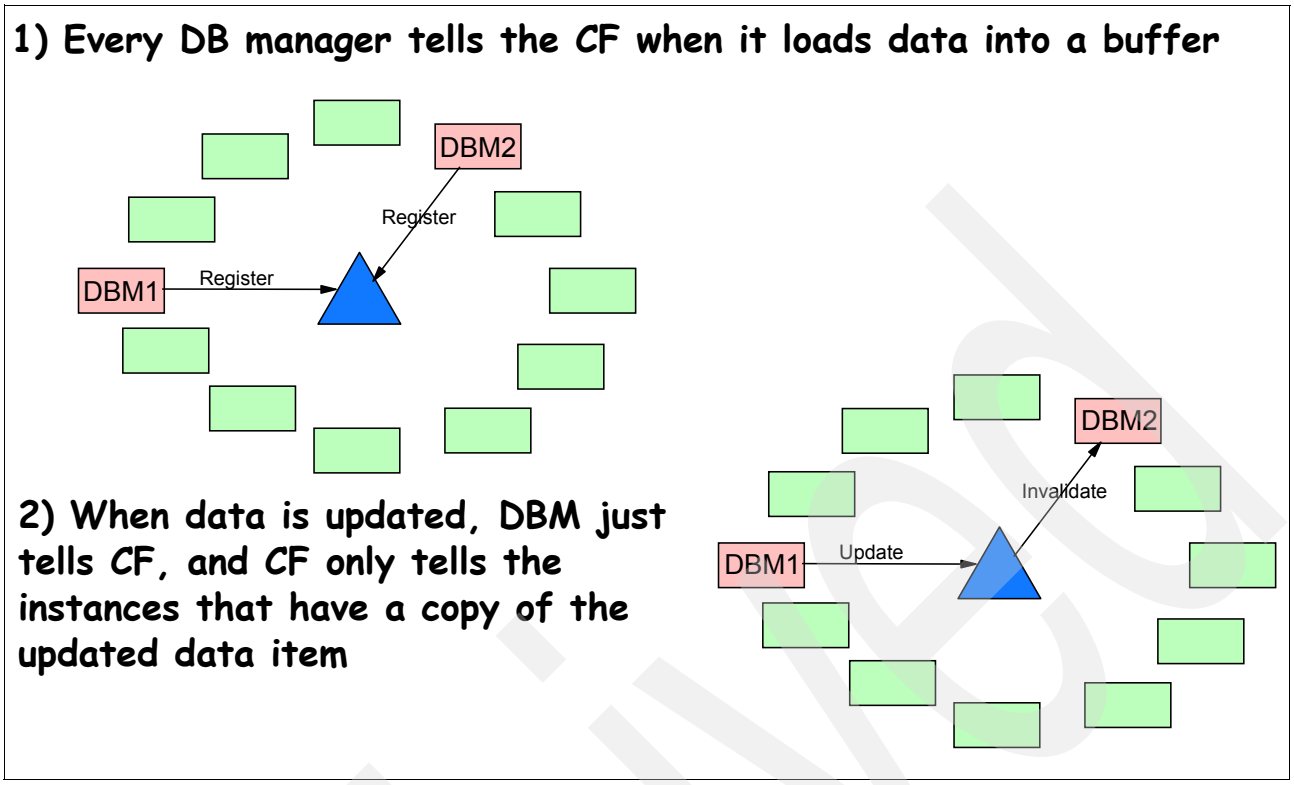


Figure 2-5 CF cross-invalidation services

So, assume we have data page XX, and it has been loaded into the local buffers of DBM1 and DBM2. Both database managers will have done a Read and Register to the CF, so the CF knows who has a copy of that data, as shown in Figure 2-5. Then DBM1 updates the data. Before it releases the lock on the data, it will send a request to the CF, informing it that the data has just been updated. Upon receiving this signal, the CF immediately sends a cross-invalidate signal to the HSA of the processor containing DBM2 (but not any of the other processors), instructing that the bit representing the buffer containing data page XX should be reset. After this task has completed successfully, the CF responds to the request from DBM1.

Every time a database manager wants to use some data from a local buffer (when using a shared database), it knows that it must check the bit in HSA to see if the data is still valid. In this example, if DBM2 tries to use its copy of the data, it will see that its copy is outdated and that it must retrieve an up-to-date copy.

The relevance of this processing to distance is that the further away the target of the cross-invalidate is from the CF, then the longer it will take to send the cross-invalidate signal, thereby elongating the CF request from the updating database manager, and in turn increasing the response time of the associated transaction. It is important to realize that this even applies if the database manager is in the same site as the cache structure, but the target(s) of the cross-invalidate signals are in the other site.

2.3 Disk

There are two aspects to the impact of distance on disk response times in a multisite data sharing configuration. One is the time required to mirror primary disk writes to the secondary devices. The other is the impact of the increased CPC-to-subsystem distance on all host I/Os issued to a remote primary subsystem.

For example, for systems SYS1 and SYS2 in Figure 2-6 which are in the remote site (Site 2, in this example), every I/O (both reads and writes) to the primary disk in Site 1 will be impacted as the distance between those systems and the primary disk increases.

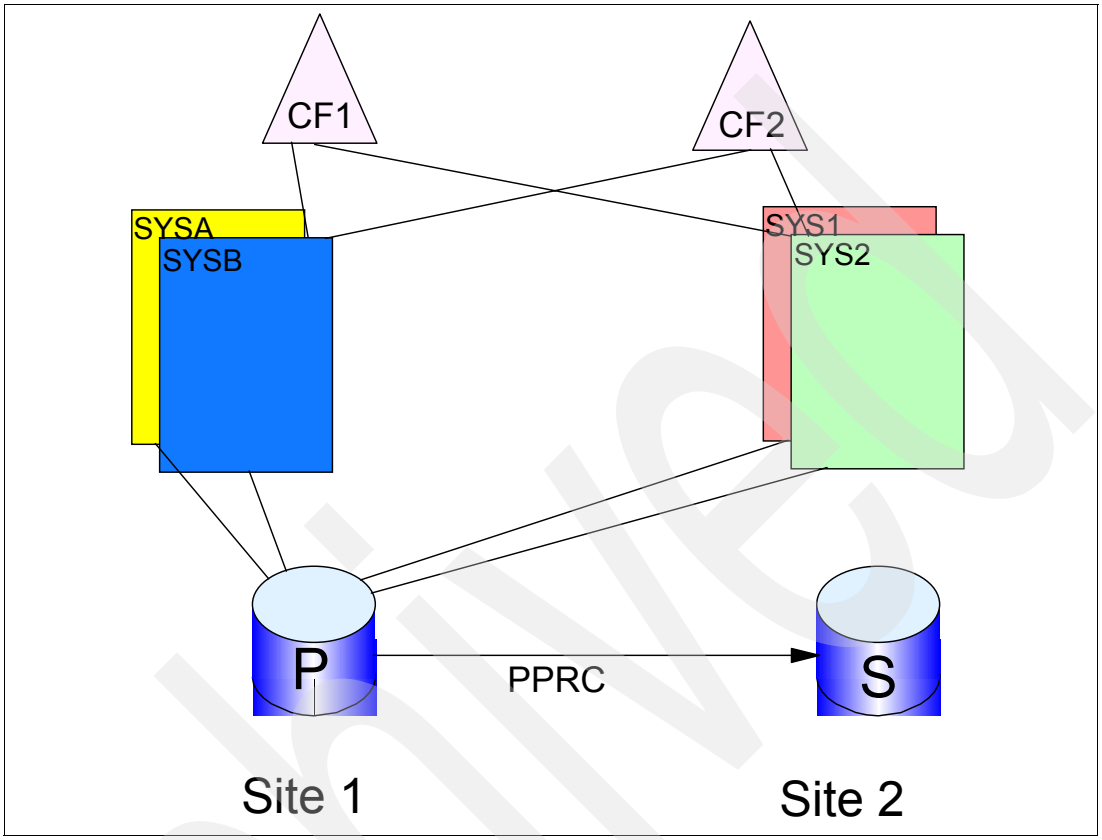


Figure 2-6 DASD connections

We expect that most customers considering a multisite data sharing configuration will already be mirroring their production volumes using some form of synchronous remote copy. Therefore, the mirroring cost is probably already included in your transaction and batch job elapsed times.

However, we also believe that most customers running single-site data sharing today will have all their primary disk subsystems in the same site as the z/OS systems. This means that whichever systems will be moved to the second site will have an additional impact; specifically, the additional time it takes for the I/O to travel the distance between the z/OS system and the primary disk subsystem. And, unlike mirroring (which only impacts writes), the additional distance will impact both reads and writes.

Using current-technology disk subsystems (DS8000™), both our modelling and our actual measurements showed that the impact of distance between the systems and the primary disk is fairly predictable, about an additional .1 ms. on each I/O (both reads and writes) per 10 kms. However, in some cases we also observed a small impact on the systems that are local to the primary disk subsystems. We believe this is because the long distance to the remote systems can result in higher utilization of the disk interfaces, which in turn would impact *all* systems using those interfaces.

The IntelliMagic™'s Disk Magic™ modelling tool (available to your IBM account team) can be used to predict the impact of changes in both the PPRC distance and also the processor-to-disk subsystem distance. It can also model the impact of changing the number

of channels or the number of PPRC links, changing the PPRC link type from ESCON® to FICON®, enabling PAV, and so on. We highly recommend the use of this or an equivalent tool to predict the primary impact of changes in the disk configuration.

PAV and HyperPAV

Assume you have a disk subsystem that today is 20 meters from a processor, and the response time for a given I/O is 1000 microseconds. This means that the UCB associated with the disk device will be busy for 1000 microseconds, and the physical device will probably be considered busy for a great amount of that time (say 700 microseconds).

Now, consider what would happen if that disk subsystem were moved 50 kms from the processor. The additional distance will increase response time by about 500 microseconds, meaning that the UCB associated with the device will also be busier for 500 microseconds longer. However, the device itself will still only be busy for the same amount of time (the 700 microseconds). This means that the limitation of how many I/Os we can perform to the device is actually the UCB, rather than the device itself. The resulting delay would show up in RMF reports as IOSQ time.

But, what would happen if we were able to have two UCBs for the device? We could now initiate a second I/O before the first one had completed, because we do not have to wait for the first UCB to become not-busy.

This is basically what Parallel Access Volumes (PAVs) do for you. PAVs allow a single physical disk device to be represented by more than one UCB, enabling multiple I/Os to be in process to the same device at the same time. Given what was said about longer distances driving up UCB utilization, you can see that the use of PAVs would be especially beneficial in a configuration where there is a considerable distance between the system and the primary disk devices *or* when there is a considerable distance between the primary and secondary disk devices.

MIDAWs

Every traversal of the distance between two disk subsystems, or the CPC and the primary disk subsystem, has a greater negative impact as the distance between the two boxes increases. Therefore, anything that can be done to reduce the number of traversals will provide performance improvements. One example would be the use of larger blocksizes (the use of System Determined Blocksizes would be especially helpful), where many data records can be sent or retrieved with a single traversal, rather than many traversals if the data set only had a single record per block.

Another way to reduce the number of interactions needed to move a given amount of data is by using MIDAW. The Modified Indirect Data Address Word (MIDAW) facility is a new and improved method of gathering or scattering data into and from non-contiguous storage locations during the execution of an I/O operation by the channel. Before the advent of MIDAW, CCWs with data chaining were used, but this technique adversely impacted the performance of FICON channels.

The use of MIDAW reduces the number of frames and sequences flowing across the link, making the channel more efficient.

Note: Optimizing the utilization of the channels and the switching infrastructure is especially important when using a geographically distributed configuration. For this reason, we used MIDAW for our measurements.

MIDAW is implemented by the z/OS Media Manager, and requires a IBM System z9® or later server, and z/OS 1.7 or later.

Media Manager is a z/OS operating system component responsible for performing disk I/O operations on behalf of most higher-level software components. It provides its users with a level of abstraction from the physical characteristics of the actual media.

Today, Media Manager is so pervasive that most I/Os in a z/OS system are generated by it. All Extended Format data sets use Media Manager, both for VSAM and non-VSAM. Prior to z/OS 1.3, VSAM non-Extended Format data sets did not use it, but now they do. Media Manager has always been used for linear VSAM data sets, such as those used by DB2, z/FS file systems, and the System Logger. HFS and PDSEs also use Media Manager. And it is used for the VTOC index and ICF catalogs.

Even though the most significant performance benefits of MIDAW are achieved with Extended Format data sets, non-Extended Format data sets also take advantage of MIDAW when using sequential processing.

For sequential accesses, z/Architecture®, in conjunction with the IBM 2105 or later disk subsystems, introduced the concept of “track-level” command operations as opposed to the original S/360™ architecture defined “record-level” command set. Media Manager implements track-level commands which are commonly used, for example, by DB2 during sequential read and write operations.

When using a track-level command to read data, the disk subsystem will allow a single CCW to read multiple records. When using a track-level command to update data, the disk subsystem will allow a single CCW to update multiple records that reside on the same track, but at the same time prohibiting a partial record update.

Track-level operations are beneficial because they potentially reduce the number of CCWs needed to transfer a given amount of data. And as data transfer speeds increase, the channel engine overhead of each CCW represents a larger impact. However, before the advent of MIDAW, these track-level operations did not achieve all the expected results, at least not with FICON channels, because of the need to use data chaining CCWs to address multiple main storage buffers.

By eliminating the need for CCWs with data chaining, MIDAW reduces the number of CCWs which compose a channel program, which in turn affects the number of frames sent across the link. By reducing the number of frames, it takes less time for the FICON channel to process the channel program. As a side effect, MIDAW also helps reduce disk subsystem host adapter utilization; although the host adapter has to process every CCW, indirect data addressing is transparent to it.

For more information about MIDAW, refer to the IBM Redpaper *How does the MIDAW facility improve the performance of FICON channels using DB2 and other workloads,* REDP-4201.

2.4 CTCs

During the measurements we ran in this project, we discovered an interesting aspect of CTC performance. We found that the response time for XCF signals going in one direction was significantly different to the response time of signals going in the opposite direction over the same CTC. This led us to a better understanding of data flow when using FICON CTCs.

When z/OS sends a signal from one system to another over a CTC, the sending system sends the I/O to the CTC control unit. The control unit then sends an interrupt to the z/OS at the other end of the CTC, and that system then sends an I/O to collect the data.

When using FICON CTCs to connect two CPCs, the channel microcode attempts to balance the number of control unit functions at each end of the channel. However, if you only define one CTC control unit per FICON channel, then obviously the control unit function must be at one end or the other. This is shown in Figure 2-7.

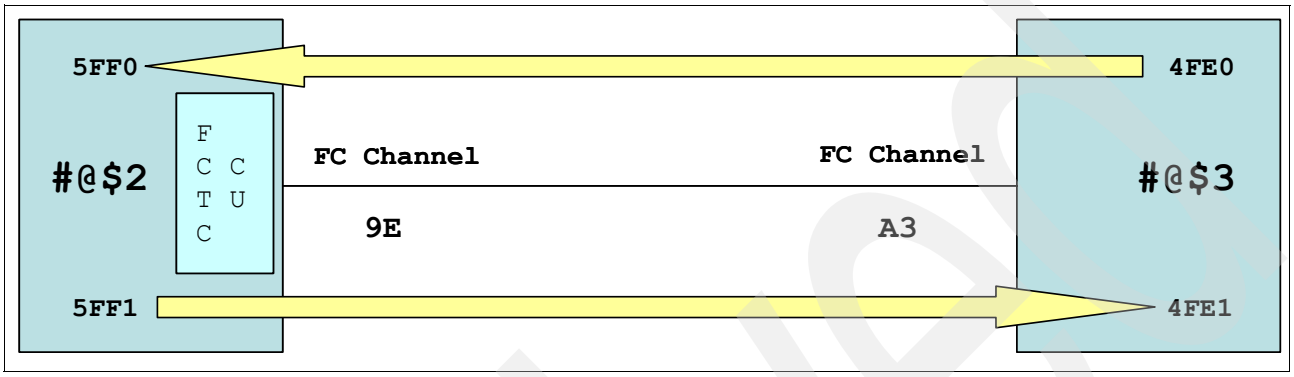


Figure 2-7 Control unit placement with FICON CTCs

In this example, if system #@$2 sends some data to system #@$3 via CTC 5FF1, the I/O from #@$2 to the CTC control unit takes very little time. The control unit then sends an interrupt to system #@$3. System #@$3 next sends an I/O to device 4FE1—which is managed by the CTC control unit close to #@$2—to collect the data. If the distance between the two systems is considerable, the time to send the interrupt signal to #@$3 and the subsequent retrieve I/O will be significantly longer than the time to send the data from system #@$2 to the CTC.

On the other hand, if system #@$3 is using the same CTC control unit to send data to system #@$2, there is one long I/O to send the data over to the control unit, but the two subsequent signals will be much shorter. The effect of this can be seen in Figure 2-8.

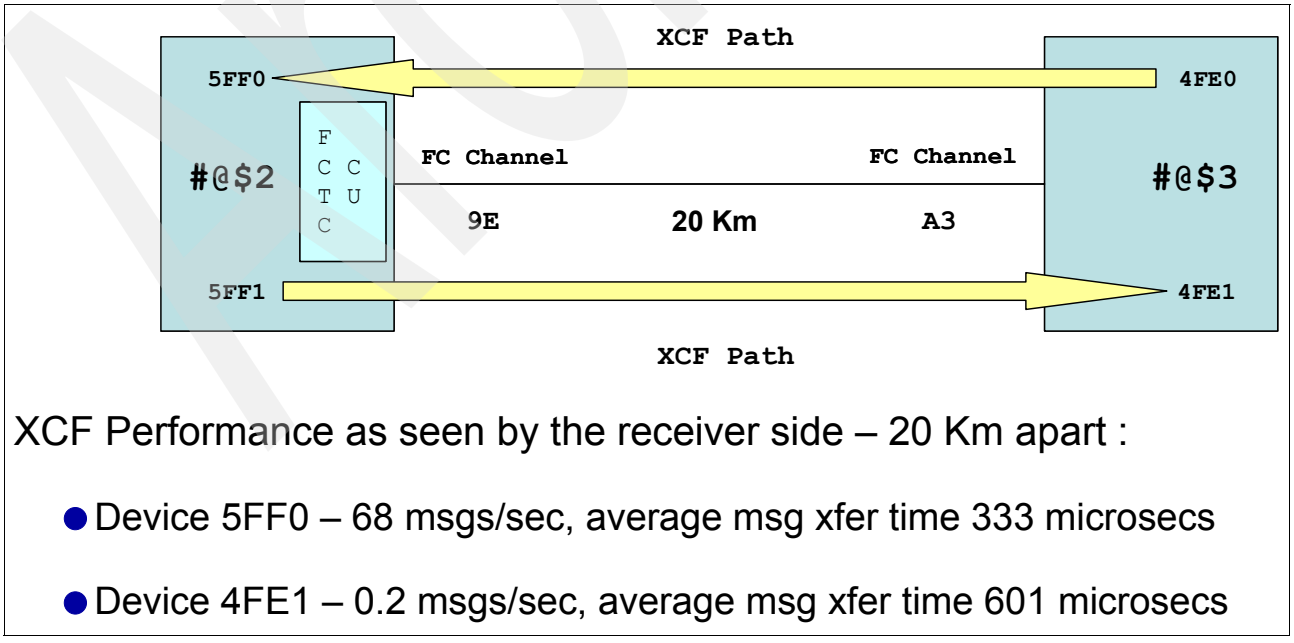


Figure 2-8 Response time impact of CTC CU function placement

You can see that the time to transfer data from system #@$2 to #@$3 (601 microseconds) was nearly twice as long as the time to transfer data over the same CTC, but in the other direction (333 microseconds). As a result, XCF tended to prefer the CF when sending messages from system #@$2 to #@$3, but used the CTC much more when sending messages in the opposite direction (because XCF will automatically select the available path that is providing the best performance).

To get equivalent performance in both directions, we recommend defining *two* CTC control units per FICON CTC channel, as shown in Figure 2-9. This should generally result in a balanced number of CTC CUs in each processor.

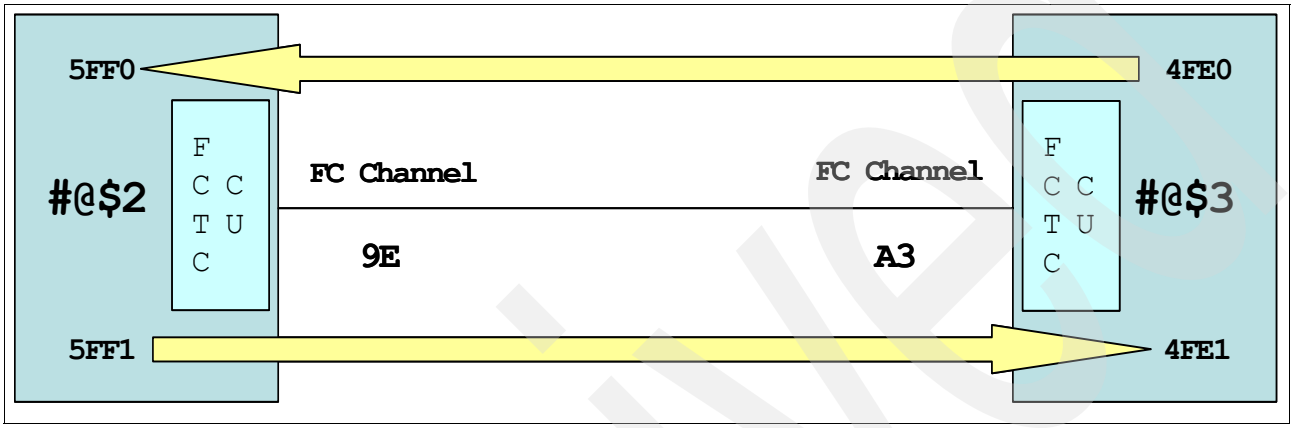


Figure 2-9 CTC CU configuration for optimum performance

2.5 Other important considerations

In addition to the increased elapsed times caused by longer distances to the disk, the CF structures, and to systems at the other end of a CTC, there are other aspects of a sysplex that are especially important when the sysplex members are geographically distributed. In this section we provide information that will be beneficial in understanding the results that we present later in this document.

2.5.1 XCF/XES role as global lock manager

As mentioned previously, all CF requests from exploiters such as DB2 and IMS are actually sent to XES rather than directly to the CF. Depending on the type of request, XES may make some changes to the request; for example, when the exploiter indicates it wants the request to be sent synchronously, but XES (using the heuristic algorithm) decides that it is more efficient to send the request asynchronously.

For lock requests, XES (together with XCF) may actually play a significantly larger role than most people are aware of. To help you interpret RMF and database manager contention reports, it is useful to understand how XES is involved in processing CF lock requests, especially when there is contention.

First, we need to examine the layout of a lock structure in the CF, as shown in Figure 2-10 on page 21. You see that each lock entry contains a single byte that identifies the system having an exclusive interest in a resource that is protected by this lock entry. The other byte(s) (there can be between 1 and 7 other bytes) have a bit for each system in the sysplex.

In the example, systems 1, 2, and 7 have a shared interest in a resource that is protected by the second lock entry, and system 03 is identified as the global lock manager for that lock entry (because system 03 is the system identified in the Excl Byte and the lock entry is in contention).

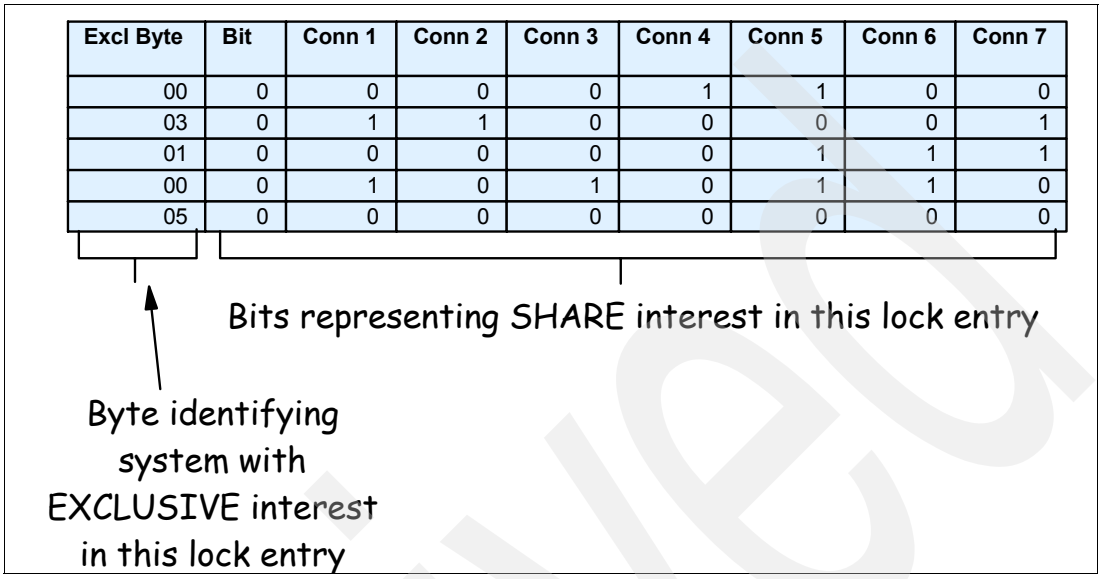


Figure 2-10 Layout of lock structure in the CF

Because the potential number of serializable resources in a sysplex is huge, it is infeasible to have one lock entry for every possible resource. Therefore, the system uses a hashing algorithm to identify which lock entry should be used to represent a given resource. And, because there are fewer lock entries than potential resources, it is possible to have more than one resource protected by a given lock entry.

So, assume a user on SYSA wants to read a data set called AAA.BBB that is protected by lock entry 1. And another user on SYSA (or any system in the sysplex) wants to update a data set called YYY.ZZZ that is also protected by the same lock entry. Because the CF does not know the names of the resources associated with each lock request, all it can tell is that one user wants shared access and another wants exclusive access to a resource protected by that lock entry. The CF considers this to be a contention situation and passes a return code indicating this back to the requesting XES.

In this case, because both requests are originating in the same system, XES has access to information about which resources are being protected by each lock entry. If XES sees that the contending requests are actually for different resources (as in this case), it can determine that this is false contention (there really are not two different types of requests for the same resource) and is able to grant access. However, if the two requests actually are for the same resource, then XES must pass the request to the contention exit of whichever product owns that lock structure (GRS or IRLM, for example).

So, what does all this have to do with a “global lock manager”, and why do we care? It is because the global lock manager is the first system to request exclusive access to a lock entry when there are already other requestors that have shared access to that lock entry. Therefore, from this point until all the contention goes away, that system will continue to be the global lock manager for that lock entry, and will have to be consulted on *every* request to obtain or release the lock from any system in the sysplex.[2]

[2] Note that it is likely that different systems will be the global lock manager for the different lock entries.

Consider a successful lock request (that is, there is no contention for the lock entry). The resource owner sends an IXLLOCK request to XCF. XCF sends the request to the CF. The CF determines that there is no contention, and responds that the request was successful, potentially in as little as 5 microseconds.

Now consider what happens when there *is* contention for a lock entry. In this case, the request will again be sent to the CF, but the CF fails the request, and provides the requestor with the identify of the global lock manager. The requesting system then has to package up the request and send it over XCF to the global lock manager. The global lock manager then has some work to do, to determine whether the request can be granted (potentially including calling the resource owner's contention exit). It then packages the response and sends that back over XCF to the requestor.

All of these communications back and forth to the global lock manager consume both resources and time. In addition, not only is the global lock manager called when you request a lock entry that is in contention—but it is also called when you free up that resource, because the global lock manager must be aware of *all* activity in the sysplex related to obtaining or freeing lock entries that it is managing.

All communication relating to lock entries that are in contention are sent over XCF groups with names that start with IXCLOxxx. Each CF structure that contains a lock component will use a different XCF group. To find the XCF group that is associated with a given structure, issue a **D XCF,STR,STRNM=structure_name** command, as shown in Figure 2-11.

The XCF GRPNAME line contains the name of the XCF group related to this structure. The level of activity in the XCF group associated with a lock structure is an indication of the level of contention experienced by that structure, because most of the activity in that group is related to requests to obtain or release a lock entry that is in contention.

```
D XCF,STR,STRNM=DB8QU_LOCK1
IXC360I  15.50.18  DISPLAY XCF 983
STRNAME: DB8QU_LOCK1
 STATUS: ALLOCATED
 EVENT MANAGEMENT: MESSAGE-BASED
 TYPE: LOCK
...
ACTIVE STRUCTURE
----------------
 ALLOCATION TIME: 12/17/2007 17:15:03
 CFNAME         : FACIL05
 COUPLING FACILITY: 002094.IBM.02.00000002991E
                    PARTITION: 1E   CPCID: 00
 ACTUAL SIZE    : 4608 K
 STORAGE INCREMENT SIZE: 512 K
 USAGE INFO      TOTAL     CHANGED    %
  ENTRIES:        6942           0    0
  LOCKS:       1048576
 PHYSICAL VERSION: C1A92309 BB90A48B
 LOGICAL  VERSION: C1A92309 BB90A48B
 SYSTEM-MANAGED PROCESS LEVEL: 8
 XCF GRPNAME    : IXCLO015
...
```

Figure 2-11 Displaying the XCF Group for a CF lock structure

Compared to the 5 microseconds for a successful lock request, this process is likely to take hundreds of microseconds. It also generates additional work for the global lock manager (meaning more CPU time) and for XCF on each of the interested systems.

The reason this is relevant to a discussion about distance is that the longer elapsed times that result from the longer distances increase the likelihood of lock contention (because each lock is likely to be held for longer.) So, not only does increased contention result in longer elapsed times for transactions and batch jobs, but it also generates additional work for XES and XCF.

The subject of the global lock manager role of XCF and XES is too complex to cover in more detail in this document. More information may be found in *z/OS Sysplex Services Guide*, SA22-7617. The important thing to remember here is that, as contention for the use of lock entries increases, it is likely that you will see a corresponding increase in the CPU used by the XCF address space, and also an increase in the number of signals in the IXCLOxxx XCF groups.

2.5.2 Failure isolation

Some Coupling Facility exploiters have a requirement that the CF structure should not have any single points of failure in common with any of the systems connected to that structure. We will use the IRLM lock structure as an example to illustrate this.

In the illustration in Figure 2-12, we have two z/OS systems (called #@$2 and #@$3), each in their own CPC, one DB2 system in each z/OS (called D8Q1 and D8Q2), and two standalone Coupling Facilities. The IRLM lock structure is in the Coupling Facility on the left, and you can see that the structure contains locks from D8Q1 and D8Q2.

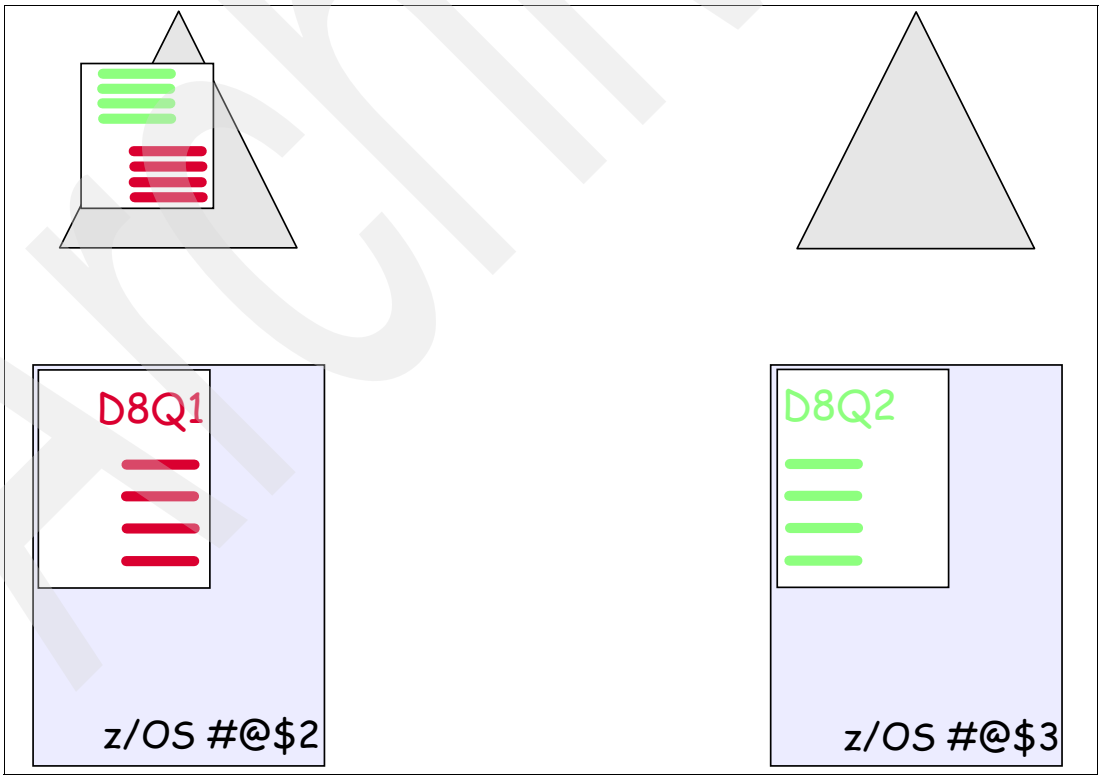


Figure 2-12 Single point of failure considerations for CF structures

So, what happens if either of the IRLMs, either of the z/OS systems, or even either of the CPCs were to fail? The surviving DB2 must be sure that it will not try to update any data that was in the process of being worked on by the failed system. However, because the lock structure is still accessible, and still contains the locks that were held by the failed DB2, all that data is protected and the surviving DB2 can proceed.

And what would happen if the Coupling Facility were to fail? In that case, there is no longer a central repository of sysplex-wide lock information. However, all the members of the data sharing group are still alive, so the IRLMs will allocate a new structure in the other Coupling Facility and repopulate it with the lock information from their own in-storage copies. As soon as the lock structure is up-to-date again (which should only take seconds), both DB2s can continue operating.

So, in both of these cases (single failure), there is only a pause in DB2 processing.

But, what happens if a single failure were to affect both the Coupling Facility *and* one or more of the connected DB2s? In this case, it would not be possible to repopulate the lock structure because at least one of the IRLMs that was connected to it would be gone. Therefore, all surviving DB2s in that data sharing group would immediately abend. On the subsequent group restart, they would realize that one member of the group was missing and would read the logs of that member to determine what locks it held at the time of the failure. A new lock structure would be allocated and populated with this information, and the other DB2s would then be ready to resume work.

Now, one of the reasons for implementing a multisite sysplex configuration is that you would hope that the systems and applications in one site would be able to withstand the loss of the other site. This is a very complex topic, which is beyond the scope of this book. However, it does have a performance-related aspect that must be considered here.

In a configuration where all systems and CFs are in the same site, and the relevant structures are in failure-isolated CFs, it is unlikely that a single failure (short of a site-wide failure that stops everything) will impact both a CF and a connected system.

However, in a multisite sysplex configuration, it is possible that the loss of one of the sites would impact both a CF and one or more of the connected systems. If that CF contains structures that have a failure-isolation requirement, then the connected exploiters in the other site would be affected, even though the systems they are running on may continue to run.

One way to address this is to use System-Managed Duplexing for those structures that require failure isolation. However, the performance impact of using System-Managed Duplexing over large distances must be considered. You need to investigate the performance impact on a case-by-case basis, also bearing in mind the CF Level being used.

Another option is to place one of the CFs in a third (failure-isolated) location and place the structures that require failure isolation in that CF. This will allow you to potentially lose any one of the three sites without that resulting in a data sharing group-wide outage (this assumes that you are exploiting a function like HyperSwap to mask primary DASD failures).

2.6 Secondary effects

At this point, we have examined all the components that are *directly* affected by increased distances. And with the right reporting tools and a significant amount of effort, it is possible to estimate the primary effect that distance will have on the response time of a given transaction. What is *not* possible to accurately predict is the number of *secondary* effects that will result from increased CF and I/O response times. And, because these effects cannot be

sized, it is also not possible to project how these will impact transaction response times and batch job elapsed times. However, by understanding the factors that can cause the secondary effects, you can act to keep those effects to a minimum.

But, what exactly are “secondary effects”? Secondary effects are delays or additional resource used as a follow-on result of the primary effects of distance (longer CF and I/O response times). One example might be additional CPU usage by a transaction manager that has to handle more concurrent transactions (because each one runs for longer). However, based on our experiences, the largest secondary effect was increased contention.

Accurately predicting contention changes is impossible because it depends not only on the number of transactions running concurrently, but also on the resources requested by those transactions.

Contention considerations

This section discusses some of the characteristics of a workload that determine the level of contention likely to be experienced by the associated applications.

If only one transaction (or job) is running at a time, there is no chance that a resource it requires is already serialized by someone else. Thus, there is no contention. But as the number of concurrent transactions (or jobs) increases, the likelihood that a required resource is already held by another transaction will also increase.

The possibility of contention also depends on the number of resources. If you have a DB2 table with a million rows, and each transaction only selects one row, the chance of another transaction holding that same row is probably quite low. But if the table only has two rows, and some or all transactions require serialized access, then the chances of encountering contention are far greater.

The distribution of work is also a factor in contention. DB2, for example, behaves differently if a table is being updated from more than one system than if only one system is using it. So, depending on which system the transactions get routed to, the resulting contention could be different.

You also need to consider contention for physical resources. For example, CF links only have seven subchannels. If the synchronous response time at 0 distance is 20 microseconds, and you are driving 70,000 requests per second over just one CF link, the subchannel utilization will be 20%. If you move the CF to 3 km away, the requests would now be mainly asynchronous, raising the response time to perhaps 100 microseconds. With the same number of requests, subchannel utilization will now be 100%, which would obviously result in significant contention for subchannels. Although we can estimate the subchannel utilization, we cannot predict how the increased contention for access to the CF will impact the exploiters.

Of course, in a real production environment, things are much more complex than this. For your important applications, it is highly likely that there will be a large number of transactions running concurrently. And each transaction might access 10 different DB2 tables, some of which might contain a small number of rows, and others which might contain millions of rows. And some tables might only be used by transactions running in one DB2, but others might be used by transactions spread over multiple DB2s.

It is this complexity and unpredictability that makes it impossible to accurately predict how CF and I/O response time changes will translate into increased contention. In the mainframe area, customers have become accustomed to the idea that IBM has tools to help project the impact of many configuration changes (an example is the Disk Magic tool, which accurately models the effect of changing the DASD configuration). Based on this experience, it is

understandable that people would expect IBM to have a tool that would accurately predict the impact that distance would have on response times and utilization. However, this is not possible because of the unpredictable nature of contention for resources.

For a given distance, you cannot reduce the response time impact of the distance, so the main way to minimize the impact is to try to keep contention as low as possible. We will discuss contention in a lot more detail in subsequent database chapters, but for now we simply want to state that secondary effects cannot be ignored, and in some cases they may actually cause a much larger impact on response and elapsed times than the primary effects of distance.

2.7 General capacity issues

To this point, the discussion has focused on the impact of distance on transaction response times. And this is likely to be the area that would receive the most attention if you implement a multisite data sharing configuration.

However, as the z/OS system programmer, you are also concerned about the "bigger picture"; that is, how the implementation of such a configuration will impact your system and hardware utilization as a whole.

Obviously, channel utilization will increase, in line with the increase in I/O response times. However, the relative impact on channels is significantly lower than the corresponding impact on CF links, mainly because many more subchannels (and therefore, parallel operations) are supported on channels than on CF links. Nevertheless, it would be prudent to monitor your channel utilization and perhaps rebalance the workload or add channels if channel utilization is already high before you make any changes.

As described in the example in 2.6, "Secondary effects" on page 24, CF subchannel utilization, and corresponding link buffer utilization, would be expected to increase significantly as the distance between the CF and the connected systems increases.

As discussed in "PAV and HyperPAV" on page 17, the longer I/O response times caused by the increased distances will drive up IOSQ time. You should monitor this and possibly plan on increasing the number of PAV aliases if you find that number increases.

In our measurements we saw a small increase in the amount of CPU time per transaction as the distance increased. We also found a small decrease in the MVS capture ratio, dropping by about 1% between 0 km and 80 kms. More information about the specifics of these observations is provided in the chapters where we discuss the results of each set of measurements.

3

DB2 data sharing

This chapter discusses the aspects of DB2 data sharing that are particularly relevant when the sysplex components are spread over a distance. In order to understand the results of our DB2 workload measurements and how they can be related back to your environment, it is important to understand the interactions between DB2, IRLM, XES, and the Coupling Facility and disk.

3.1 Introduction

As mentioned in Chapter 1, “Introduction to multisite sysplex data sharing” on page 1, Coupling Facility response times are especially sensitive to distance. And because DB2 data sharing is an intensive user of the Coupling Facility, we feel it is valuable to provide a reasonably detailed description of how DB2 data sharing works, concentrating on the interactions between DB2 and the Coupling Facility. We then move on to briefly describe the life of a DB2 transaction, focusing on which actions delay the execution of the transaction, and which actions can execute in parallel with the transaction.

This chapter is not intended to provide a comprehensive description of DB2 data sharing; that information can be found in *DB2 UDB for z/OS V8 Data Sharing Planning and Administration*, SA18-7417, and in various IBM Redbooks publications about DB2. Instead, here we only examine one aspect of DB2 data sharing, to provide readers who are unfamiliar with DB2 with some background and enable them to better understand the chapters in which our measurement results are discussed.

3.2 DB2 sysplex data sharing

Before discussing how DB2 sysplex data sharing works, this section briefly explains the concept of inter-DB2 read/write interest.

Prior to the introduction of sysplex data sharing, DB2 always knew that any data in its local buffers was the latest available version of that data. However, the advent of DB2 sysplex data sharing introduced the possibility that another DB2 could potentially have updated a copy of that data since the first DB2 read it into its buffers. In order to provide DB2 with information about whether or not a table space is actually being updated by another DB2, DB2 introduced the concept of Inter-DB2 read/write interest.

Inter-DB2 read/write interest indicates a situation where one DB2 subsystem is updating a table space (or table space partition), and one or more other DB2 subsystems are reading or updating the same table space (or table space partition). If this situation exists, DB2 will use facilities in the CF to ensure that the data is protected, and that all DB2s with an interest in that table space (or table space partition) are made aware if any other DB2 updates a related page that is contained in a given DB2’s local buffers. A table space or table space partition that has inter-DB2 read/write interest is known as being *GBP-dependent*.

Because the use of the CF adds time to each transaction, DB2 does all it can to avoid the use of the CF. For example, if only one DB2 is using a table space, there is no need to send updated pages to the Group Buffer Pools (GBPs) in the CF. Also, it is not necessary to send page or row-level serialization information to the lock structure if no other DB2 is using that table space.

In the example of DB2 data sharing we present in this chapter, it is assumed that at least one of the two DB2 subsystems in the example already has a write interest in the table space and the other DB2 has at least a read interest. It is also assumed that table space is therefore deemed to be GBP-dependent, meaning that all committed updates will be sent to the GBP structures.

For a more detailed introduction to DB2 sysplex data sharing, we recommend reading the IBM Redbooks publication *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322.

3.3 DB2 data sharing - the high altitude view

In this example, we will show a variety of reads and writes from a 2-way DB2 data sharing group and use these to illustrate how DB2 interacts with its CF structures.

The configuration we use for the example consists of two z/OS systems (called #@$2 and #@$3), each running a member of the DB2 data sharing group (D8Q1 and D8Q2), as shown in Figure 3-1.

We have two Coupling Facilities (FACIL05 and FACIL06). FACIL05 contains the GBP structures, and FACIL06 contains the lock structure.[1]

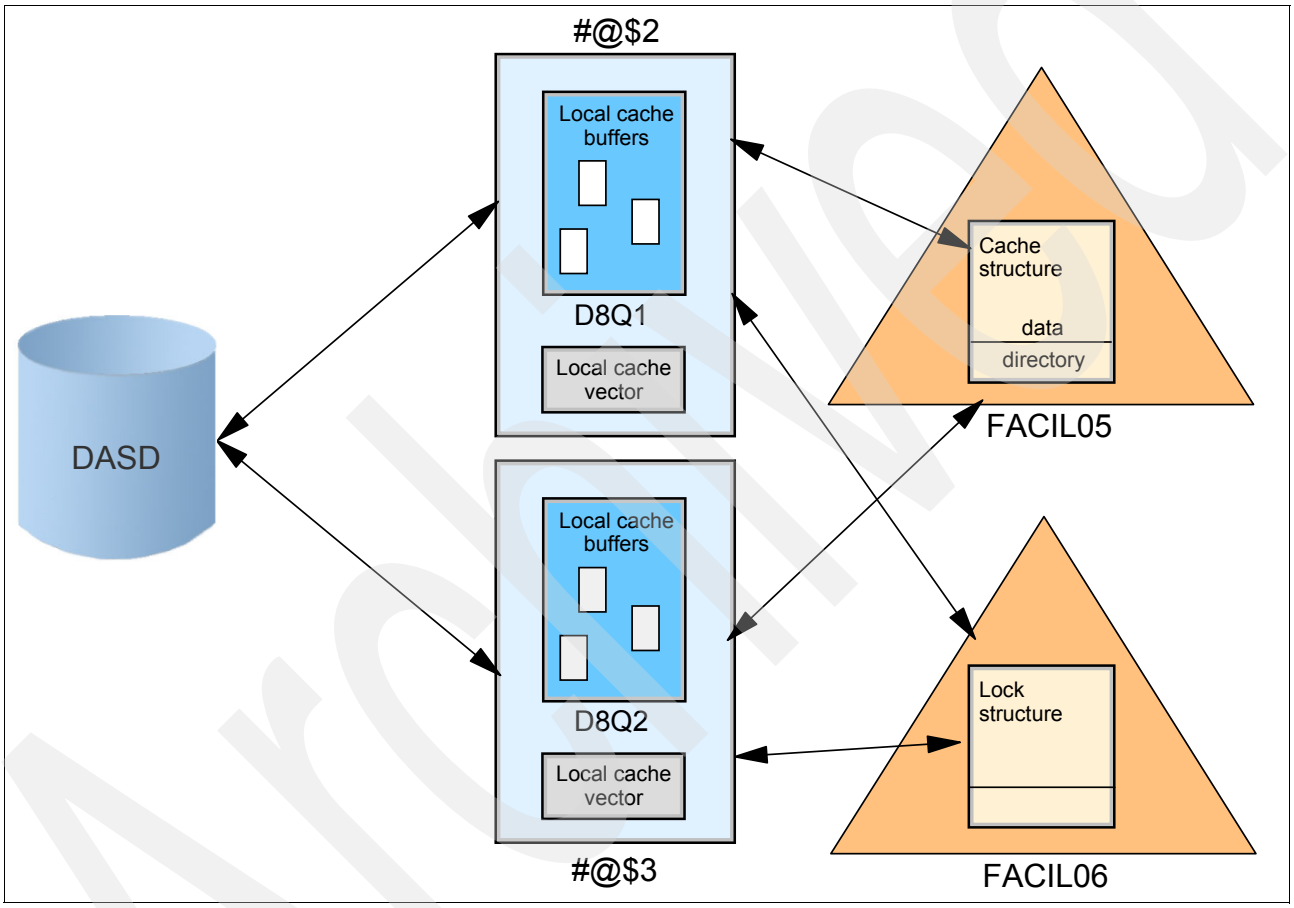


Figure 3-1 Configuration for DB2 data sharing explanation

[1] For simplicity, we show GBP structures that are in simplex mode. However, we recommend that all production DB2 data sharing configurations use duplexed GBP structures.

[2]

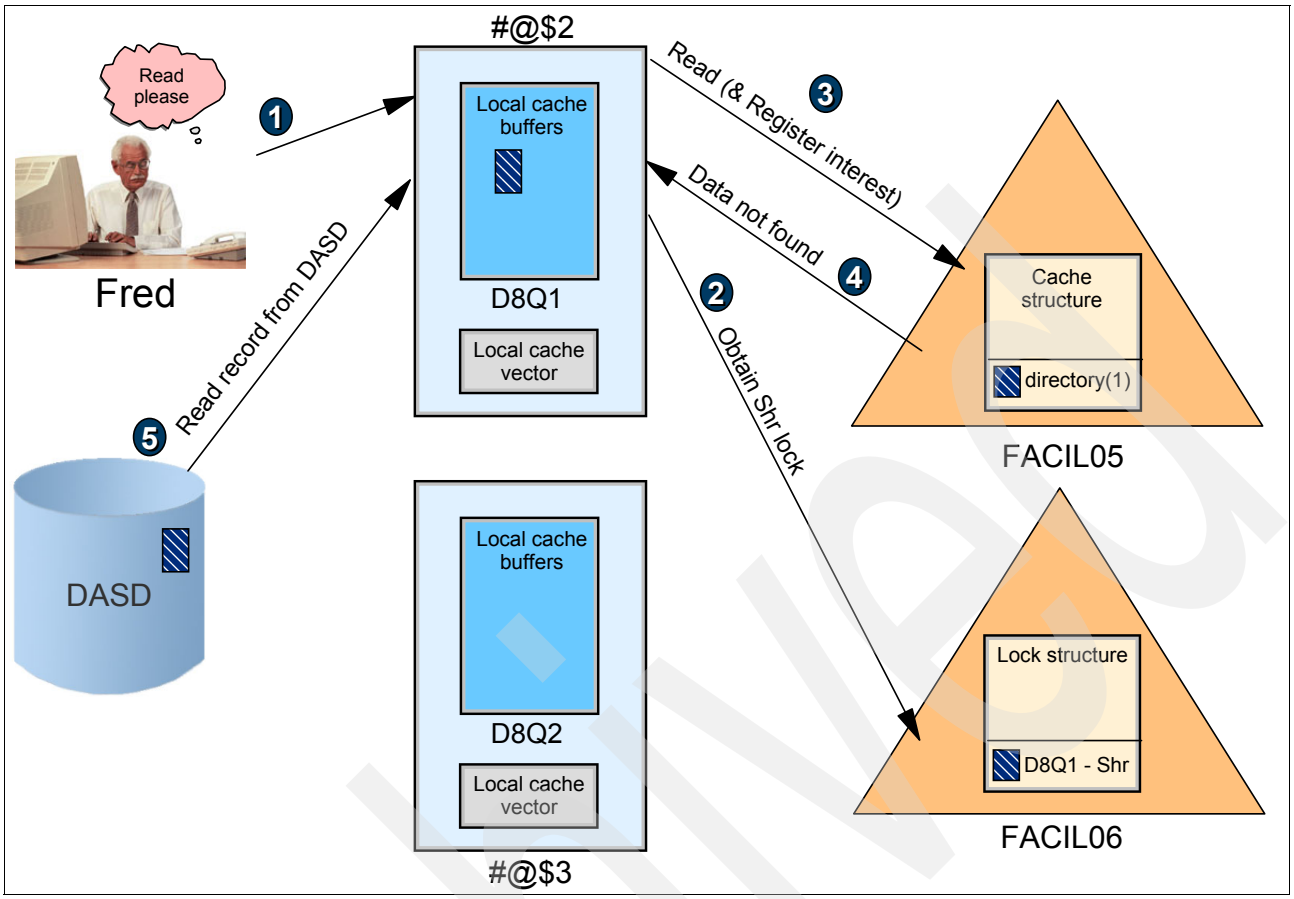


Figure 3-2 Step 1 of DB2 data sharing example

In Figure 3-2, we see that user Fred has sent a request to D8Q1 to read a row of data (1). Before progressing further, D8Q1 obtains a shared lock on this row (2) to ensure that some other process is not updating the data while Fred is using it.

When the lock has successfully been obtained, D8Q1 then tells the Group Buffer Pool structure (via a Read and Register command) that it is going to have a copy of that database page in its local buffers (3). This updates the registration information in the GBP structure so that the CF now knows which DB2 subsystems have a copy of that database page.

As part of the same request, D8Q1 asks the CF for a copy of the page it is about to read. In this case, there is no copy of that page in the GBP (4), so D8Q1 has to read the page from disk (5).

[2] For simplicity, we show GBP structures that are in simplex mode. However, we recommend that all production DB2 data sharing configurations use duplexed GBP structures.

At this point, we take a small detour to describe the workings of registration and cross-invalidation. As previously mentioned, there is a mechanism to inform DB2 if another system updates a page that is in its buffers. There are three considerations for this mechanism:

1. How does the CF know which DB2s have a copy of a database page?
2. How does the CF know when a page has been updated?
3. How does the CF inform DB2 that a copy of a database page it has in its local buffer is no longer valid, in a manner that generates minimum overhead?

In shown in step 3 of Figure 3-2 on page 30, D8Q1 issued a Read and Register request to the CF. As part of that request, D8Q1 identified the following items:

- The database page that it is going to be working on.
- The bit, in the local cache vector of the CPC that it is running on, that D8Q1 will use to represent the buffer that will contain that page.

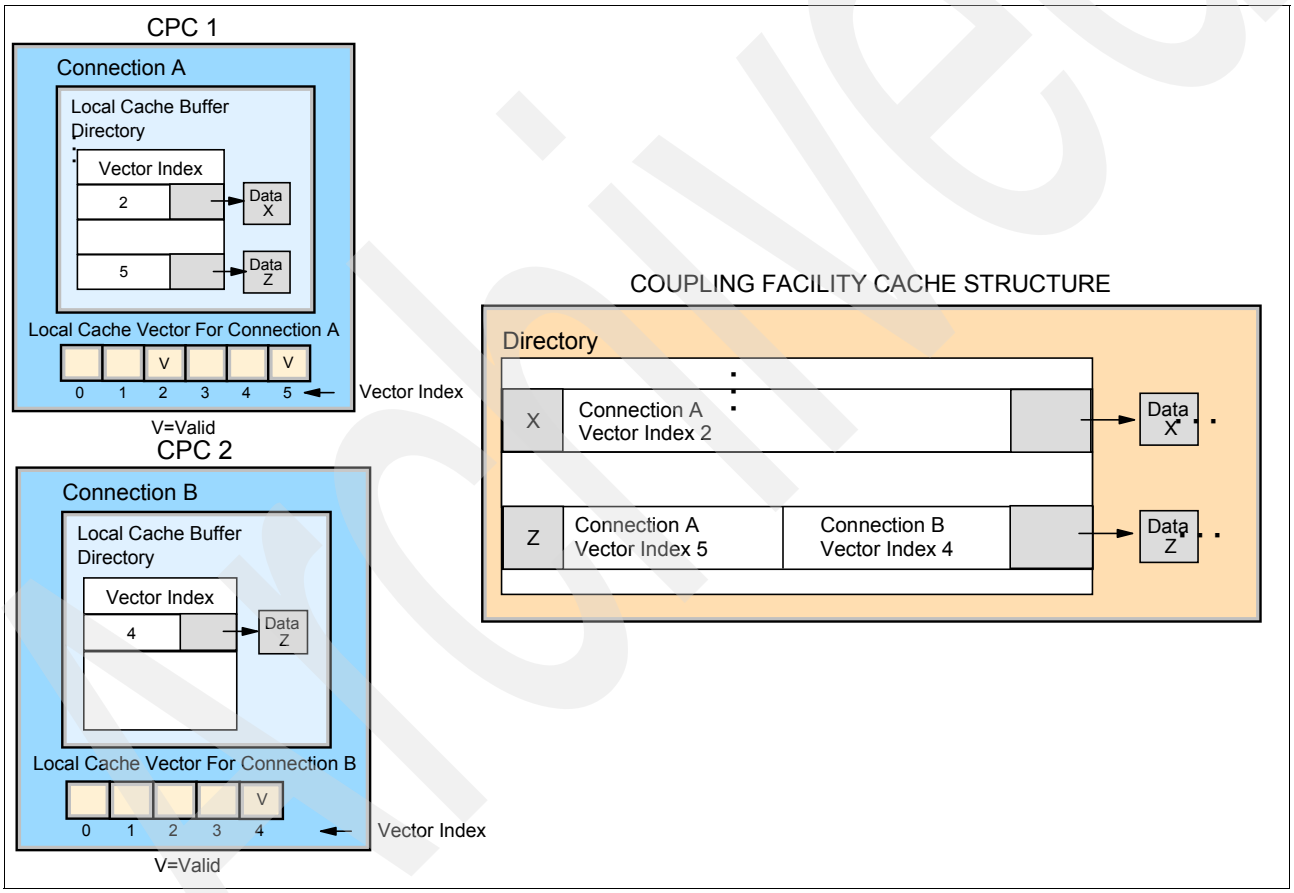


Figure 3-3 Registering use of database pages

This information is stored in the directory part of the GBP structure, as shown in Figure 3-3. In this example, you can see directory entries for two database pages: page X and page Z. You will also see DB2 in CPC1 has in-storage copies of pages X and Z, and DB2 in CPC2 has an in-storage copy of page Z. DB2 in CPC1 is using bit 2 in the local cache vector to represent page X, and bit 5 to represent page Z.

This information is saved in the directory portion of the GBP structure in the CF. DB2 in CPC2 is using bit 4 to represent page Z, and this information is also stored in the GBP structure.

This achieves the first of the three requirements: informing the CF about which pages each DB2 has a local copy of.

Now, if either of the DB2s updates page Z, that DB2 sends the updated page to the GBP structure. As part of the write request to the GBP, DB2 turns on an indicator telling the CF that this page contains changed data. This achieves the second requirement: letting the CF know that one of the DB2s has updated the data.

Using the information in the directory entries, the CF determines which other DB2 subsystems have an in-storage copy of that page, and therefore must be informed that their copy is now outdated as a result of the write request. This notification is handled by the CF sending a signal to the relevant CPC to update the bit in the local cache vector in HSA that represents the updated page. This request operates at the hardware level, and does not cause an interrupt to either DB2 or the operating system.

Every time DB2 attempts to use a page from a GBP-dependent table space from its local buffer pool, it will first check the setting of the related bit in the local cache vector to ensure the data is still valid. It is important to understand that the CF does not respond to the write request to the GBP structure until *all* of the required cross-invalidate signals have been successfully sent. This achieves the third requirement: telling the other interested DB2s that their local copies of the updated page are now invalid.

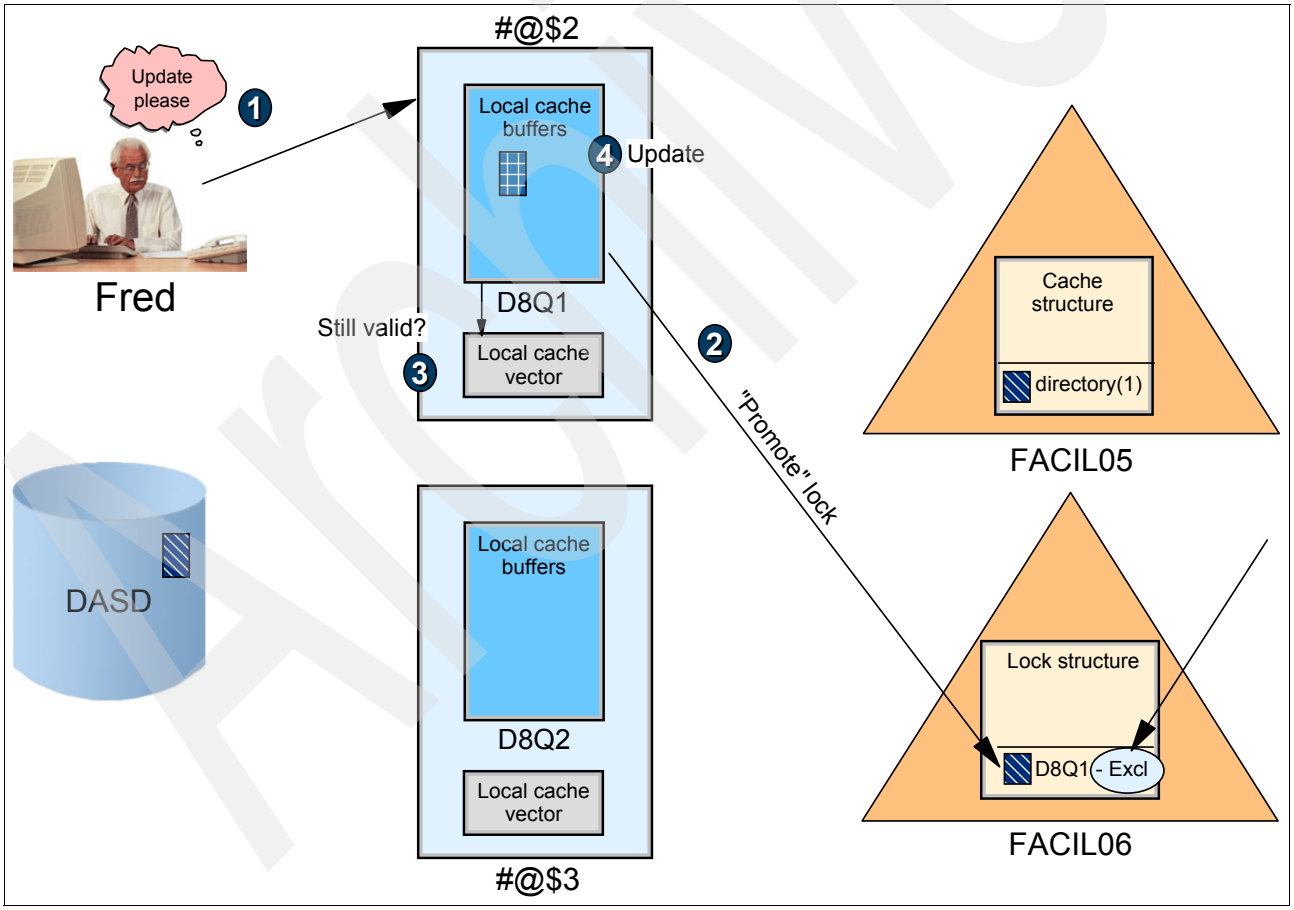


Figure 3-4 Step 2 of DB2 data sharing example

Getting back to our example, in Figure 3-4, we see that user Fred has now decided that he wants to update the row that he was looking at (1).

Before D8Q1 can proceed with the update, it must obtain an exclusive lock on that data. Whether the lock is obtained at the page or row level depends on how the table space is defined in DB2 and on what DB2 decides is the most efficient type of lock to use with this table space at this time. Because D8Q1 already has a shared lock on this piece of data, it issues an alter request to XES stating that it wishes to promote the lock to an exclusive one (2). In this case, no other DB2 is using that lock entry in the DB2 lock structure, so the promote request ends successfully.

D8Q1 knows that it has a local copy of the page to be updated. However, it needs to verify that the page has not been changed by another DB2 subsystem because D8Q1 read it in from disk. To do this, D8Q1 checks the bit in the local cache vector that represents the buffer containing the page (3). In this case, the bit is still valid, so D8Q1 performs the update (4) and writes an update record to the D8Q1 log. At this instant, we have the original level of the data still on disk, and the updated level now in the local buffer pool in D8Q1. Also, D8Q1 still holds the exclusive lock for this piece of data.

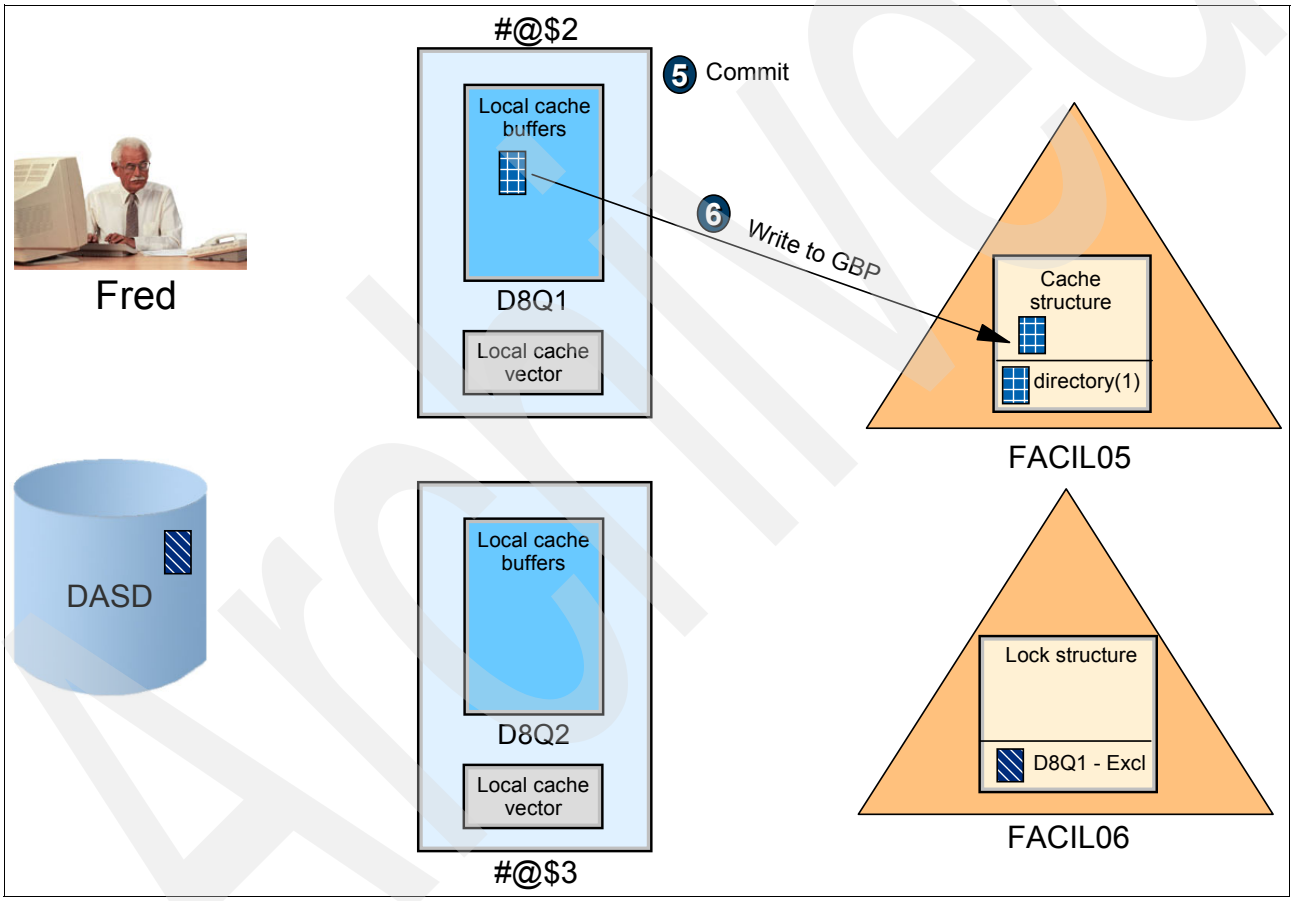


Figure 3-5 Step 3 of DB2 data sharing example

The next step, shown in Figure 3-5, is for the transaction to issue a DB2 COMMIT when it is ready to complete (5). The COMMIT causes a log record to be written to the D8Q1 log file on disk, indicating that the update that had previously been logged is now committed, and the updated page to be written to the GBP in the CF (6).

If the transaction had updated multiple pages within a single COMMIT, all those updated pages would be written to the GBP at this point (potentially with a single CF request, depending on the number of pages and how many DB2 pagesets are affected).

At this point, we have the original level of the data in the database on disk, and the updated level of the data in D8Q1's local buffer pool, in the GBP in the CF, and in the D8Q1 log file.

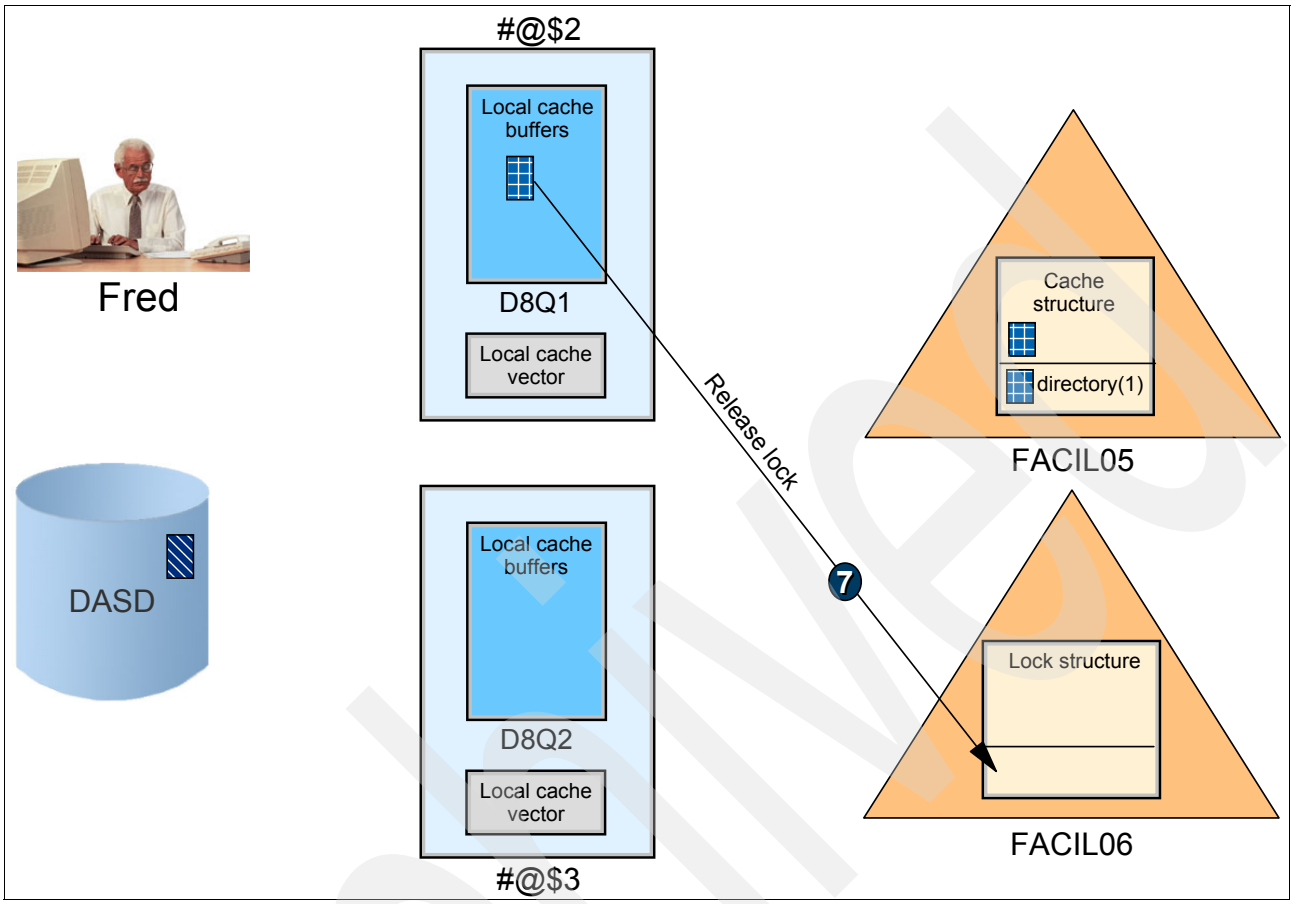


Figure 3-6 Step 4 of DB2 data sharing example

Finally, the lock held by this DB2 is released, as shown in Figure 3-6 (7).

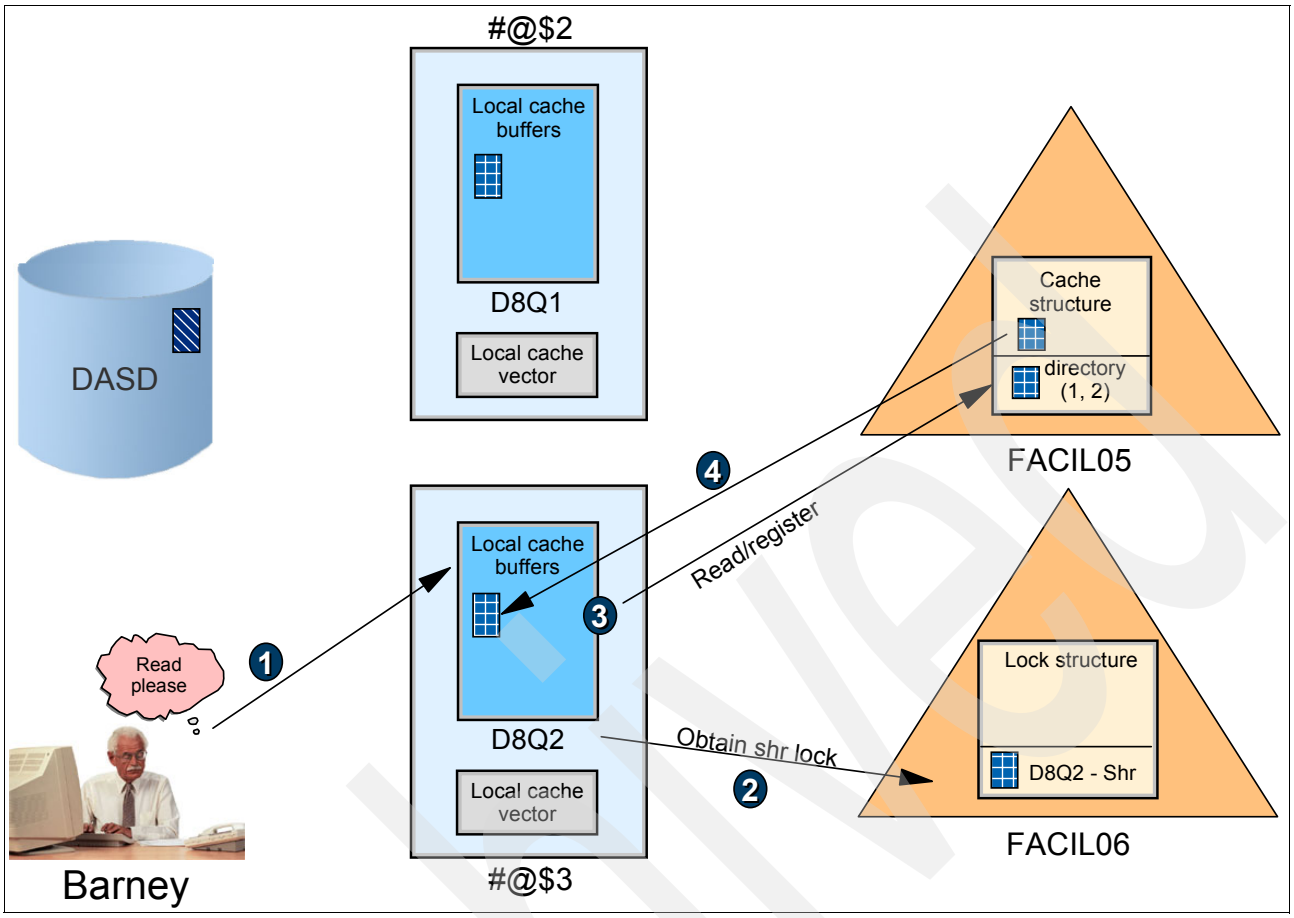


Figure 3-7 Step 5 of DB2 data sharing example

As the next step in our example, another user called Barney on system #@$3 issues a request to D8Q2, requesting to read the same piece of data that Fred just performed the COMMIT for. This is shown in Figure 3-7 (1).

As before, D8Q2 first requests a share lock on the piece of data. To do this, it issues a request to the Lock structure (2). Once again, no other DB2 has an exclusive hold on this lock entry, so the request is granted.

D8Q2 now sends a Read and Register request to the GBP structure (3), telling the CF that it is going to have a copy of this page in its local buffer pool, and passing the information about which bit in the local cache vector will represent that page. At this point, the CF knows that D8Q1 and D8Q2 are both going to have copies of this page in their local buffer pools. Also, because there is now a copy of that page in the GBP, the data is returned to D8Q2 (4).

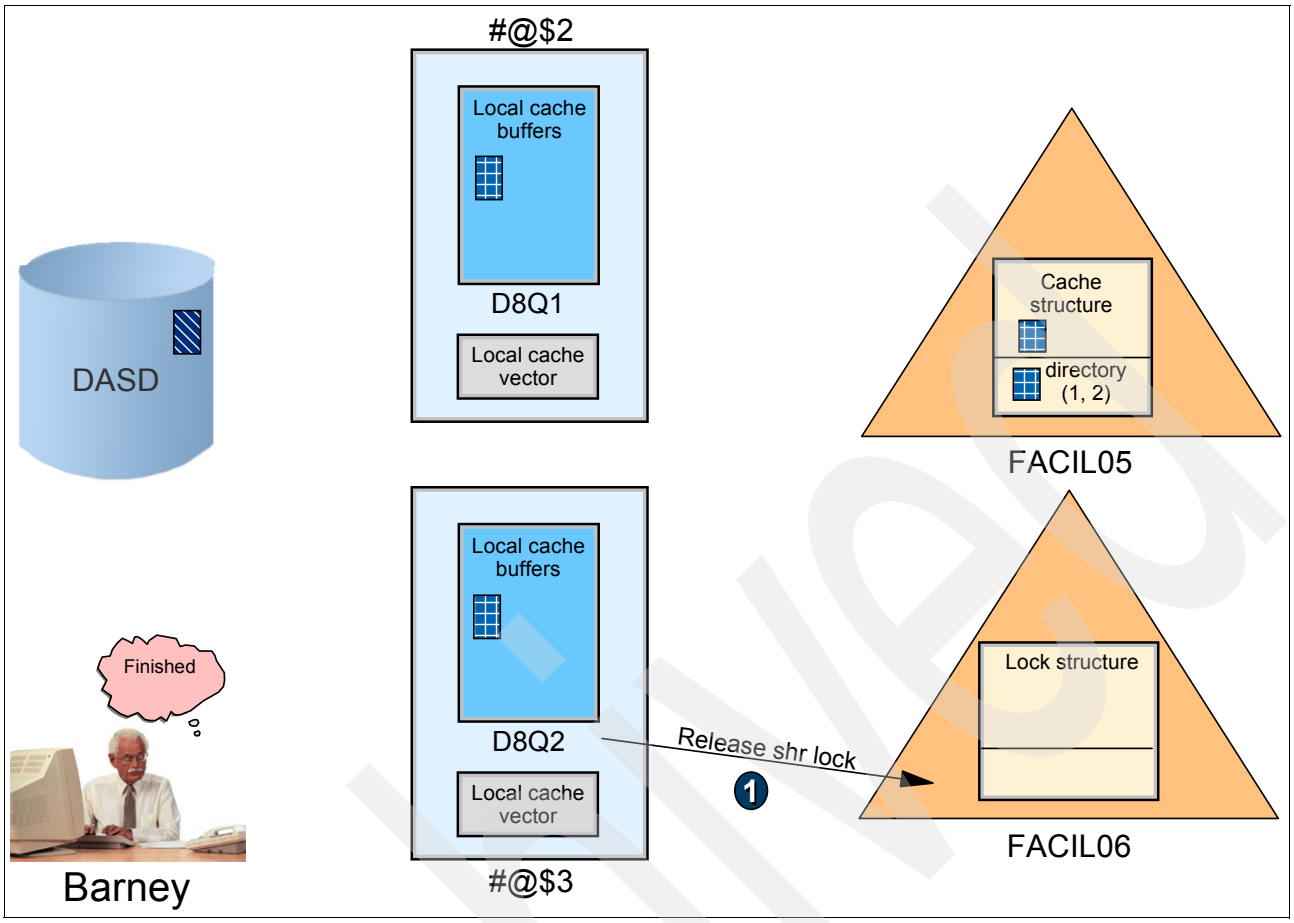


Figure 3-8 Step 6 of DB2 data sharing example

In Figure 3-8, Barney finishes looking at the data without making any changes, and ends the transaction. This results in the shared lock being released (1).

At this point, we have the original level of the page in the database on disk, and the updated level (we call it “version 2”) in the local buffer pools of D8Q1 and D8Q2, in the GBP structure, and in the log data set of D8Q1. The directory in the GBP indicates that both DB2s have an in-storage copy of the data. And no one currently has any locks on this data.

Now we are ready for the next-to-last part of this example.

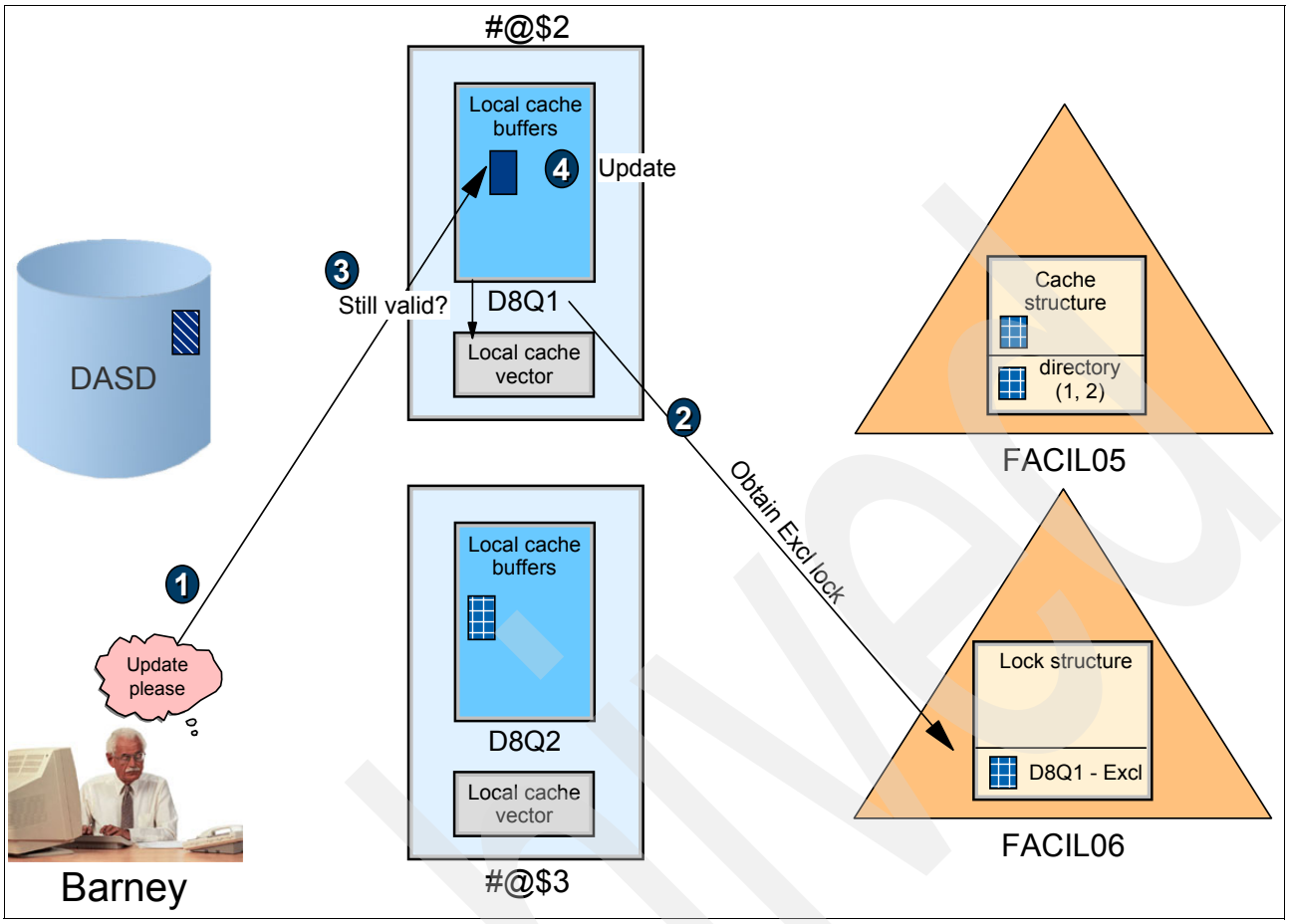


Figure 3-9 Step 7 of DB2 data sharing example

User Barney now submits a transaction to update our piece of data. Because the installation is using dynamic workload balancing, this request arrives at D8Q1 (1), as shown in Figure 3-9. Once again, before D8Q1 can change anything, it needs to obtain an exclusive lock for this piece of data (2). Because no other DB2 has an interest in this lock entry, the lock is granted.

D8Q1 is aware that it already has a copy of the data that is to be updated in its local buffers. However, before it can use that data, it must verify that the data is still valid. To do this, it checks the status of the relevant bit in the local cache vector (3). Because D8Q2 has not updated this data, the bit is still valid, meaning that D8Q1 can use the copy of the data in its buffer. Also, because the bit is still valid, D8Q1 is aware that the GBP still knows that D8Q1 has a copy of this page, so there is no need to do a Read and Register again.

Having obtained the lock and ensured the validity of the local copy of the data, D8Q1 performs the update (4) and logs the update to the D8Q1 log data set.

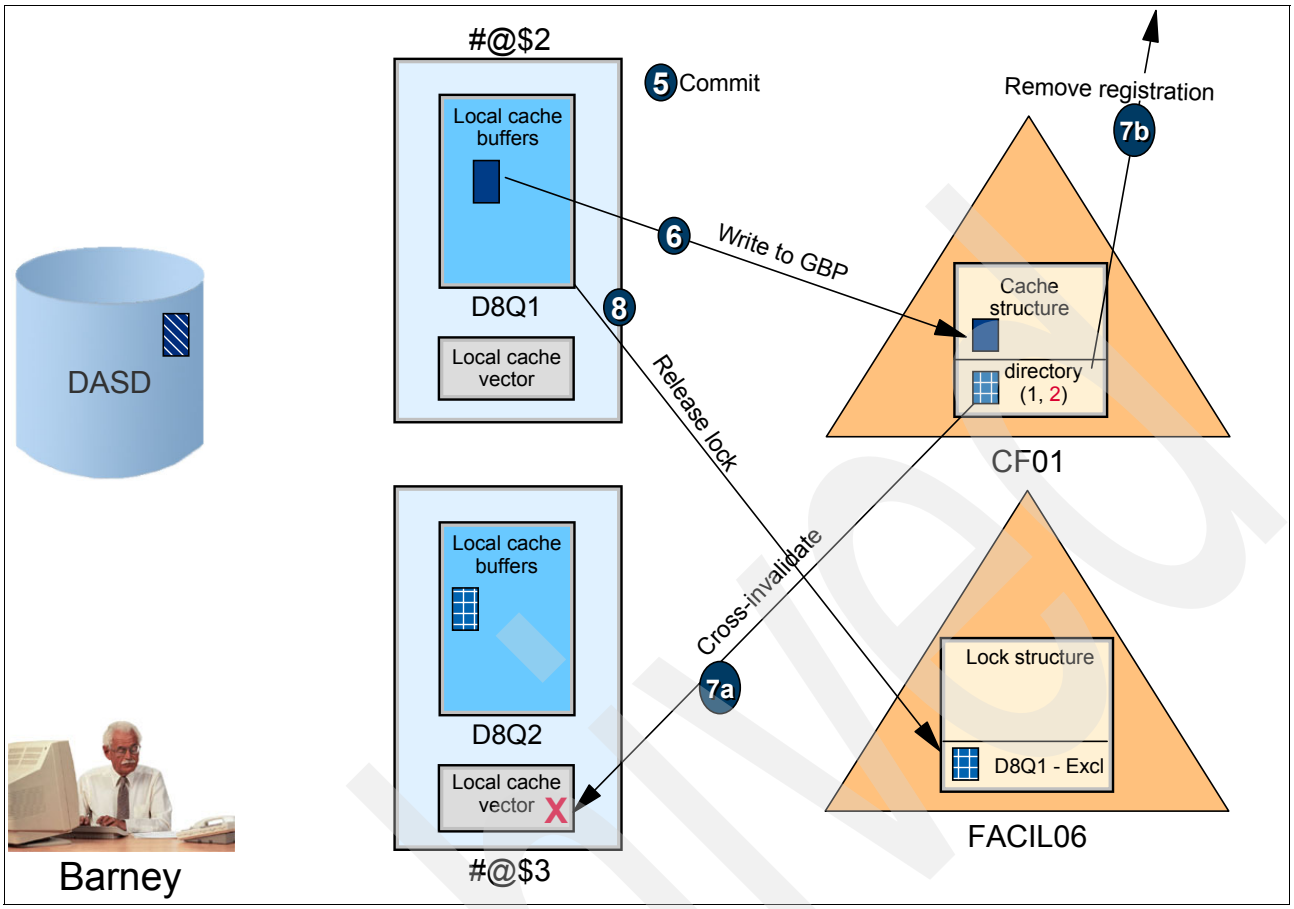


Figure 3-10 Step 8 of DB2 data sharing example

When the transaction is ready to complete, it issues a COMMIT (5). This results in an update being written to the DB2 log data set, identifying the changed data that has now been committed. Then the updated page is written to the GBP (6).

As part of handling the update, the CF checks to see if any other DB2 has an interest in the page that was just updated. In this case, it finds that D8Q2 has registered its interest in this page. This results in a cross-invalidate (7a) being sent to the processor containing D8Q2, indicating which bit in the local cache vector is to be reset. Note that the CF request from D8Q1 to send the updated page to the GBP does not complete until the CF has successfully sent all the cross-invalidate notifications. Also, because the cross-invalidate effectively means that the target DB2 subsystem (D8Q2, in this case) no longer has a copy of this page in its local buffers, the CF removes D8Q2 from the directory entry for this page (7b).

At this point, the original level of the page still resides out in the database on disk; “version 2”, which is now invalid, resides in the local buffers of D8Q2 (but has been marked invalid in D8Q2’s local cache vector); and the latest level, “version 3”, exists in the local buffers of D8Q1, in D8Q1’s log data set, and in the GBP in the CF.

Finally, D8Q1 releases the lock in the lock structure (8).

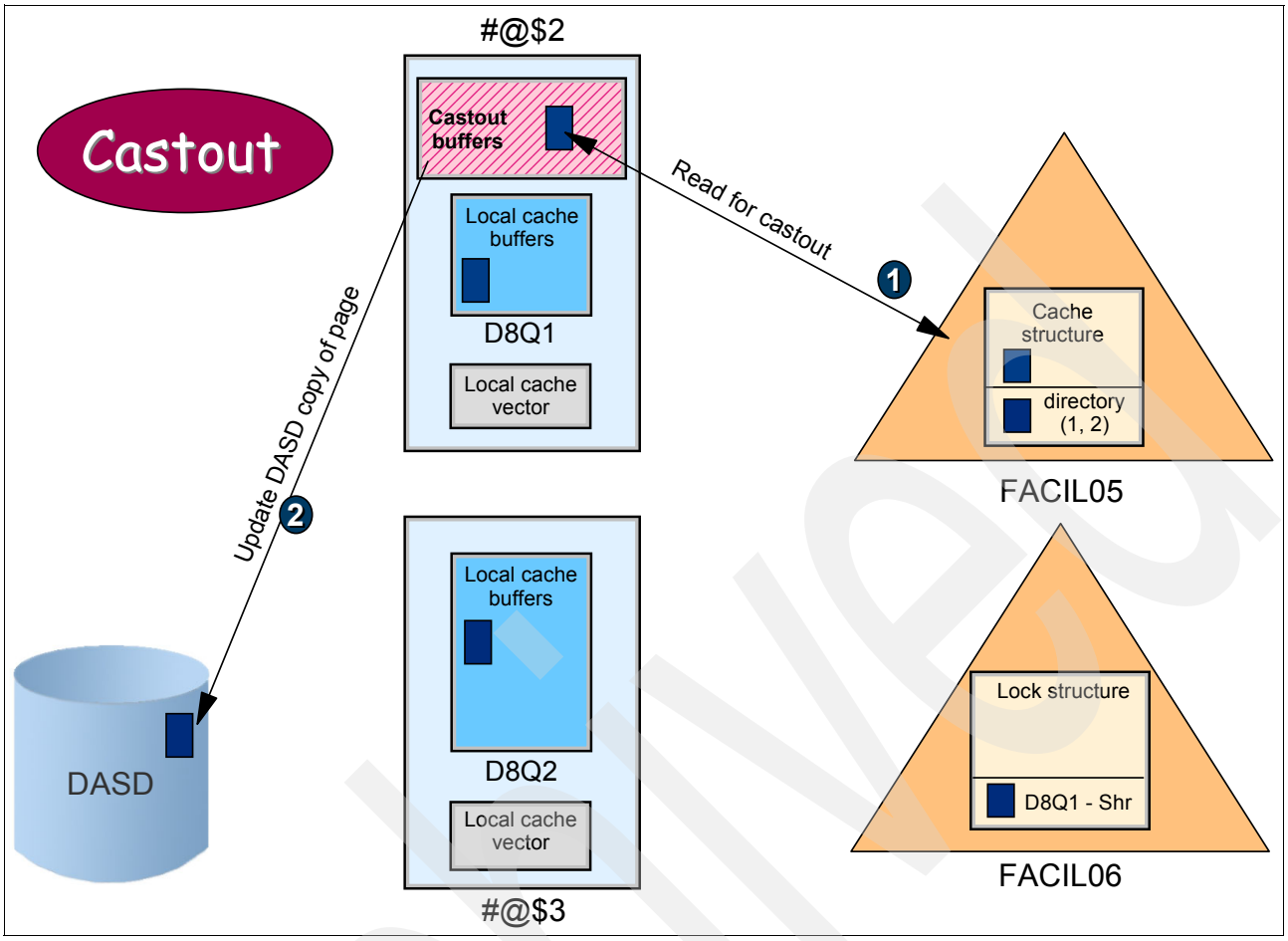


Figure 3-11 Step 9 of DB2 data sharing example

Finally, we show what happens to the updated pages that have been written to the GBP structure. When the number of changed pages in a GBP exceeds a specified threshold, or when a GBP checkpoint is taken, DB2[3] decides that it is time to harden those updated pages out to the database on disk.

To do this, DB2 issues a Read for Castout request to the GBP. The pages that should be cast out are returned to the requesting DB2, which places them in a separate set of buffers to avoid polluting its data buffers. The changed pages are then written to the databases.

The DB2 log is updated to reflect the fact that the changed pages are now on disk and the "changed" bit in the GBP is turned off for each page that now matches the corresponding page on disk. Turning off the "changed" bit tells the CF that page can now be reused to contain some other data, if necessary. The most important point is that this processing occurs *after* the transaction has written the page to the GBP, and has no impact on transaction elapsed times. In fact, it is more than likely that the transaction ended long before the page is cast out.

[3] The first DB2 to open a pageset for update becomes the "castout owner" for that pageset, and remains so until that pageset is closed by that DB2. There is no command or DB2 parameter to control which DB2 subsystem will be the castout owner for a pageset.

3.4 Related DB2 considerations

The description of DB2 data sharing provided in 3.3, “DB2 data sharing - the high altitude view” on page 29, is the greatly simplified version. There are also other parts of DB2 processing that are relevant to this discussion about cross-site data sharing, as explained in the following sections.

3.4.1 Duplexed Group Buffer Pool structures

DB2 supports a form of structure duplexing known as User-Managed Duplexing. This means that DB2 is responsible for updates to, and management of, duplex copies of its GBP structures. DB2’s implementation of User-Managed Duplexing is extremely efficient and we normally recommend that everyone performing production DB2 data sharing should duplex their GBP structures.

In cases where there is considerable distance between the CFs containing the primary and secondary GBPs, we still recommend duplexing the structures. However, it is important to understand the role that distance plays in such cases.

But first, we provide a brief overview of how DB2 GBP duplexing works. When a DB2 GBP structure is duplexed, DB2 is aware of it and is responsible for keeping both structures in sync with each other, and for recovering from the loss of either structure. To minimize the cost of keeping two copies of the structure, DB2 only duplexes information that is vital to recovery should either of the structures fail. Specifically, DB2 only sends *changed* pages to the secondary structure. Any unchanged pages are not placed in the secondary structure, even if you specified that you want to cache all pages. Also, registration information is only kept in the primary structure[4].

If the secondary structure is lost, all required information is still available in the primary structure. And if the primary structure is lost, all the pages that have not yet been cast out to disk are available in the secondary structure.

However, the registration information that is used to enable cross-invalidates if a page is updated is *not* kept in the secondary. Therefore, if the primary structure is lost, all DB2s in the data sharing group will mark all pages in their corresponding local buffer pools as invalid. This will result in a slowdown in DB2 until the buffer pools are populated again. However, no recovery will be required and there will be no interrupt to processing.

[4] This is how DB2 V7 and earlier releases, and V9 and later releases, work. DB2 V8 sometimes sends registration information to the secondary GBP. This is unlikely to be an issue except at the longest distances.

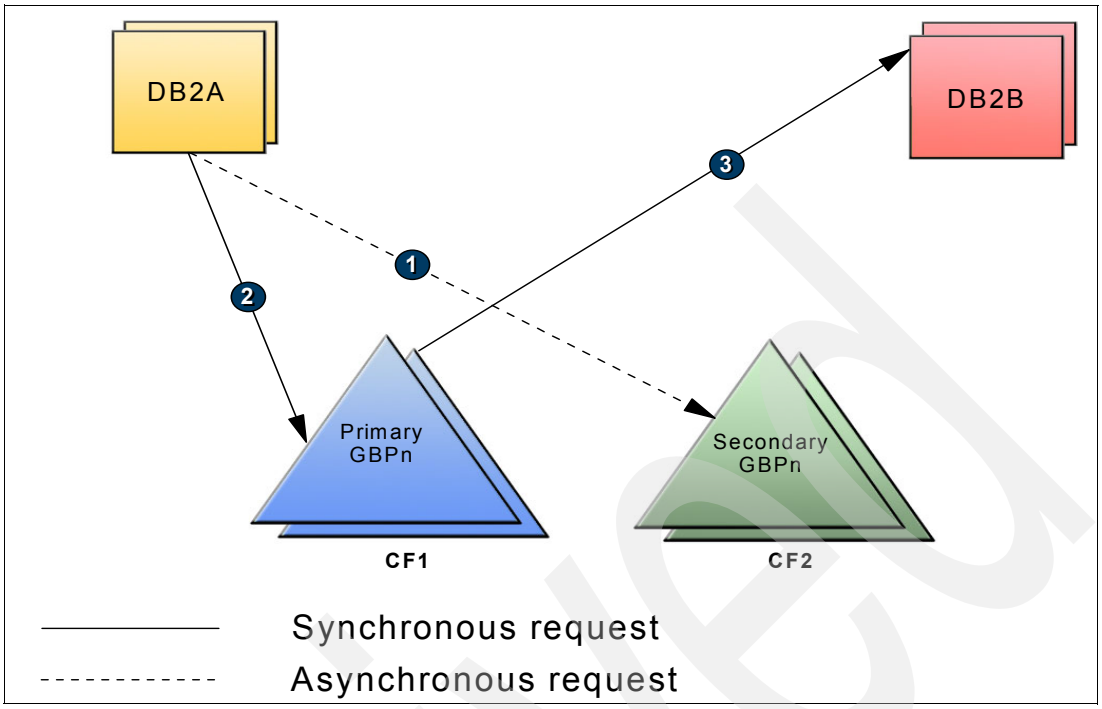


Figure 3-12 User-Managed structure duplexing

When DB2 wants to send a changed page to the GBP, and it knows that the GBP is duplexed, it first sends an asynchronous request to the secondary GBP (shown at 1, in Figure 3-12). This is followed immediately by a synchronous request (2) to the primary GBP (we discuss the differences between synchronous and asynchronous CF requests in 2.2.1, “Synchronous and asynchronous CF requests” on page 9).

This allows the two requests to run in parallel, thus shortening the time required to complete the write of the updated page to the GBP. If another DB2 has a copy of the changed page, the cross-invalidate will be sent from the primary GBP (3).

After the request to the primary GBP completes, control is returned to DB2, which then checks to see whether the request to the secondary has completed. If it has, DB2 checks to ensure that both requests completed successfully, and processing continues. If the secondary GBP request has not completed yet, the task driving that request will wait.

This time spent waiting is reported in DB2 accounting records as either “Asynch CF Request” wait time or as part of “Update Commit” wait time. Obviously, as the distance to the secondary GBP increases, this wait time will also increase, and DB2 processing for this transaction cannot proceed until the writes to both the primary and secondary GBPs have completed.

Equally, if the primary GBP is further away from DB2 than the secondary GBP, the task will still have to wait. Therefore, one way or another, when either of the GBP structure instances are distant from DB2, the writing task will see increased response times.

3.4.2 DB2 logging

Like all database managers, DB2 maintains logs of all it activities in order to be able to back out failed transactions, recover to the point of failure following an outage, and restore databases from image copies.

Numerous actions by a transaction cause a log record to be written. In some cases, such as when the transaction issues a commit, the transaction must wait until the log record has been hardened to disk. For these log writes, the distance between DB2 and the primary disks, and between the primary and secondary disks, has an impact on the transaction elapsed time (because the transaction cannot move on until the writes are complete).

Other log writes, such as some of those that occur when a transaction makes an update (but before the commit) are considered complete by the transaction as soon as the log data is stored in DB2's log buffers. In these cases, the transaction can continue running concurrently with the log data being written to disk. Unfortunately, the DB2 PE Accounting reports do not break out the different type of log writes, so it is not possible to identify how many of each type of log write is done by a given transaction.

You also need to consider the effect of dual logging data sets. IBM recommends that DB2 is run with two copies of the log data sets, to protect DB2 from a failure impacting one of the data sets. Having two copies of the log data sets is good for availability, but it makes projecting the impact of distance more complex.

Sometimes, DB2 sends a given log block to both log data sets in parallel. Other times, it is necessary to do the write to one log data set first, wait for the write to complete successfully, and then issue the write to the other log data set. The challenge is that the logic behind this is very complex, and the DB2 accounting information does not make it clear how the writes are being done.

From an MVS perspective, DB2 log writes are charged to the DB2 MSTR address space. The I/O response times are variable, because sometimes DB2 will gather a number of log blocks into a single large write request, but other times it may only have a single log block per I/O. Remember, however, that all logging I/Os, being writes, will carry the cost of mirroring the write to the secondary disk control unit, and therefore will be impacted by the distance between the sites.

3.4.3 DB2 database reads

No matter how large DB2's local or group buffers pools are, there will always come a time when DB2 requires some information that must be obtained from the database on disk.

If the transaction is trying to access directly a piece of the data that is not already available in DB2's local buffer pool, DB2 must retrieve the page before the transaction can start working on it. So, these database reads are synchronous from the transaction's perspective, because the transaction must wait until the data has been retrieved. Some of the requested records may be found in the CF, and some may have to come from the database on disk.

If the page is found in the CF, how the time to retrieve the page is reported depends on whether the CF request is sent to the CF as a synchronous request or an asynchronous request. If it is sent synchronously, then the time will be clocked up as DB2 CPU time for the thread. If it is sent asynchronously, it will appear in the ASYNCH CF REQSTS category.

If the page must be retrieved from disk, the time it takes for these I/Os to complete is reported in DB2 accounting records as a SYNCHRON DATABASE I/O. From an MVS perspective, the I/Os are charged to the user's address space (CICS, for example).

Because these are reads, there is no impact from mirroring. Therefore, DB2 subsystems in the same site as the primary disk should not be impacted by the distance between the sites. However, DB2 subsystems in the remote site will see increased response time for these reads.

If the transaction is processing the data sequentially, DB2 will attempt to read ahead of the transaction, so that when the transaction needs the data, it has already been fetched into a DB2 local buffer. This processing is called *pre-fetch*.

Normally, this processing has no impact on response times, because it will usually complete before the transaction requests the data. However, if the I/Os have not completed in time, it will be reported as a Class 3 suspension of type “other I/O wait time”. These I/Os are charged to the DB2 DBM1 address space. Obviously, as the distance between the DB2 subsystem and the primary disk increases, the time to complete these reads will also increase.

3.4.4 Levels of DB2 locking

For readers who are familiar with z/OS, serialization of resources is very simple: you either have a exclusive ENQ or a shared ENQ (think in terms of DISP=OLD and DISP=SHR). However, to maximize performance and concurrency, DB2 has a much more elaborate locking mechanism.

Before discussing the levels of DB2 locking, we must explain the hierarchy in DB2. At the highest level, there is a database. Beneath the database there are tables or table spaces. Table spaces consist of one or more tables. A table space containing a single table can optionally be partitioned. Tables consist of one or more pages. And pages may contain one of more rows.

When you want to serialize a resource in DB2, you can serialize at the table or table space level (or table space partition), at the page level, or at the row level. One of the arts of DB2 tuning is identifying the most appropriate serialization level for a given object.

If you want to minimize overhead, you might want to serialize at the table level. However, if someone else wants to access just one row in the table in a manner that is not compatible with your serialization at the table level, then one of you must wait for the other to complete.

The alternative is to serialize at the page or row level. This increases the cost, but allows many different types of concurrent access to the table. If you serialize at the page or row level, you must also get what is known as an “intent” lock at the table level.

You specify the level that you want to serialize at when you define the table to DB2, or possibly in your application program. The recommendation is to let DB2 dynamically decide on the serialization level, which it does using information about how the table is currently being used.

Table 3-1 on page 44 lists the different types of locks that can be obtained at the table or table space level, as well as the types of access that are compatible with each other. Compatible lock requests are ones that can coexist without blocking each other.

Table 3-1 DB2 table or table space serialization types

	Owner can read	Owner can update	Sharer can read	Sharer can update
IS (intent shared)	Yes (after getting row/page lock)	No	Yes (after getting row/page lock)	Yes (after getting row/page lock)
IX (intent exclusive)	Yes (after getting row/page lock)	Yes (after getting row/page lock)	Yes (after getting row/page lock)	Yes (after getting row/page lock)
S (Share)	Yes	No	Yes	No
U (Update)	Yes	Yes (after getting X lock on table)	Yes	No
X (Exclusive)	Yes	Yes	No	No
SIX (Shared intent exclusive)	Yes	Yes (after getting row/page lock)	Yes	No

The meanings of the specific lock types in relation to tables or table spaces are as follows:

IS The program requires only shared access and will be serializing at the page or row level.

IX The program will require exclusive access, but at the row or page level, not at the table or table space level.

S The program requires only share access, and serialization is only at the table or table space level.

U The program is serializing at the table or table space level and indicates that at some point in its execution, it may do an update. However, the level of serialization will be increased before the update is attempted.

X The program is serializing at the table or table space level and is ready to do an update, so it requires exclusive access to the table.

SIX The program will be able to read and change data in the table or table space. Concurrent processes can read the data in the table or table space, but not change it. This differs from an IX lock, where concurrent processes can read or change data in the table or table space. When the lock owner is ready to change some data, the X-lock is acquired on the corresponding page or row.

You can see that intent locks (meaning that you will be serializing at the page or row level) are far more flexible. For example, many different users could hold IX locks on the table concurrently, with the exclusive only locks being held on the pages or rows that are being updated.

Table 3-2 on page 45 lists the types of serialization that can be obtained at the page or row level, and what other type of access is compatible with each one.

Table 3-2 DB2 page/row serialization types

	Owner can read	Owner can update	Sharer can read	Sharer can update	Table/Table space lock
S (Share)	Yes	No	Yes	Can get S or U lock	IS or IX
U (Update)	Yes	Yes (after getting X lock on row/page)	Yes (but not concurrent with X lock)	No	IX
X (Exclusive)	Yes	Yes	No (1)	No	IX or SIX

This information is important to understanding the results we obtained in our CICS/DB2 workload measurement. These locks would be held concurrently with the table or table space locks indicated in the right-most column of the table.

3.4.5 Lock contention

In 3.3, “DB2 data sharing - the high altitude view” on page 29, every time either of the DB2s in the examples requested a lock on the piece of data, the lock was available (that is, no one else had serialized the lock when we tried to get it). While this is a pleasant outcome, it is probably not representative of a typical production environment.

In 3.4.4, “Levels of DB2 locking” on page 43, we introduced the idea of compatible lock requests.

In this section, we briefly discuss what happens when there are two *in*compatible requests for the same lock entry; that is, when you have “lock contention”.

When you add distance to an existing data sharing configuration, it is likely that transactions and batch jobs will see an increase in elapsed times. When these jobs and transactions run for longer, it should be expected that they will also hold onto any resources they have serialized for longer. This may result in increased contention, as new work is arriving and trying to serialize resources that are still held by already-running work.

Whether more contention is actually created, and to what extent, is completely workload-dependent. If you have very few transactions (such as in a test system), then there is less likelihood that the transactions will be competing for the same resources. Similarly, if the transactions are doing direct updates to a table with millions of rows, there is less chance that two transactions will be fighting over the same row, especially if there are very few transactions. However, if the number of transactions increases greatly, if large batch jobs are running concurrently, if the transactions run for much longer, or if the table is much smaller, the opportunity for encountering contention will increase.

The question is, what impact does more contention have? One obvious impact is that the work unit encountering contention will run for longer, waiting for the current holder of the resource to get out of the way. However, another impact is increased work for the system when contention is encountered. In 2.5.1, “XCF/XES role as global lock manager” on page 20, we describe the additional work that XES and the resource owner (IRLM, in the case of DB2) have to do when lock contention is encountered.

Thus, increased contention is a likely secondary effect of distance, and increased contention will drive up elapsed times and also increase CPU consumption, because both XES and IRLM have more work to do.

3.4.6 Lock avoidance

What is the most efficient type of lock request? The one that is never issued! Based on this realization, DB2 Version 3 introduced a function known as lock avoidance.

Depending on the options you use when you bind your application in DB2, it is possible to tell DB2 that data returned to your application must be committed data, but that it is acceptable if an update is made to the data immediately after your application has read it (that is, you do not need to lock the data while you are using it). If your application is bound with the appropriate options, DB2 can determine, from information in the page and in the row that it is looking at, whether there is a possibility that an update has been made to the page, or is in the process of being made. This means that the application must wait to get a lock on the page before it can use the data.

Each DB2 page has appended to it the time stamp of when the page was last updated (this is known as the “Page RBA”). DB2 is also aware of the time stamp of the oldest unit of work in the data sharing group that has not yet been committed. If the page RBA is older than the oldest uncommitted unit of work, then DB2 knows that page contains committed data and does not have to issue a lock request to serialize the data.

But what happens if the page contains multiple rows, and the row that you are interested in was committed a long time ago, but the page RBA is newer than the oldest uncommitted log record? To address that situation, DB2 maintains an “invisible” bit in each row called the “Possibly UNChanged bit”, or PUNC bit. When the row is updated, the PUNC bit is turned on. As a result, as long as the PUNC bit is off, DB2 knows that the row contains committed data, even if the page RBA is newer than the oldest uncommitted unit or work. In this case, DB2 once again does not have to issue a lock request to serialize the data.

However, if the PUNC bit is still turned on, and the page RBA is newer than the oldest uncommitted unit of work, then DB2 must issue a lock to ensure no one is in the middle of updating the requested piece of data.

The PUNC bit is not turned off as soon as the commit is complete; instead, it gets turned off after a number of criteria are met.

To achieve the maximum benefit from lock avoidance, it is important that units of work do not run for a long time, because this increases the likelihood that the page RBA will be newer than the oldest uncommitted unit of work.

For more information about DB2 lock avoidance, refer to *Locking in DB2 for MVS/ESA Environment*, SG24-4725, and *DB2 for z/OS Performance Monitoring and Tuning Guide*, SC18-9851.

3.5 Life of a data sharing DB2 transaction

In this section we use all the information provided thus far in this chapter to describe the life of a DB2 transaction, focusing on the events in that life that cause the transaction to be delayed.

If we look at what makes up the elapsed time of a DB2 transaction, it consists mainly of:

- Time using the CPU, or waiting to be dispatched on the CPU
- Time talking to the CF, or waiting for a response from the CF
- Time waiting for a disk I/O request to complete
- Time waiting for a resource held by the same or another DB2

Now we quickly trace the life of a DB2 data sharing transaction and then see how distance will affect each of these components. This discussion assumes that the table space is already GBP-dependent (that is, that there is inter-DB2 read/write interest in the table space), and that the resources required by the transaction are not being used elsewhere (that is, no lock contention is encountered).

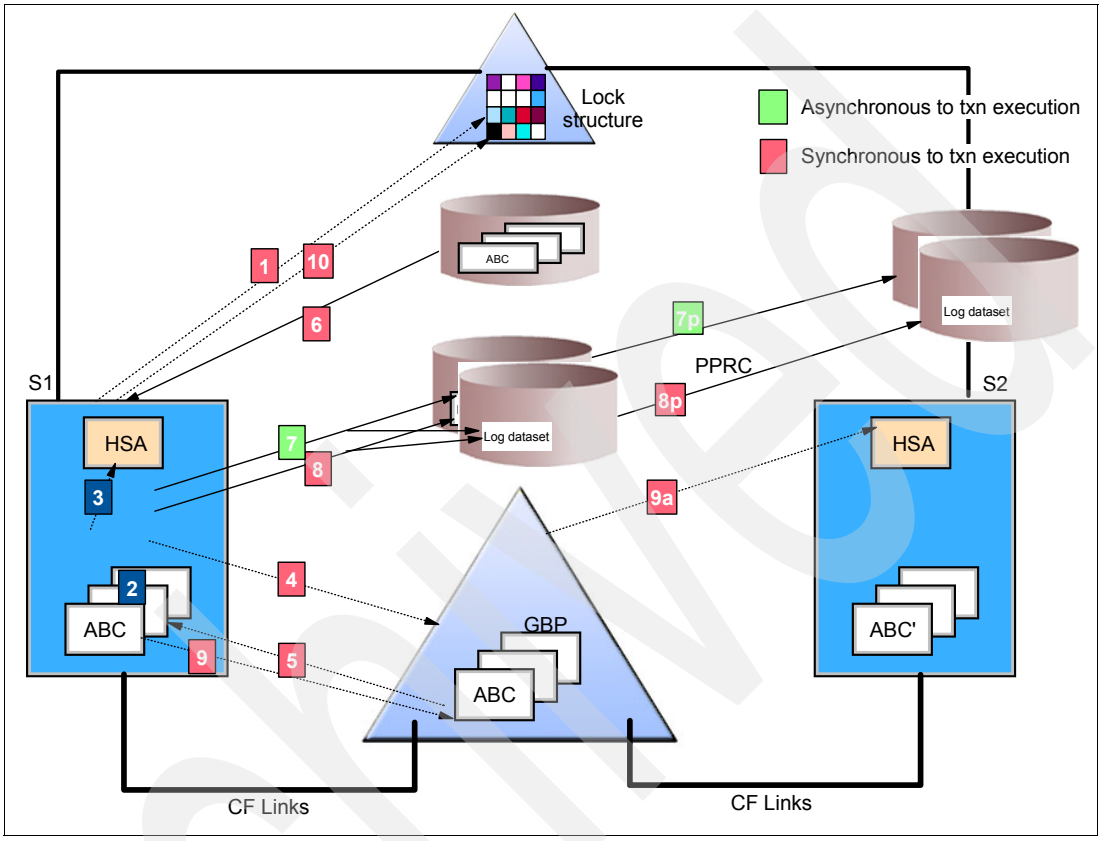


Figure 3-13 Life of a DB2 data sharing transaction

Tip: This discussion is based on the example shown in Figure 3-13. If you are able to view the picture in color, the numbered red boxes are events where the transaction must wait for successful completion. The green boxes are events that can run in parallel with the transaction continuing its execution.

When a request arrives in DB2 asking for a particular piece of data, DB2 first has to obtain a lock on the data item to ensure it is not currently serialized by anyone else (1). The DB2 thread cannot proceed until the requested level of serialization has been granted.

If another thread already has serialization on the requested resource, this thread will be suspended until that other thread gets out of the way. The longer the request takes (either because someone else has an incompatible lock on the resource, or because it takes the CF a long time to respond), the longer the task will be suspended for.

After the lock is granted, DB2 checks to see if it has a valid copy of the requested data in one of its local buffers (2). It verifies the buffer validity by checking the local cache vector in HSA (3). If it does have a valid copy, it will use it. This processing all takes place within storage, so is unaffected by distance.

If a valid copy of the requested page is not available in the local buffers, DB2 sends a request to the primary GBP structure, informing it about which bit in the local cache vector it is going to use to represent this piece of data, and requesting a copy if there is a valid copy in the GBP (4).

At this point, the thread is suspended, waiting to get the item of data it needs to start execution. If the GBP contains a valid copy, it registers the interest of this DB2 subsystem and returns the data (5). Otherwise, it still registers interest, but responds that it does not have a copy of the data.

Note the following points:

- The best case scenario is that a valid copy of the requested data is found in the local buffer pool. In that case, the thread only has to wait for a tiny amount of time before the data is made available to it.
- If there is no valid local copy, DB2 must register its interest with the primary GBP structure. So the amount of time the thread is suspended depends on how long it takes the CF to respond. If the structure is in the same site as the DB2 subsystem, the response should be short. If it is in the site remote from the subsystem, the thread will be suspended for longer (depending on the distance to the CF).

  Given that there is no local copy, and that DB2 had to go out to the CF to register its interest anyway, the next best case is that a valid copy of the data is found in the GBP.
- The least favored scenario is that the requested page is not found in the CF either. DB2 had to go to the CF anyway, to register its interest, so it incurred that time cost, but now it finds that it *also* has to go to disk to get the data.

In the case where there is not a copy of the requested data in the GBP, DB2 performs a synchronous read to get the date from the database on primary disk (6). Because this is a read, there is no impact from the disk being mirrored. If the DB2 subsystem is in the same site as the primary disk, the response time should be short. If it is in the remote site, you would expect to see an increase in the response time for these reads.

Tip: Although effective buffer pool utilization and high hit rates are always important to performance, they become even more so in a long-distance sysplex because of the higher time impact incurred every time DB2 has to go outside the processor for something.

As the thread proceeds, it may request more data, and similar actions will take place for each requested page.

As the transaction updates data (but prior to the commit), log records are written to the primary log data sets (7) and mirrored to the secondary data sets (7p). The rules determining whether these writes are synchronous to the elapsed time of the transaction are complex.

If the write is synchronous, the response time will depend on the location of the DB2 subsystem in relation to the primary disks and the distance between the primary and secondary disks (because presumably the writes are being mirrored). Also, there is the question of whether DB2 writes the log block to the dual log data sets in parallel or serially.

Also, if the DB2 is in the same site as the secondary disk, it suffers a double impact; the cost of communicating across the distance to the primary disk in the other site, *plus* the cost of mirroring the write back to the secondary site. If DB2 can process the write asynchronously to the transaction execution, the elapsed time for these writes should not delay the thread.

Eventually, the thread finishes whatever it is doing and issues a commit, indicating that it has completed successfully and wants to save all its changes. This results in the commit log

records being written to the DB2 log data sets (8 and 8p), the updated pages being written to the GBP structures in the CF (9), and unlock requests being sent to the lock structure, releasing any serialization it held (10).

The writes of the commit log records are always synchronous to thread execution; the thread cannot move on to the next instruction until we are sure that the commit log records have been safely recorded on disk. Because the thread needs to wait for the write(s) to disk to complete, the distance between DB2 and the primary disk, and between the primary and secondary disk will impact thread elapsed times.

In the example in Figure 3-13 on page 47, the other DB2 system also has an in-storage copy of the data that has just been changed. Therefore, the elapsed time of the request to write the updated pages to the GBP (9) includes the time required for the CF to send a cross-invalidate to the HSA of the second CPC (9a). The write requests to the primary and secondary GBP structures run in parallel, so the suspend time seen by the thread is determined by the response time of the longer of the two requests.

For this reason, threads will see an impact of the distance, regardless of which site they run in. If they are in the same site as the primary GBP, the request to the secondary structure (in the remote site) will presumably take longer, so the thread will need to wait for that to complete. If the DB2 is in the same site as the secondary GBP, the request to the primary GBP will probably take longer, so the thread will need to wait for that to complete.

In fact, for the DB2s that are in the remote site to the primary GBP, there could be a double impact from the distance. Not only will it take a relatively long time for the request to reach the CF, but the CF will not respond until it has successfully sent the cross-invalidate notifications to any other DB2s that have a copy of the changed page. And some of those could be back in the same site as the requesting DB2, meaning that another 10 microseconds per kilometer will be spent sending that notification.

For this reason, at times of heavy update activity (during the batch window, for example), it would be wise to try to run the batch jobs in the same site as the primary GBP, and minimize the amount of inter-DB2 R/W interest with DB2s in the remote site (for example, by running all the work with an interest in a given table space in the same site).

And finally, DB2 has to release the locks held by this thread. The transaction cannot proceed until the update to the lock structure to release the locks has completed. (There are also other tasks that must be completed at this time, but they all run in the processor and therefore are unaffected by any distance in the configuration).

There are a number of interesting points in this discussion:

- The majority of “external” requests during the life of this transaction were to the CF rather than to disk. And, as pointed out in 2.2, “Coupling Facility” on page 8, the relative impact of distance is much higher on CF response times than it is on disk response times.
- Using this description and a DB2 Performance Expert report (and significant effort), it should be possible to reasonably accurately predict the *primary* impact of the additional distance on transaction elapsed times.
- What is not possible to predict or model are the secondary effects.

Returning to the four main components of elapsed time for a DB2 transaction, we have communication with disk, communication with the CFs, using or waiting for CPU, and waiting for DB2 resources. It is this last component, waiting for resources, that makes it so difficult to model the impact of distance.

If we assume that every “external” request will take an extra 10 microseconds per kilometer, and assume that each transaction makes 50 external requests, then this indicates that each transaction will take roughly .5 milliseconds longer per kilometer. However, if each transaction is taking longer to complete, that means it is holding onto the resources (including DB2 threads) it is using for longer, and this then impacts any other transaction trying to use those resources. It is not possible to model this impact. However, it is fair to say that you would expect to see a correlation between the degree of real contention and how severely transactions will be impacted by this additional wait.

All of this discussion has focused on the impact from the perspective of transaction response times. However, we also need to look at “the bigger picture”; that is, what is happening at the subsystem and system level.

Assuming a given transaction rate, and assuming that each transaction will continue to use a similar amount of CPU and have a similar number of “external” requests, then the result of longer transaction elapsed times will be:

- More storage usage, as each transaction remains in DB2 and the owning transaction manager for longer.
- Higher utilization of DB2 threads.
- Assuming that the increased elapsed times result in higher lock contention, you would expect to see more CPU time being consumed by IRLM and XCF, and more XCF messages in the groups associated with the DB2 lock structure.

Other considerations

In addition to the implications for individual transactions and batch jobs, there are events that can take place in DB2 that are somewhat asynchronous to a given piece of work.

One example of such an event is when a DB2 pageset becomes GBP-dependent. When this happens, the DB2 that was previously using the pageset must register all the pages that it has an in-storage copy of with the CF. It must also move a copy of all updated-but-not-yet-written-to-disk pages into the GBP, and it must promote any locks that were held locally into the Coupling Facility.

The frequency with which this happens is related to the PCLOSET and PCLOSEN parameters, which specify how quickly DB2 should convert a pageset to read-only after the last update was made. If all the systems and structures are located close together, there may be performance advantages to moving pagesets out of being GBP-dependent as quickly as possible. However, if the structures are located at a significant distance from some of the connected systems, the intermittent overhead that is incurred when a pageset becomes GBP-dependent again may offset any advantages of removing DB2 interest in a pageset as quickly as possible.

Other situations that warrant consideration are when recovery from some failure is required. If the failure is limited to a DB2 subsystem, or to the z/OS image it was running in, the distance should not introduce a significant change in elapsed times. No additional accesses to the GBP structures should be required, and the recovery actions for the lock structure mainly consist of commands that run entirely within the CF.

If the failure was of a Coupling Facility, or a CF Link, the impact will depend on the structure that was affected. If the structure is a GBP structure that was duplexed (in line with IBM recommendations), then no data movement is required to revert the structure back to simplex mode in the remaining CF.

However, such recovery situations can be time-consuming because every system in the sysplex queues up for access to the CFRM Couple Data Set. If half of the systems are remote

to that data set, the increased access times could increase contention on the CDS, resulting in elongated recovery times.

To address this situation, z/OS 1.8 introduced a new Message-Based protocol for accessing the CFRM CDS. Tests in IBM indicate significant reductions in recovery times, and you would expect those results to be even greater when some of the systems were experiencing long I/O times when accessing this data set. Note that this function is not rolled back to earlier releases, and cannot be used until all systems in the sysplex move to this release or to a later release.

4

CICS/DB2 workload and metrics

This chapter discusses the workload used for the DB2 measurements and the metrics and reporting tools we used to describe the results.

4.1 Workload description

The CICS/DB2 workload we used was based on a set of transactions developed following the specifications of the Transaction Processing Council “C” set of benchmarks. It consisted of a number of CICS transactions performing a variety of calls to DB2.

Important: Although the workload mix we used is modeled after the TPC-C benchmark, the data presented in this book is not intended to provide general information about system performance, but only hints about the effect of distance on them.

The workload we used was comprised of six different online transactions. One of them (called TPNO) had many DB2 calls, others had few calls, and one (TPDF) had no calls. The percent of the workload by transaction type, together with the average number of DB2 reads and writes for each transaction, are listed in Table 4-1.

Table 4-1 CICS/DB2 transactions

Txn name	% of total txns	Avg DB2 updates/inserts per txn	Avg DB2 fetches and selects per txn
TPCI	20.1%	0	1
TPDF	34.8%	0	0
TPNO	35.0%	23	22 fetch + 1 select
TPOS	3.5%	1	13 fetch + 1 select
TPPA	3.4%	4	4
TPSL	3.2%	0	2

We report on the performance of the TPNO transaction in detail, because of its high level of interaction with DB2. We also provide summary level information about the complete mix of transactions.

Additionally, we report on the TPOS transaction. Even though the TPOS transaction used some of the same tables as TPNO, it did not encounter the same contention and therefore reacted very differently to the distance increases than did TPNO. We specifically wanted a mix of transaction profiles to stress how distance impacts different applications in different ways, and we believe these two transactions meet this requirement.

Important: We cannot stress enough that the impact of distance varies from one application to another, and even from one transaction to another. Even given the same hardware, software, connectivity, and data, the effect on applications will vary depending not only on how update-intensive they are and how well-tuned the applications and databases are, but also (especially) on how prone they are to contention.

In this book we provide the results of a number of measurements. However, the only definitive way to determine the impact distance will have on *your* installation is to install the DWDMs and dark fiber and measure how it affects your production environment.

At this point, we provide information about the logic of the TPNO transaction. Although this may seem superfluous now, as the distance in the measurements increases, you will see that this information is actually quite important.

The TPNO transaction uses a number of DB2 tables. The table names, the type of access that TPNO uses, the number of rows in each table, and whether the table has an index or not are listed in Table 4-2.

Table 4-2 Tables used by TPNO transaction

Table name	Access type	Number of rows	Indexed
DISTRICT	Update	20	No
WAREHOUSE	Read	2	No
ITEM	Read	100000	Yes
STOCK	Update	200000	Yes
NEWORDERS	Update	18000	Yes
ORDERS	Update	60000	Yes
ORDERLINE	Update	600000	Yes

TPNO performs the following processing:

- TPNO reads the Customer and the Warehouse tables for the inputting terminal (the terminal name determines the warehouse). Any table that does not have an index will need to be scanned sequentially.
  - TPNO scans the Warehouse table looking for the warehouse (because there is no index on this table). Because it is only reading the row, TPNO gets an S-Lock on the required warehouse (remember that there are only 2 rows in the Warehouse table). The S-lock does not contend with another S-Lock or a U-Lock, but it can contend with an X-lock if another transaction were to issue one.
- TPNO updates the District table.
  - TPNO scans the table looking for the district to be updated (because there is no index on this table either). TPNO attempts to exploit lock avoidance; if it is unsuccessful, it has to get a U-Lock on each row (remember that the District table is also small, only containing 20 rows). These row-level U-Locks can conflict with another U or X-Lock.
  - When the desired district is found, the U-Lock is changed to an X-Lock and the lock is held until the Commit; the X-Lock conflicts with any other request for that district.
- TPNO inserts a row into the New Order table.
- TPNO inserts a row into the Order table.
- TPNO inserts an average of 10 rows into the Orderline table.
- For each Orderline, TPNO reads the Item table and updates the stock table.

To understand how the different types of lock interact with each other, refer to 3.4.4, “Levels of DB2 locking” on page 43.

4.1.1 Coupling Facility structures

The DB2 transactions in our workload accessed tablespaces and indexes associated with a number of Group Buffer Pool structures. In line with the IBM recommendation to duplex all production Group Buffer Pool structures, all the GBP structures were duplexed.

Although a normal configuration would place some primary GBP structures in one CF and the remaining primary GBP structures in the other CF, we deliberately placed all the primary GBP structures in the CF in Site 1. This enabled us to more clearly see the impact of having the

primary structures at a distance from one of the DB2 subsystems. The DB2 Lock structure and the SCA structure were also placed in that CF. The placement of the DB2 structures is shown in Table 4-3.

Table 4-3 Structure placement

FACIL05 (Site 1)	FACIL06 (Site 2)
Lock structure	
SCA structure	
All Primary GBP structures	All Secondary GBP structures

Castout owner

We also engineered the workload so that the DB2 subsystem running in Site 1 was *always* the castout owner for all GBPs. The reason for this was to reduce the number of variables from one measurement run to the next.

In a production environment, it would not be possible to control the castout ownership in this way. However, because the castout process is asynchronous to the transaction execution, the location of the castout owner should not make a material difference to our overall results.

4.2 Components of DB2 transaction response time

The elapsed time for a transaction accessing DB2 data consists mainly of:

- Time spent consuming CPU, either while processing SQL code or other application code
- Time spent waiting for some DB2-related event (such as writing a log record or reading a page from the database on DASD) to complete
- Time spent waiting for non-DB2 events to complete, such as reads or writes to a non-DB2 file

DB2 categorizes each of these components into one of three “classes”, as shown in Figure 4-1 on page 57.

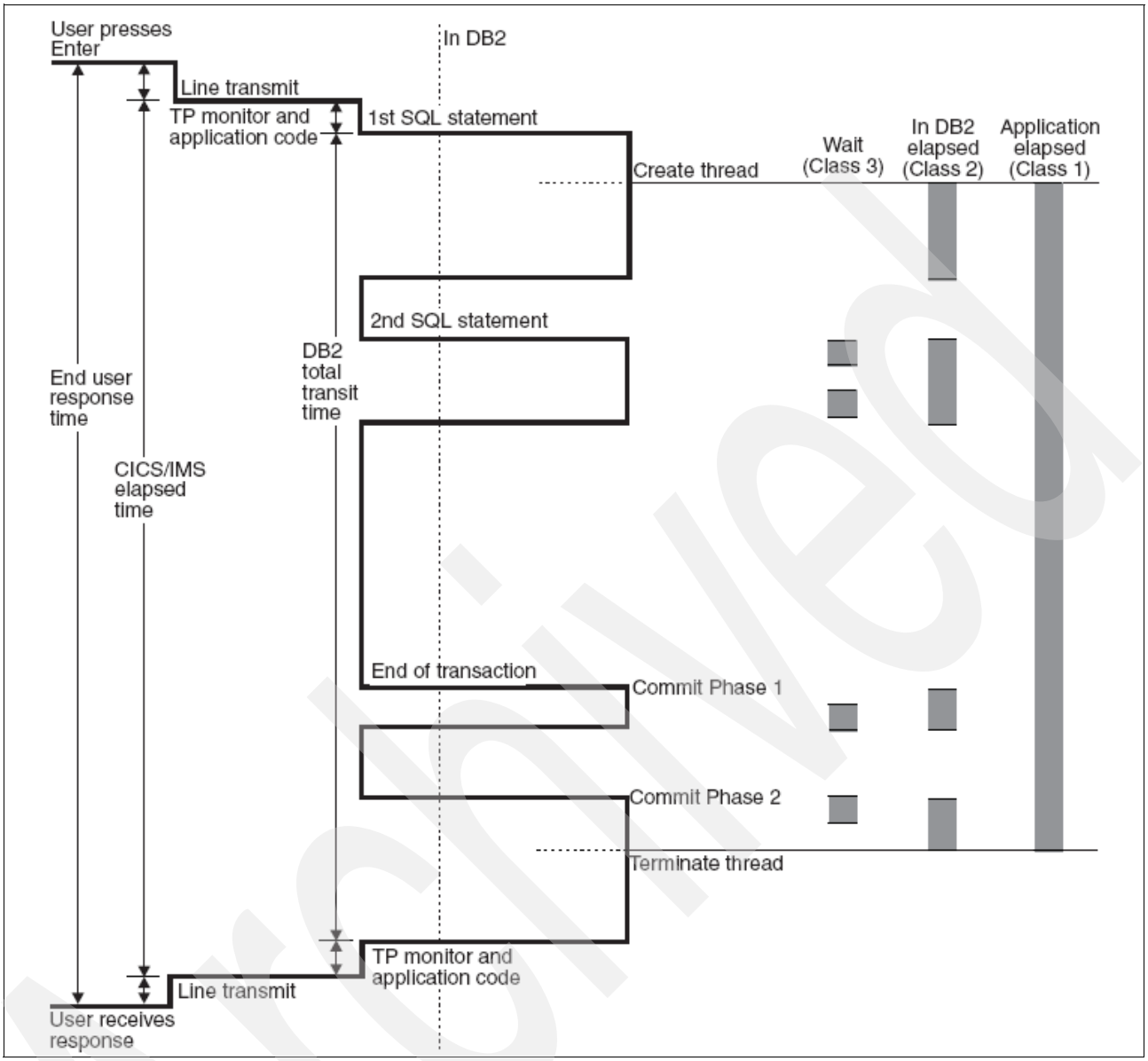


Figure 4-1 Components of DB2 transaction response time

As you can see in Figure 4-1, DB2 only "sees" a subset of the total elapsed time of the thread. In the following sections we describe the main components of DB2-visible response time, and how they can be affected by distance.

4.2.1 DB2 and application CPU times

As shown in Figure 4-1, the DB2 Class 1 elapsed time starts shortly after the first SQL call from the application, and ends shortly before DB2 returns control to the caller after the last COMMIT. This time will include periods when the caller is executing its own instructions, possibly on the DB2 data that was returned by the first call, as well as time during which the transaction was waiting for some other non-DB2 resource or event.

Note: While it does not apply to our workload, if you are exploiting thread reuse with DB2, meaning that the thread is not deleted at the end of each transaction and instead is available for use by another transaction, then the numbers reported by DB2PE reflect the total activity for the thread.

To get the numbers for each transaction, you need to normalize those values, adjusting for the average number of transactions per thread. This topic is discussed further in 4.4.1, “Threads and transactions” on page 69.

The DB2 Class 2 time is a subset of the Class 1 time, and consists of the time when DB2 is performing work for the application. This period includes time when DB2 is using the CPU at the request of the application (DB2 CPU time), and time when the transaction is waiting for some DB2-related processing to complete - this waiting time is called “Class 3 Suspension time” by DB2.

The DB2 Performance Expert (DB2PE) product breaks down the “using the CPU” time as follows:

DB2 (Class 2) CPU time The CPU time spent in DB2 for processing SQL statements, stored procedures, user-defined functions, or triggers. This is a subset of the Class 2 elapsed time, and is specifically reported in the `CPU TIME` field in the DB2PE Accounting report (marked with (1) in the sample report in Figure 4-8 on page 68).

Application (Class 1) CPU time This is the *total* CPU time spent in the application, from the point of the first SQL call, until the thread ends (or is reused by another transaction). For example, if the application makes one SQL call, followed by 1,000,000 COBOL statements, the time for the COBOL statements would be reported as Class 1 CPU time,

In this book, we use the term “non-DB2 Appl CPU time” to describe CPU time that is used by the application *outside* DB2. We define this as the difference between the Class 1 and Class 2 CPU times (marked with (2) in the sample report in Figure 4-8 on page 68).

Note: One factor that can affect this value is the use of the THREADSAFE option in CICS TS 2.2 and later. However, we did not use THREADSAFE in this project.

In addition to the CPU time used by DB2 and the application program, the DB2PE report also provides the total Class 1 elapsed time and the total Class 2 elapsed time, as follows:

DB2 (Class 2) Elapsed time This represents the end-to-end elapsed time for the DB2 part of the transaction execution. This is shown in Figure 4-1 on page 57.

Application (Class 1) Elapsed time This consists of the total Class 2 Elapsed time, plus the Application CPU time, plus time that the application spends waiting for non-DB2 resources or events. Note that although DB2 is aware of the total elapsed time (Class 1 time), it is not able to categorize all the time that the application is waiting for something outside DB2, so this wait time is not explicitly reported.

4.2.2 DB2 Class 3 Suspension

The main components of DB2 Class 3 Suspension time are as follows:

Lock/Latch This is the total elapsed time that the transaction waited for locks and latches held by other tasks in *this* DB2. It is reported in the LOCK/LATCH field in the DB2PE Accounting report (marked with (3) in the sample report in Figure 4-8 on page 68). This value does not include suspensions due to group-level lock contention in a data sharing environment.

There is an exception, however. Assume you have three transactions. Transaction A is running on D8Q1 and is holding resource ABC. Transaction B is running on D8Q2 and is delayed, waiting for resource ABC to be released. Transaction C is also running on D8Q2, and is also waiting (behind Transaction B) for resource ABC. Assuming that all transactions want to use resource ABC exclusively, transaction C cannot proceed until both transactions A and B have released the resource. So, some of the time that transaction C is waiting is actually caused by transaction A (and that you would expect to be reported as global contention), and some of it is caused by transaction B. However DB2 reports *all* the wait time for transaction C as Lock/Latch (that is, as local contention).

Synchronous I/O This is the total time spent waiting for synchronous I/O operations to complete. It is reported in the SYNCHRON. I/O field in the DB2PE Accounting report (marked with (4) in the sample report in Figure 4-8 on page 68). These can be both reads and writes, and can involve both the databases and the DB2 log data sets (the report tells you how many events were for database I/O and how many were for log data set I/O). DB2 calculates this value by subtracting the store clock time when an agent begins waiting for a synchronous I/O, from the time when the agent is resumed.

The majority of the synchronous I/Os we encountered in this project were related to log writes prior to COMMIT that had to be synchronous (log writes prior to COMMIT can be synchronous or asynchronous). The remainder occurred when DB2 required a database page that was not already in its local buffers or in the GBP in the CF.

Global contention This is the total time spent waiting for locks that are currently held by another DB2 subsystem in the data sharing group. It is reported in the GLOBAL CONTENTION field in the DB2PE Accounting report (marked with (5) in the sample report in Figure 4-8 on page 68).

There is also another situation that can contribute to the number of reported Global Contention events and the related suspend time. If DB2 issues a lock request, and XES converts that request to be an asynchronous one, IRLM is unable to differentiate between that situation and when XES encounters false contention. This even applies to situations where the contention is actually local, but where the associated CF request was converted to be an asynchronous one.

Therefore, the reported number of Global Contention events is likely to be greater than the actual number, especially if the issuing DB2 is not physically close to the Coupling Facility that contains the lock structure. For more information about the XES algorithm that controls this processing, refer to 2.2.1, “Synchronous and asynchronous CF requests” on page 9.

Asynch CF Reqs

This is the total time spent waiting for asynchronous requests, initiated by the thread prior to commit, to the Group Buffer Pool structures in the Coupling Facility to complete. It is reported in the `ASYNCH CF REQUESTS` field in the DB2PE Accounting report (marked with (6) in the sample report in Figure 4-8 on page 68). All DB2 write requests to the secondary GBP structure are issued asynchronously. Requests to the primary GBP structure are normally issued by DB2 as synchronous CF requests. However, depending on actual CF response times, the XES heuristic algorithm may decide to issue them asynchronously instead.

For writes to duplexed GBPs, if the write to the primary structure is sent synchronously, this field represents how long DB2 had to wait between when the write to the primary GBP and the write to the secondary GBP completes. If the write to the primary GBP is issued asynchronously, this field represents the time from sending the request to the secondary GBP through to the completion of whichever request finishes last.

If the request to the primary structure has actually been issued synchronously by XES, then the elapsed time of that request would be added to the DB2 CPU time counter.

Note: GBP write requests that are issued as a result of a COMMIT (when doing single-phase commit) are accounted for in the Update Commit category rather than in this category.

Most of the requests in this category are probably reads, although some writes may be included as well, particularly if DB2 has to write updated pages to the GBP prior to the COMMIT being issued. The sections titled “Writing to the group buffer pool” and “Group buffer pool thresholds” in *DB2 UDB for z/OS V8 Data Sharing Planning and Administration*, SA18-7417, discuss the situations in which such writes may occur. Unfortunately, the DB2PE reports do not provide a breakdown of how many of these requests were reads and how many were writes.

Update Commit

The events that get counted in Update Commit depend on the version of DB2, and whether the application is doing single-phase or two-phase commit.

For single-phase commit, as used in this project, it consists of the total wait time due to a synchronous execution unit switch for DB2 commit, abort, or deallocation processing. Commit processing consists of synchronous writes to the primary and secondary DB2 log data sets; writing all pages that have been changed within the scope of this commit to the primary and secondary GBPs; and releasing any locks associated with this commit, plus some other general DB2 cleanup processing associated with ending this transaction[1].

The elapsed time for all this processing is reported in the `Update Commit` field in the DB2PE Accounting report (marked with (7) in the sample report in Figure 4-8 on page 68). Unfortunately, the report does not provide the response times for the individual components of Update Commit.

[1] DB2 V8 added support for the ability to send multiple pages in the same pageset on a single IXLCACHE request. This may increase CF response times slightly, but should result in fewer CF requests. Reducing the number of interactions with the CF should be especially beneficial in a configuration with large distances to the CF.

For two-phase commit (which was not used by the transactions in this project), the writes to the GBPs are included in the ASYNCH CF REQUESTS field, and the phase 1 log I/Os are included in the LOG WRITE I/O field. The only thing that is included in UPDATE/COMMIT in that case is the time to perform the commit phase 2 logging.

Tip: One item that is “hidden” in the different suspension reasons is the time that the task spends waiting to get dispatched on a CP. For example, DB2 starts a timer when it calls the process to write a log record, and ends the timer when control is returned after the records have been written. However, it is possible that the process lost the CP to some other task at some point before control is returned to the thread. If you look at the end-to-end elapsed time, it is more than likely longer than the time to simply issue the I/Os.

To investigate this, we ran a trace to look at the I/O time and the end-to-end time to write some log records. The end-to-end time for the first log record was 6 ms, consisting of two I/Os at 1 ms each, plus 4 ms of unaccounted-for time. The end-to-end time to write another log record a few seconds later was just 4 ms; the two I/Os again took 1 ms each, but this time we had just 2 ms of unaccounted time.

These unaccounted times tend to average out, so you do not need to be particularly concerned about them. Just remember that an end-to-end process often takes longer than the sum of its constituent parts.

Not Account

In addition to these categories, there is one other significant component of DB2 transaction elapsed time. This is a catch-all for all suspensions that DB2 is unable to categorize. It is *not* included in the DB2 Class 3 suspension time - the Class 3 suspensions are all the ones that DB2 is able to categorize. This other category is called Not Account, and (among other things) consists of:

- Time spent waiting in z/OS for a CP. In a non-data sharing environment, it could be due to running on a very high CPU utilization environment and waiting for CPU cycles, especially if DB2 has a low dispatching priority in WLM.
- In some cases, high Not Accounted time is caused by too much detailed online tracing or bugs in some performance monitors. Reducing the level of tracing or stopping the performance monitor may help to reduce the Not Accounted time to an acceptable level. This situation is usually the primary cause of high Not Accounted values on systems that are not CPU-constrained.
- Instrumentation Facility Interface (IFI) log reads.
- If the environment is very I/O intensive, the Media Manager might be running out of request blocks.
- z/OS events can cause Not Accounted time (for example, when an SMF SRB is triggered to collect open data set statistics, or waiting for MVS to page-in a page that DB2 needs).
- Waiting for package accounting.
- A PREPARE which is not found in the Dynamic Statement Cache.
- Data set open contention related to PCLOSET being too small.
- Time for RMF interval data set statistics gathering.

The amount of time that gets reported in Not Account is generally a fairly consistent percentage of all wait times. As you will see, the time reported in Not Account increases as the distance increases. However, as a percentage of total wait time, the Not Account time is actually fairly consistent throughout the measurements.

APAR PK27934 delivers significant improvements in DB2's categorization of suspensions, resulting in much less time being reported in the Not Account field.

For a more complete list of contributors to Not Account time, refer to the discussion available on the Web at the following address:

http://www.ibm.com/support/docview.wss?&uid=swg21045823&lang=en

4.2.3 DB2 I/Os

DB2 in data sharing mode issues a number of different types of I/O requests, as explained in this section.

The first of these is related to castout processing. When certain thresholds are reached, the DB2 DBM1 address space retrieves updated pages from the DB2 Group Buffer Pools and writes them out to disk in a process called "castout" (as discussed in 3.3, "DB2 data sharing - the high altitude view" on page 29). These writes are asynchronous from the perspective that they have no impact on transaction response times; the writes happen long after the transaction has completed its updates. Because of the large volume of data being moved, these I/Os would typically have relatively long disk response times (DB2 can move up to 128 KB in a single I/O request).

The next type of I/O is "prefetch". When DB2 determines that an application is processing a table sequentially, it starts prefetching the data in advance, in an effort to ensure the data is available in the DB2 local buffers when the application requests it. These I/O requests will fetch many pages in a single I/O, and will typically result in longer disk response times (and higher CONNECT times) because of the large amount of data being moved on each I/O. Both the castout and prefetch I/O requests are charged to the DBM1 address space.

The third type of I/O, random reads for pages that are not available in the local or Group Buffer Pools, are also done by the DBM1 address space. However, these are charged to the requesting user (in our case, to the CICS Application Owning Regions).

The final type of I/O done by DB2 is logging. All log I/Os are done by the DB2 MSTR address space, and DB2 normally operates with dual logs, so every log write request actually results in two I/Os. Most log writes prior to COMMIT are asynchronous, but log writes done as a result of a COMMIT (and in certain other situations) are synchronous.

DB2 and Parallel Access Volumes

Parallel Access Volumes (PAVs) can have a beneficial effect on the performance of DB2 I/O processing. The availability of PAV means that DB2 I/O processing can be accelerated by being able to drive multiple I/Os at a time to a single DB2 volume. You can effectively have heavy castout activity for a volume running in parallel with synchronous reads, and still be able to deliver acceptable response times for the read I/Os. Similarly, PAVs should improve performance when DB2 is performing multiple concurrent synchronous reads against the same volume.

Remember that in certain conditions, DB2 runs multiple castout processes concurrently, each of them dealing with a separate tablespace, and this processing could also benefit from the availability of PAVs.

The importance of PAVs in a multisite sysplex environment, or even when simply performing synchronous remote copy, is discussed in "PAV and HyperPAV" on page 17.

4.2.4 DB2 and CICS threads

When you create the DSNZPARM member to customize your DB2, one of the values you specify is the number of threads that DB2 can use. This is specified on the `CTHREAD` parameter. Unfortunately, there is no way to display the value in use by DB2 using a DB2 command.

On the CICS side, you can also specify how many threads are to be available for talking to DB2. You can have “protected threads”, which are only available to a specified group of transactions, and you can have “pool threads”. The number of pool threads is specified on the `threadlimit` parameter in the `DB2Connect` entry in the CICS CSD.

If the number of threads defined to CICS is not large enough, there will be queuing within CICS, waiting for a thread to become free. This is reported as `DB2 Thread Wait Time` in CICS PA reports, but this time is not visible to DB2PE Accounting reports.

If there are sufficient threads defined to CICS, but the CTHREAD value in DB2 is too small, queuing will take place within DB2. Because transaction level reporting in DB2 is based on threads, the time that is spent *waiting* for a DB2 thread is not included in the transaction times as seen by DB2. It is possible, however, to see if this queuing is taking place by checking the `QUEUED AT CREATE THREAD` field in the DB2PE Statistics reports.

4.3 Reporting tools used for measurements

Anyone familiar with performance monitoring will know that in order to get a complete picture of what is happening in a z/OS system, with sufficient detail to understand the impact of configuration changes, you must use a number of monitors. A DB2 monitor will provide very detailed information about the life of a DB2 thread. A CICS monitor enriches this with information about what is happening to the transaction in addition to the DB2 processing. And an MVS monitor shows information about the hardware and software configuration, including how it is being utilized and the performance it is delivering. In this section, we discuss the CICS Performance Analyzer, RMF, and DB2 Performance Expert reports that we include in this book.

Note: The reports presented as samples in this part of the chapter were taken from various, different tests that we performed during this project. They do not necessarily relate to any of the formal measurements presented later in this book, nor were they necessarily all taken from the same test run.

4.3.1 CICS Performance Analyzer reports

We start with the CICS Performance Analyzer (CICS PA) reports. The CICS PA Summary report we use provides summary information about each transaction type executed in a specific CICS region over the interval. Because most of the information we will be using comes from DB2PE, we really only use the CICS PA Summary report to validate the transaction numbers (reported in the #Tasks column) and the response times (reported in the Avg Response Time column) against those reported by DB2PE and RMF.

Tran	#Tasks	Avg Response Time	Max Response Time	Avg Dispatch Time	Avg User CPU Time	Avg SYNCProc Time	Max SYNCProc Time	Avg DispWait Time	Avg FC Wait Time	Avg IR Wait Time	Avg DB2ThdWt Time	Avg DB2SQLWt Time
TPCI	16633	.0047	.3768	.0014	.0007	.0005	.0232	.0005	.0000	.0000	.0028	.0000
TPDF	28934	.0004	.0134	.0003	.0002	.0001	.0037	.0000	.0000	.0000	.0000	.0000
TPNO	29017	.0454	.3881	.0386	.0055	.0074	.1739	.0029	.0000	.0000	.0029	.0000
TPOS	2926	.0081	.2205	.0037	.0013	.0004	.0048	.0012	.0000	.0000	.0028	.0000
TPPA	2787	.0264	.3744	.0223	.0017	.0034	.1184	.0009	.0000	.0000	.0029	.0000
TPSL	2640	.0486	.2804	.0453	.0142	.0006	.0056	.0006	.0000	.0000	.0026	.0000
Total	82937	.0197	.3881	.0162	.0027	.0028	.1739	.0012	.0000	.0000	.0018	.0000

Figure 4-2 Sample CICS PA report for one AOR

Each of our CICS PA reports provides information about a single region, either a Terminal Owning Region (TOR) or an Application Owning Region (AOR); Figure 4-2 illustrates a sample report. Each TOR can potentially route any transaction to any of the four AORs across the two systems. However, each AOR is only associated with a single DB2 subsystem. Remember, as shown in Figure 4-1 on page 57, that the total elapsed time that DB2 is aware of (its Class 1 elapsed time) is a subset of the total CICS transaction response time, so we would expect the response times seen in the CICS PA report for an AOR to be a little longer than the Class 1 elapsed time for the associated DB2 subsystem.

The other CICS PA report we used was the Wait Analysis report, an example of which is shown in Figure 4-3. This report provides information about waits in the transaction that may not be visible to DB2 accounting. In this example, 99.8% of the wait time for the application was due to a shortage of CICS threads to DB2 (a situation we subsequently addressed).

```
-------- Time --------                                  ------ Ratio ------
                                                           Total     Average

 # Tasks                                                   11145
 Response Time                                          19198.1814      1.7226
 Dispatch Time                                           2964.8341      0.2660                  15.4% of Response
 CPU Time                                                  70.1317      0.0063                   2.4% of Dispatch
 Suspend Wait Time                                      16233.2714      1.4566                  84.6% of Response
 Dispatch Wait Time                                        13.3639      0.0012                   0.1% of Suspend
 Resource Manager Interface (RMI) elapsed time          19159.4274      1.7191                  99.8% of Response
 Resource Manager Interface (RMI) suspend time          16206.6972      1.4542                  99.8% of Suspend

                                                  ------------------- Suspend Time -------------------
                                                        Total    Average    %age  Graph
DB2RDYQW DB2 Thread wait time                       16205.8626      1.4541  99.8% |********************
DSCHMDLY Redispatch wait time caused by change-TCB mode    26.0474      0.0023   0.2% |
```

Figure 4-3 CICS PA Wait Analysis report

In this report, the following fields are of interest:

- The Response Time is equal to the Dispatch Time plus Suspend Wait Time.
- The Resource Manager Interface time includes some time in CICS and some time in the Resource Manager (DB2 in this case). In this example, the time that CICS spends waiting for a thread (because we had defined the CICS threadlimit value to be too low) is included in the Resource Manager Interface suspend time.
- The Suspend Wait Time includes the suspend time within the Resource Manager Interface elapsed time.

For more information about correlating CICS PA and DB2PE reports, refer to the chapter titled “Accounting and Monitoring in a CICS DB2 Environment” in *CICS DB2 Guide,* SC34-6457.

4.3.2 RMF reports

We used four types of RMF reports for this project:

- Workload Activity reports
- Disk Activity reports
- Coupling Facility Activity reports
- XCF Path Statistics reports

We explain these reports in more detail in the following sections.

Workload Activity report

The Workload Activity reports externalize information that the MVS Workload Manager provides to RMF, reporting information such as transaction counts, average response times and response time distributions, and achievement against the installation-specified service class goals. A sample of such a report is shown in Figure 4-4.

```
REPORT BY: POLICY=WLMPOL     WORKLOAD=ONL_WKL    SERVICE CLASS=CICSHI     RESOURCE GROUP=*NONE      PERIOD=1 IMPORTANCE=1
                                                 CRITICAL     =NONE

TRANSACTIONS    TRANS.-TIME  HHH.MM.SS.TTT
AVG      0.00   ACTUAL                 9
MPL      0.00   EXECUTION              8
ENDED  333333   QUEUED                 0
END/S  555.56   R/S AFFINITY           0
#SWAPS      0   INELIGIBLE             0
EXCTD  333307   CONVERSION             0
AVG ENC  0.00   STD DEV                23
REM ENC  0.00
MS ENC   0.00

           RESP  ---------------------------------- STATE SAMPLES BREAKDOWN (%) -----------------------------  ------STATE------
SUB    P   TIME  --ACTIVE-- READY IDLE  ----------------------------WAITING FOR----------------------------  SWITCHED SAMPL(%)
TYPE        (%)   SUB  APPL              I/O LOCK MISC TIME PROD CONV LTCH                                    LOCAL SYSPL REMOT
CICS  BTE   745   0.4   0.0   0.1 24.1   0.0  0.0 48.3 16.1  0.0 11.0  0.0                                    10.9   0.2   0.0
CICS  EXE  40.8  64.9   0.0  21.6  0.0   0.1  0.0  0.0  0.0 13.4  0.0  0.0                                     0.0   0.0   0.0
DB2   BTE   0.0   0.0   0.0   0.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0                                     0.0   0.0   0.0
DB2   EXE  38.1  26.6   0.0   0.0  0.0   5.7 64.6  3.0  0.0  0.0  0.0  0.1                                     0.0   0.0   0.0

GOAL: RESPONSE TIME 000.00.00.500 FOR  90%

         RESPONSE TIME EX   PERF
SYSTEM      ACTUAL%    VEL% INDX

*ALL           100      N/A  0.5
#@$2           100      N/A  0.5
#@$3           100      N/A  0.5
----------RESPONSE TIME DISTRIBUTION----------
   ----TIME----    --NUMBER OF TRANSACTIONS--    -------PERCENT-------  0    10   20   30   40   50   60   70   80   90   100
   HH.MM.SS.TTT    CUM TOTAL        IN BUCKET    CUM TOTAL   IN BUCKET  |....|....|....|....|....|....|....|....|....|....|
<  00.00.00.250         333K             333K         100         100  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<= 00.00.00.300         333K              29          100         0.0  >
<= 00.00.00.350         333K               1          100         0.0  >
<= 00.00.00.400         333K               0          100         0.0  >
<= 00.00.00.450         333K               0          100         0.0  >
<= 00.00.00.500         333K               0          100         0.0  >
```

Figure 4-4 Sample RMF Workload Activity report

We used the RMF Workload Activity report to determine the transaction rate on each z/OS system, the transaction response times, and the goal achievement.

In an environment where you are using CICS TORs and AORs as we did, the ACTUAL transaction time reported by RMF is the value it gets from the TORs. The EXECUTION time is the time reported by the AORs. *Normally*, the ACTUAL time is larger than EXECUTION time. However, if a significant number of transactions are being routed from this TOR to an AOR on another system (because the AORs on this system are delivering poorer response times), you may see longer EXECUTION times than ACTUAL times.

We did not use any of the State Samples information from RMF because we obtained the detailed information about the transactions from DB2PE.

DASD Actiivty report

We also used the DASD Activity reports; specifically, the SDEVICE reports that provide a sysplex-wide view of disk activity. These reports show the I/O rate and response time as seen by each system, as well as a breakdown of the components of I/O response times. An example is shown in Figure 4-5.

```
                          SMF            DEVICE    AVG  AVG    AVG  AVG  AVG  AVG  AVG    %      %      %     AVG
DEV  DEVICE   VOLUME PAV  SYS  IODF LCU  ACTIVITY  RESP IOSQ   CMR  DB   PEND DISC CONN   DEV    DEV    DEV   NUMBER
NUM  TYPE     SERIAL     ID   SUFF      RATE      TIME TIME   DLY  DLY  TIME TIME TIME   CONN   UTIL   RESV  ALLOC


D052 33909    #@$#D2     *ALL           682.769   0.7  0.0    0.0  0.0   0.2  0.3  0.2   7.47  19.01   0.0   160
                     2  #@$2  29  00B0  343.488   0.7  0.0    0.0  0.0   0.2  0.3  0.2   3.59   9.36   0.0  81.0
                     2  #@$3  29  00B0  339.281   0.7  0.0    0.0  0.0   0.2  0.3  0.2   3.88   9.65   0.0  79.0

D253 33909    DISTD1     *ALL           182.579   1.2  0.0    0.0  0.0   0.2  0.6  0.5   1.51   3.25   0.0   128
                     6  #@$2  29  00B2   96.519   1.9  0.0    0.0  0.0   0.2  1.0  0.8   1.25   2.83   0.0  46.0
                     6  #@$3  29  00B2   86.060   0.4  0.0    0.0  0.0   0.2  0.1  0.2   0.26   0.42   0.0  82.0
```

Figure 4-5 Sample RMF SDEVICE report

It is important to remember that the DB2PE fields related to waits for log write I/O activity include more than simply the time from the I/O was issued to when it completed, so we normally see DB2PE I/O wait times that are a little higher than the corresponding disk response times reported by RMF.

Coupling Facility Activity report

The next type of RMF report that we used is the Coupling Facility Activity report, especially the sections that provide details about the activity for each structure, and the section that provides subchannel utilization information. Figure 4-6 on page 67 shows a sample Coupling Facility report providing information for a lock structure and the subchannel activity report.

```
STRUCTURE NAME = DB8QU_LOCK1       TYPE = LOCK   STATUS = ACTIVE
          # REQ    -------------- REQUESTS -------------   -------------- DELAYED REQUESTS -------------
SYSTEM    TOTAL             #     % OF  -SERV TIME(MIC)-    REASON    #   % OF  ---- AVG TIME(MIC) -----  EXTERNAL REQUEST
NAME      AVG/SEC         REQ      ALL     AVG   STD_DEV            REQ    REQ   /DEL    STD_DEV   /ALL   CONTENTIONS

#@$2       3628K   SYNC   2213K   31.1    21.3      5.3     NO SCH  302    0.0    7.8        7.7    0.0   REQ TOTAL     4404K
           6047    ASYNC  1415K   19.9    62.4     42.6     PR WT     0    0.0    0.0        0.0    0.0   REQ DEFERRED    56K
                   CHNGD     0    0.0  INCLUDED IN ASYNC     PR CMP    0    0.0    0.0        0.0    0.0   -CONT           37K
                                                                                                          -FALSE CONT     13K

#@$3       3484K   SYNC   2111K   29.7    21.8      6.6     NO SCH  279    0.0    7.8        8.1    0.0   REQ TOTAL     4257K
           5807    ASYNC  1373K   19.3    63.5     42.3     PR WT     0    0.0    0.0        0.0    0.0   REQ DEFERRED    53K
                   CHNGD     0    0.0  INCLUDED IN ASYNC     PR CMP    0    0.0    0.0        0.0    0.0   -CONT           36K
                                                                                                          -FALSE CONT     12K

-------------------------------------------------------------------------------------------------------------------------------
TOTAL      7113K   SYNC   4324K   60.8    21.5      6.0     NO SCH  581    0.0    7.8        7.9    0.0   REQ TOTAL     8661K
          11854    ASYNC  2788K   39.2    63.0     42.4     PR WT     0    0.0    0.0        0.0    0.0   REQ DEFERRED   109K
                   CHNGD     0    0.0                      PR CMP    0    0.0    0.0        0.0    0.0   -CONT           74K
                                                                                                          -FALSE CONT     25K



                                                SUBCHANNEL  ACTIVITY
-------------------------------------------------------------------------------------------------------------------------------
          # REQ                              ----------- REQUESTS -----------   ------------------ DELAYED REQUESTS --------------
 SYSTEM   TOTAL   -- CF LINKS --  PTH              #   -SERVICE TIME(MIC)-              #    % OF  ------ AVG TIME(MIC) ------
 NAME     AVG/SEC TYPE  GEN  USE  BUSY            REQ     AVG    STD_DEV               REQ    REQ   /DEL    STD_DEV     /ALL

 #@$2       5048K CFP     2    2     0   SYNC      2223K   21.5       8.2   LIST/CACHE    5   0.0   26.2        7.7      0.0
           8412.6 ICP     2    0         ASYNC     2785K   91.5      90.4   LOCK        302   0.0    7.8        7.7      0.0
                  SUBCH  28   14         CHANGED      5  INCLUDED IN ASYNC  TOTAL       307   0.0
                                          UNSUCC       0     0.0       0.0
 #@$3       4848K CFP     2    2     0   SYNC      2119K   21.9       8.7   LIST/CACHE    4   0.0   39.0        9.9      0.0
           8079.5 SUBCH  14   14         ASYNC     2691K   89.0      79.0   LOCK        279   0.0    7.8        8.1      0.0
                                          CHANGED      3  INCLUDED IN ASYNC  TOTAL       283   0.0
                                          UNSUCC       0     0.0       0.0
```

Figure 4-6 Sample RMF Coupling Facility Activity report

In this report, we monitored the request response time, shown under SERV TIME AVG, and the percentage of requests that are sent synchronously or asynchronously. As the response time increases, the percent of requests that are asynchronous will also increase.

XCF Activity reports

The last RMF report we used is the one that provides information about how many XCF messages were sent in each transport class, and which paths were used; this is the RMF XCF Path Statistics report. A sample report is shown in Figure 4-7.

```
TOTAL SAMPLES = 540                                XCF PATH STATISTICS
-------------------------------------------------------------------------------------------------------------------------------
                          OUTBOUND FROM #@$2                                              INBOUND TO #@$2
-------------------------------------------------------------------------------------   -----------------------------------
        T FROM/TO                                                                              T FROM/TO
 TO     Y DEVICE, OR      TRANSPORT       REQ     AVG Q                                FROM    Y DEVICE, OR          REQ BUFFERS
 SYSTEM P STRUCTURE       CLASS           OUT     LNGTH    AVAIL   BUSY   RETRY        SYSTEM  P STRUCTURE            IN UNAVAIL
 #@$3   S IXC_BIG_1       BIG             211      0.00      211      0       0        #@$3    S IXC_BIG_1           991       0
        S IXC_BIG_2       BIG             139      0.00      139      0       0                S IXC_BIG_2         1,041       0
        S IXC_DEFAULT_1   DEFAULT      65,760      0.01   65,760      0       0                S IXC_DEFAULT_1    60,086       0
        S IXC_DEFAULT_2   DEFAULT      36,366      0.01   36,366      0       0                S IXC_DEFAULT_2    30,280       0
        C 4FF1 TO 5FE1    DEFAULT           1      0.00        1      0       0                C 5FE0 TO 4FF0     12,012       0
        C 4FF3 TO 5FE3    BIG             110      0.00      110      0       0                C 5FE2 TO 4FF2      1,126       0
        C 5FF1 TO 4FE1    DEFAULT          15      0.00       15      0       0                C 4FE0 TO 5FF0          9       0
        C 5FF3 TO 4FE3    BIG             643      0.01      643      0       0                C 4FE2 TO 5FF2        896       0
```

Figure 4-7 Sample RMF XCF Path Statistics report

In this report you can see that the bulk of messages are in the DEFAULT transport class (messages of 956 bytes or less), and that most of them were sent using the CF structures

rather than the FICON CTC paths. As the distance between the sites increased, we saw a gradual shift to more requests going over the CTC links.

The use and setup of FICON CTCs is discussed in more detail in 2.4, “CTCs” on page 18.

4.3.3 DB2 Performance Expert reports

The DB2 Performance Expert (DB2PE) reports provide a plethora of information about DB2, a subset of which is specifically relevant to a sysplex data sharing environment. DB2PE produces Statistic reports, which provide information about DB2 activity at the subsystem level, and Accounting reports, which provide information at the thread level.

Because we were focusing on the impact of distance on transaction response times, the DB2PE report that we used most are parts of the Accounting report. A sample report is shown in Figure 4-8. The fields that we focused on are discussed in 4.4, “Understanding DB2PE reports” on page 69.

```
   LOCATION: DB8Q                          DB2 PERFORMANCE EXPERT (V2)                    PAGE: 1-1
      GROUP: DB8QU                          ACCOUNTING REPORT - LONG          REQUESTED FROM: 08/10/06 17:57:00.00
     MEMBER: D8Q2                                                                      TO: 08/10/06 18:07:00.00
  SUBSYSTEM: D8Q2                           ORDER: PRIMAUTH-PLANNAME          INTERVAL FROM: 08/10/06 17:57:00.11
DB2 VERSION: V8                               SCOPE: MEMBER                             TO: 08/10/06 18:06:59.98

PRIMAUTH: CICSUSER  PLANNAME: TPCCP2NO

ELAPSED TIME DISTRIBUTION                                          CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------      ------------------------------------------------------
APPL   |> 1%                                                       CPU    |> 1%
DB2    |> 1%                                                       NOTACC |> 1%
SUSP   |================================================> 98%      SUSP   |=============================================> 99%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    -------------------   ------------  --------    --------------------------
ELAPSED TIME    0.895887 (2) 0.888769         N/P   LOCK/LATCH(DB2+IRLM)     0.642215 (3)   6.26   #OCCURRENCES    :    18562
 NONNESTED      0.895887    0.888769         N/A   SYNCHRON. I/O         (4) 0.005843       2.10   #ALLIEDS        :    18562
 STORED PROC    0.000000    0.000000         N/A    DATABASE I/O             0.001803       1.52   #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A    LOG WRITE I/O            0.004040       0.58   #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A   OTHER READ I/O            0.001443       1.20   #DBATS DISTRIB. :        0
                                                   OTHER WRTE I/O            0.000005       0.00   #NO PROGRAM DATA:        0
CPU TIME        0.006501    0.005277 (1)     N/P   SER.TASK SWTCH            0.025937       1.00   #NORMAL TERMINAT:    18562
 AGENT          0.006501    0.005277         N/A    UPDATE COMMIT            0.025600 (7)   1.00   #ABNORMAL TERMIN:        0
  NONNESTED     0.006501    0.005277         N/P    OPEN/CLOSE               0.000000       0.00   #CP/X PARALLEL. :        0
  STORED PRC    0.000000    0.000000         N/A    SYSLGRNG REC             0.000000       0.00   #IO PARALLELISM :        0
  UDF           0.000000    0.000000         N/A    EXT/DEL/DEF              0.000336       0.00   #INCREMENT. BIND:        0
  TRIGGER       0.000000    0.000000         N/A    OTHER SERVICE            0.000000       0.00   #COMMITS        :    18361
 PAR.TASKS      0.000000    0.000000         N/A   ARC.LOG(QUIES)            0.000000       0.00   #ROLLBACKS      :      201
                                                   ARC.LOG READ              0.000000       0.00   #SVPT REQUESTS  :        0
SUSPEND TIME    0.000000    0.876475         N/A   DRAIN LOCK                0.000000       0.00   #SVPT RELEASE   :        0
 AGENT              N/A     0.876475         N/A   CLAIM RELEASE             0.000000       0.00   #SVPT ROLLBACK  :        0
 PAR.TASKS          N/A     0.000000         N/A   PAGE LATCH                0.001894       1.47   MAX SQL CASC LVL:        0
 STORED PROC    0.000000        N/A          N/A   NOTIFY MSGS               0.000000       0.00   UPDATE/COMMIT   :    22.96
 UDF            0.000000        N/A          N/A   GLOBAL CONTENTION         0.179935 (5)  44.55   SYNCH I/O AVG.  : 0.002779
                                                   COMMIT PH1 WRITE I/O      0.000000       0.00
NOT ACCOUNT.        N/A     0.007017         N/A   ASYNCH CF REQUESTS    (6) 0.019201      19.74
DB2 ENT/EXIT        N/A       139.74         N/A   TOTAL CLASS 3             0.876475      76.32
EN/EX-STPROC        N/A         0.00         N/A
EN/EX-UDF           N/A         0.00         N/A
DCAPT.DESCR.        N/A          N/A         N/P
LOG EXTRACT.        N/A          N/A         N/P
```

Figure 4-8 Sample DB2PE Accounting report

An extract from a sample Statistics report is shown in Figure 4-9 on page 69. The Statistics report provides information about everything that happened in that DB2 subsystem over the interval, but at the subsystem level (rather than at the transaction level). For that reason, it is not as relevant for the specific topics we focus on in this book.

LOCKING ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
SUSPENSIONS (ALL)	80734.00	147.50	1.69	1.62
SUSPENSIONS (LOCK ONLY)	72341.00	132.16	1.52	1.46
SUSPENSIONS (IRLM LATCH)	8045.00	14.70	0.17	0.16
SUSPENSIONS (OTHER)	348.00	0.64	0.01	0.01
TIMEOUTS	0.00	0.00	0.00	0.00
DEADLOCKS	0.00	0.00	0.00	0.00
LOCK REQUESTS	3297.0K	6023.45	69.08	66.35
UNLOCK REQUESTS	992.0K	1812.42	20.79	19.96
QUERY REQUESTS	144.00	0.26	0.00	0.00
CHANGE REQUESTS	300.8K	549.54	6.30	6.05
OTHER REQUESTS	0.00	0.00	0.00	0.00
LOCK ESCALATION (SHARED)	0.00	0.00	0.00	0.00
LOCK ESCALATION (EXCLUSIVE)	0.00	0.00	0.00	0.00

DATA SHARING LOCKING	QUANTITY	/SECOND	/THREAD	/COMMIT
GLOBAL CONTENTION RATE (%)	0.84			
L-LOCKS XES RATE (%)	19.73			
LOCK REQUESTS (P-LOCKS)	629.0K	1149.18	13.18	12.66
UNLOCK REQUESTS (P-LOCKS)	628.4K	1148.11	13.17	12.65
CHANGE REQUESTS (P-LOCKS)	12.00	0.02	0.00	0.00
SYNCH.XES - LOCK REQUESTS	1279.5K	2337.53	26.81	25.75
SYNCH.XES - CHANGE REQUESTS	3572.00	6.53	0.07	0.07
SYNCH.XES - UNLOCK REQUESTS	1645.5K	3006.29	34.48	33.12
ASYNCH.XES - RESOURCES	0.00	0.00	0.00	0.00
SUSPENDS - IRLM GLOBAL CONT	11877.00	21.70	0.25	0.24
SUSPENDS - XES GLOBAL CONT.	1235.00	2.26	0.03	0.02
SUSPENDS - FALSE CONT&CONV	1078.5K	1970.36	22.60	21.70
SUSPENDS - FALSE CONT.	11626.00	21.24	0.24	0.23
INCOMPATIBLE RETAINED LOCK	0.00	0.00	0.00	0.00

Figure 4-9 Sample DB2PE Statistics report - locking section

4.4 Understanding DB2PE reports

In this section we discuss the fields in the DB2 Performance Expert reports that we will be using, and explain how to interpret them.

4.4.1 Threads and transactions

Before we discuss the specific fields in the DB2PE reports, we need to talk about the relationship between threads and transactions. This will be obvious to our DB2 readers, but may not be as familiar to MVS system programmers.

When looking at DB2PE Accounting reports, keep in mind that they provide information about *threads*, and not transactions. Under specific conditions, a DB2 thread can be reused for multiple transactions. For this reason, in order to be able to relate the numbers in the DB2PE Accounting reports to an individual transaction, you may have to apply some normalization techniques to the reported data.

```
ELAPSED TIME DISTRIBUTION                                       CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------   ---------------------------------------------------------------
APPL   |=====> 11%                                              CPU    |======> 12%
DB2    |======> 13%                                             NOTACC |=> 2%
SUSP   |====================================> 76%               SUSP   |=============================================> 86%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.040546    0.036054         N/P    LOCK/LATCH(DB2+IRLM)      0.009777      1.29    #OCCURRENCES    :    29114
 NONNESTED      0.040546    0.036054         N/A    SYNCHRON. I/O             0.002155      2.48    #ALLIEDS        :    29114
 STORED PROC    0.000000    0.000000         N/A     DATABASE I/O             0.000734      1.90    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A     LOG WRITE I/O            0.001421      0.58    #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O            0.000044      0.19    #DBATS DISTRIB. :        0
                                                    OTHER WRTE I/O            0.000000      0.00    #NO PROGRAM DATA:    29114
CPU TIME        0.005652    0.004503         N/P    SER.TASK SWTCH            0.006778      1.00    #NORMAL TERMINAT:    29114
 AGENT          0.005652    0.004503         N/A     UPDATE COMMIT            0.006686      1.00    #ABNORMAL TERMIN:        0
  NONNESTED     0.005652    0.004503         N/P     OPEN/CLOSE               0.000000      0.00    #CP/X PARALLEL. :        0
  STORED PRC    0.000000    0.000000         N/A     SYSLGRNG REC             0.000000      0.00    #IO PARALLELISM :        0
  UDF           0.000000    0.000000         N/A     EXT/DEL/DEF              0.000092      0.00    #INCREMENT. BIND:        0
  TRIGGER       0.000000    0.000000         N/A     OTHER SERVICE            0.000000      0.00    #COMMITS        :    28856
 PAR.TASKS      0.000000    0.000000         N/A    ARC.LOG(QUIES)            0.000000      0.00    #ROLLBACKS      :      258
                                                    ARC.LOG READ              0.000000      0.00    #SVPT REQUESTS  :        0
SUSPEND TIME    0.000000    0.030956         N/A    DRAIN LOCK                0.000000      0.00    #SVPT RELEASE   :        0
 AGENT              N/A    0.030956         N/A    CLAIM RELEASE             0.000000      0.00    #SVPT ROLLBACK  :        0
 PAR.TASKS          N/A    0.000000         N/A    PAGE LATCH                0.000059      0.06    MAX SQL CASC LVL:        0
 STORED PROC    0.000000         N/A         N/A    NOTIFY MSGS               0.000000      0.00    UPDATE/COMMIT   :    22.99
 UDF            0.000000         N/A         N/A    GLOBAL CONTENTION         0.009580     32.60    SYNCH I/O AVG.  : 0.000868
```

Figure 4-10 DB2 threads versus transactions

Figure 4-10 contains an excerpt from a DB2PE Accounting report for one of our tests. The numbers in the left side of the report (up to and including the AV.EVENT column) provide averaged information for *each* thread. The right side of the report (the columns under HIGHLIGHTS) contains totals information for the *entire* interval being reported on.

As you can see from the report, during the interval being reported on (10 minutes), we ran 29114 threads (#OCCURRENCES), which issued 28856 commit requests (#COMMITS) and 258 backouts (#ROLLBACKS; in other words, one commit or rollback per thread. Because the associated transaction only issues a single commit, we can infer that every thread executed exactly one transaction. Although in this case there is no difference between the number of threads and the number of transactions, we will monitor these numbers as we increase the distance in the configuration. If the ratio changes, we will need to adjust our reports accordingly to ensure we are doing like-for-like comparisons.

Note: Because there is a one-to-one relationship between transactions and threads in the preceding example, for simplicity we omit discussion of the normalization process in the remaining examples in this section. However, if your environment involves more than one transaction per thread, you must remember to make this adjustment if you are carrying out this exercise for your own workload.

DB2 accounting and trace records

The amount of information you get in the DB2 PE Accounting Reports is related to the level of tracing that you enable in DB2.

Running with DB2 statistics trace classes 1, 3, 4, 5, and 6 is generally not a problem, even when gathering this data every minute (ZPARM STATIME=1), which is highly recommended by the DB2 performance team.

Running with accounting classes 1 and 3 is normally not a problem either, and most customers run with accounting classes 1, 2, and 3, or even 1, 2, 3, 7, and 8 enabled. Classes 7 and 8 provide package-level information and generate more overhead. Class 2 tracks the time inside DB2. For transactions that make a significant number of SQL calls, for example

FETCH 1,000,000 rows, the overhead of activating class 2 is higher. Class 3 is the wait times, and they are very helpful.

For the performance traces, anything that generates many trace records is costly; for example, tracing detailed locking (perf class 7), or some IFCIDs like IFCID 198, which is written for each page touched by DB2. We definitely do *not* recommend having these traces active all the time; instead, turn them on for 5 to 30 seconds if you are trying to gather diagnostic information for a problem. Also, be sure to write them to a trace destination that can handle the amount of trace records, usually GTF (in order to not overrun SMF).

Because our environment was only used for measuring the workload and not for production, the objective was to obtain as much insight into DB2's behavior as possible. Therefore, the overhead of turning on a lot of traces was acceptable. After some trial and error, we ran our tests with statistic trace classes 1, 3, 4, 5, and 6 and accounting trace classes 1, 2, and 3 turned on, which resulted in the recording of the following IFCIDs being created:

- 1
- 2
- 3
- 105
- 106
- 202
- 225
- 230

Generally speaking, the more detail the trace provides, the higher the cost. If this were a production environment, you would need to give serious consideration to whether the overhead is acceptable, and how long you want to turn the traces on for.

Lock/Latch

The LOCK/LATCH field, shown in Figure 4-11, is supposed to contain the number of times the thread had to wait because some resource that it needed was held by another task in the *same* DB2 subsystem.

```
ELAPSED TIME DISTRIBUTION                                       CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------   ------------------------------------------------------------------
APPL   |=====> 11%                                              CPU    |======> 13%
DB2    |=============> 26%                                      NOTACC |========> 17%
SUSP   |==============================> 63%                     SUSP   |===================================> 71%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    -------------------   ------------  --------    --------------------------
ELAPSED TIME    0.043041    0.038216         N/P    LOCK/LATCH(DB2+IRLM)      0.009890      1.28    #OCCURRENCES    :    26764
 NONNESTED      0.043041    0.038216         N/A    SYNCHRON. I/O             0.002039      1.96    #ALLIEDS        :    26764
 STORED PROC    0.000000    0.000000         N/A     DATABASE I/O             0.000497      1.34    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A     LOG WRITE I/O            0.001541      0.63    #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O            0.000051      0.19    #DBATS DISTRIB. :        0
```

Figure 4-11 DB2PE reporting of Lock/Latch wait times

The most significant factor in the size of this value is how long those other tasks hold that resource for. This is a perfect example of the secondary effects of distance in that the lock or latch request itself may not require access to either disk or the CF, but increased response times for either of those components may result in increased elapsed time for whoever is holding the lock or latch that this transaction is trying to obtain. This in turn increases the time this transaction must wait for those resources to be released. If every transaction looked for a different lock or latch (not a realistic scenario), then conceptually there would be no change in lock/latch time regardless of how long each transaction ran.

The difficulty is in trying to predict how increased distance will affect the level of competition for these locks and latches. A high level of competition means that you will probably see large increases as average response times increase, while little competition would indicate smaller increases. Unfortunately, there is no tool or formula or methodology that will let you accurately calculate how the combination of distance and contention will impact your transaction and batch elapsed times.

As discussed in 4.2.2, “DB2 Class 3 Suspension” on page 59, there is an additional consideration for Lock/Latch. While Lock/Latch is *supposed* to reflect contention from another transaction in this DB2, and Global Contention is *supposed* to reflect contention from a transaction in another DB2, it is not always that clear-cut. The amount of time that a thread spends waiting for a resource is assigned to a category based on who is immediately “in front” of that thread in the queue to serialize on the resource. If the requestor ahead of this thread is in the same DB2, then all the wait time will be reported as Lock/Latch. If the requestor is in another DB2 in the data sharing group, then the time will be reported as Global Contention.

This means that the type of contention that is reported for a given workload can vary, depending on the relationship between the transactions running at any given instant. For this reason, we strongly recommend that you add your Lock/Latch and Global Contention wait times together, and address the cause of the contention, without getting overly concerned about which type of contention is being reported.

Synchronous I/O

Another part of the DB2 response time we want to focus on is the time spent waiting for synchronous I/O suspensions to complete. In general, most of these will be read I/Os for individual pages that are not found in either the local or global buffer pools. They can also be synchronous write I/Os (usually for tables that are not GBP-dependent).

Also, certain events will trigger synchronous log writes. For example, updated pages that must be written back to the Group Buffer Pool before the commit (after P-lock negotiation, for example), must be logged to disk before the GBP write can be done. Events such as CI splits in an index can also force synchronous log writes. These are recorded in this category. Most SQL update activity, however, only triggers writes into the DB2 log buffer, not to disk; the write to disk is done later, asynchronously to the thread.

To get the average amount of time a thread waits for processing related to synchronous I/O, we look at the DB2PE Accounting report. In Figure 4-12 you can see the field `SYNCHRON I/O`. It contains a value of .002039 seconds, or about 2 ms.

The average number of events was 1.96, meaning that each synchronous I/O request took, from the DB2 point of view, about 1 ms.

```
ELAPSED TIME DISTRIBUTION                                       CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------   ------------------------------------------------------------------
APPL   |=====> 11%                                              CPU    |======> 13%
DB2    |=============> 26%                                      NOTACC |========> 17%
SUSP   |================================> 63%                   SUSP   |====================================> 71%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.043041    0.038216         N/P    LOCK/LATCH(DB2+IRLM)      0.009890      1.28    #OCCURRENCES    :    26764
 NONNESTED      0.043041    0.038216         N/A    SYNCHRON. I/O             0.002039      1.96    #ALLIEDS        :    26764
 STORED PROC    0.000000    0.000000         N/A     DATABASE I/O             0.000497      1.34    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A     LOG WRITE I/O            0.001541      0.63    #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O            0.000051      0.19    #DBATS DISTRIB. :        0
```

Figure 4-12 DB2PE reporting of synchronous I/O times

The SYNCHRON I/O field is further broken down into DATABASE I/O and LOG WRITE I/O fields. Note that because our workload was using single-phase commit (DB2 is the only resource manager that has done updates), the LOG WRITE I/O field does not include the time to write the commit log records—the time to do that log I/O is included in Update Commit time. If CICS had been doing a two-phase commit, the writes to the log in phase 1 of commit processing would be reported in LOG WRITE I/O (only phase 2 is reported in UPDATE-COMMIT in that case).

These fields also contain the time for DB2 to do additional processing related to the I/O—that is, this field includes more than just the time to issue the I/O request. As a result, you should expect that the I/O times you see here will be somewhat longer than the corresponding disk response times you will see in RMF. DB2 reports the time the thread was suspended waiting for the synchronous I/O activity to complete. The suspend time includes, but is not limited to, the time it takes to execute the actual I/O operations.

Additionally, allowance needs to be made for the fact that DB2 normally runs with dual log data sets. Some log writes get driven to both log data sets in parallel, and others are done serially (to one log data set first, then to the other). The decision about how DB2 will manage a particular log write I/O is very complex and is not reported in the DB2 PE. However, you can get an indication from the time DB2 PE reports for the I/Os, and the time RMF reports for the corresponding devices. You would typically expect to see DB2 PE reporting times that are a little more than the RMF times, up to a little more than double the RMF time (indicating serial log I/Os).

Global Contention

The `GLOBAL CONTENTION` field contains information about delays due to a conflict between different DB2 members trying to get control of the same resource. This can happen for logical locks, which represent the interest of threads in accessing a specific piece of data, or for physical locks (P-locks), which are mainly used to track inter-DB2 read/write interest and to ensure physical consistency of data (and index) pages.

Although the original purpose of this field was to report on actual cross-DB2 contentions, the advent of the XES heuristic algorithm introduced by z/OS 1.2 meant that this field became difficult to interpret. A lock request converted by z/OS from being synchronous to asynchronous does not complete immediately; this is perceived and accounted by DB2 as contention, even if it is not.

For this reason, as soon as XES determines that it is better to issue CF lock requests asynchronously, the global contention indicator starts including delays that are not actual contentions, but simply time spent waiting for a lock request to complete asynchronously. Keep this in mind when you look at the global contention values, because they will increase significantly whenever CF response times increase, thus making the use of asynchronous lock requests predominant.

Tip: In order to have P-lock contention correctly attributed to global contention, ensure you have DB2 APAR PK27934 applied.

```
ELAPSED TIME DISTRIBUTION                                       CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------   ------------------------------------------------------------
APPL   |=====> 11%                                              CPU    |======> 13%
DB2    |=============> 26%                                      NOTACC |========> 17%
SUSP   |================================> 63%                   SUSP   |==================================> 71%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)   CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT   HIGHLIGHTS
------------  ----------  ----------  ----------   --------------------  ------------  --------   --------------------------
ELAPSED TIME    0.043041    0.038216         N/P   LOCK/LATCH(DB2+IRLM)      0.009890      1.28   #OCCURRENCES    :    26764
 NONNESTED      0.043041    0.038216         N/A   SYNCHRON. I/O             0.002039      1.96   #ALLIEDS        :    26764
 STORED PROC    0.000000    0.000000         N/A    DATABASE I/O             0.000497      1.34   #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A    LOG WRITE I/O            0.001541      0.63   #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A   OTHER READ I/O            0.000051      0.19   #DBATS DISTRIB. :        0
                                                   OTHER WRTE I/O            0.000000      0.00   #NO PROGRAM DATA:    26764
CPU TIME        0.006033    0.004818         N/P   SER.TASK SWTCH            0.007373      1.08   #NORMAL TERMINAT:    26764
 AGENT          0.006033    0.004818         N/A    UPDATE COMMIT            0.007312      1.08   #ABNORMAL TERMIN:        0
  NONNESTED     0.006033    0.004818         N/P    OPEN/CLOSE               0.000000      0.00   #CP/X PARALLEL. :        0
  STORED PRC    0.000000    0.000000         N/A    SYSLGRNG REC             0.000000      0.00   #IO PARALLELISM :        0
  UDF           0.000000    0.000000         N/A    EXT/DEL/DEF              0.000061      0.00   #INCREMENT. BIND:        0
  TRIGGER       0.000000    0.000000         N/A    OTHER SERVICE            0.000000      0.00   #COMMITS        :    28716
 PAR.TASKS      0.000000    0.000000         N/A   ARC.LOG(QUIES)            0.000000      0.00   #ROLLBACKS      :      293
                                                   ARC.LOG READ              0.000000      0.00   #SVPT REQUESTS  :        0
SUSPEND TIME    0.000000    0.026958         N/A   DRAIN LOCK                0.000000      0.00   #SVPT RELEASE   :        0
 AGENT              N/A     0.026958         N/A   CLAIM RELEASE             0.000000      0.00   #SVPT ROLLBACK  :        0
 PAR.TASKS          N/A     0.000000         N/A   PAGE LATCH                0.000126      0.08   MAX SQL CASC LVL:        0
 STORED PROC    0.000000        N/A          N/A   NOTIFY MSGS               0.000000      0.00   UPDATE/COMMIT   :    23.02
 UDF            0.000000        N/A          N/A   GLOBAL CONTENTION         0.007475     12.15   SYNCH I/O AVG.  : 0.001038
```

Figure 4-13 Getting global contention value for TPNO transaction

Because the time spent for lock requests that become converted to asynchronous requests by XES is incorrectly reported by DB2 as global contention, we have to make some adjustments to the number reported by DB2PE in order to calculate the *actual* global contention experienced by our transaction.

If we move down to the next section of the DB2PE Accounting report, as shown in Figure 4-14, we see that each thread issued an average of 65.68 global lock requests. This is the sum of all the lock requests (53.79) and change requests (11.89) externalized to XES. Some of these requests may have been sent to the CF synchronously, and some asynchronously.

```
PRIMAUTH: CICSUSER  PLANNAME: TPCCP2NO

DATA SHARING          AVERAGE     TOTAL
--------------------  --------  --------
L-LOCKS XES RATE(%)      38.12       N/A
LOCK REQ   - PLOCKS      25.41    680042
UNLOCK REQ - PLOCKS      25.24    675410
CHANGE REQ - PLOCKS       0.00         3
LOCK REQ   - XES         53.79   1439733
UNLOCK REQ - XES         27.28    729998
CHANGE REQ - XES         11.89    318153
SUSPENDS   - IRLM         0.42     11177
SUSPENDS   - XES          0.04      1108
SUSPENDS   - FALSE         N/A       N/A
INCOMPATIBLE LOCKS        0.00         0
NOTIFY MSGS SENT          0.00        88
```

Figure 4-14 Getting average number of lock requests per TPNO transaction

In Figure 4-14, we can see that the same section of the DB2PE Accounting report also shows the number of suspensions encountered due to actual global contention (but not the elapsed time related to these suspensions). It is the sum of:

- The `SUSPENDS - IRLM` field, which reports the number of contentions due to an incompatible IRLM resource state, plus
- The `SUSPENDS - XES` field, which reports the number of contentions due to an incompatible XES resource state. An incompatible XES resource state (also known as “XES

contention") is where XES thinks that there is real contention on a resource, but DB2 has determined that it is acceptable for the requests to be granted.

In this case, the average number of global contentions per thread is 0.46 (.42+.04). This means that, on average, only about 0.71% (.46 divided by 65.68) of all lock requests issued by each thread encountered an actual global contention.

In order to calculate the actual time spent waiting for real global contention, we need to subtract out the time spent waiting for asynchronous lock requests. To do this, we adjust the reported number of global contention events (12.15 in this example) by the number of events that actually were contention events (.46 in this example). This gives us the number of asynchronous lock requests that we reported as being global contention. We then multiply this value by the average response time for an asynchronous lock request (we get this from the RMF CF report as shown in Figure 4-15).

```
STRUCTURE NAME = DB8QU_LOCK1       TYPE = LOCK   STATUS = ACTIVE
          # REQ    -------------- REQUESTS -------------   -------------- DELAYED REQUESTS -------------
SYSTEM    TOTAL             #     % OF  -SERV TIME(MIC)-    REASON    #   % OF  ---- AVG TIME(MIC) -----  EXTERNAL REQUEST
NAME      AVG/SEC         REQ     ALL     AVG   STD_DEV            REQ    REQ   /DEL    STD_DEV   /ALL   CONTENTIONS

#@$2       3553K   SYNC   2124K   29.6    21.7      5.4     NO SCH  323    0.0    8.1        8.9    0.0   REQ TOTAL     4299K
           5921    ASYNC  1428K   19.9    61.4     43.9     PR WT     0    0.0    0.0        0.0    0.0   REQ DEFERRED    58K
                   CHNGD     0     0.0  INCLUDED IN ASYNC    PR CMP    0    0.0    0.0        0.0    0.0   -CONT           38K
                                                                                                          -FALSE CONT     13K

#@$3       3620K   SYNC   2205K   30.7    22.3      6.6     NO SCH  281    0.0    8.1        7.3    0.0   REQ TOTAL     4372K
           6033    ASYNC  1415K   19.7    63.4     41.3     PR WT     0    0.0    0.0        0.0    0.0   REQ DEFERRED    58K
                   CHNGD     0     0.0  INCLUDED IN ASYNC    PR CMP    0    0.0    0.0        0.0    0.0   -CONT           38K
                                                                                                          -FALSE CONT     12K

-------------------------------------------------------------------------------------------------------------------------------
TOTAL      7172K   SYNC   4329K   60.4    22.0      6.0     NO SCH  604    0.0    8.1        8.2    0.0   REQ TOTAL     8671K
          11954    ASYNC  2844K   39.6    62.4     42.7     PR WT     0    0.0    0.0        0.0    0.0   REQ DEFERRED   116K
                   CHNGD     0     0.0                      PR CMP    0    0.0    0.0        0.0    0.0   -CONT           76K
                                                                                                          -FALSE CONT     25K
```

Figure 4-15 Response time for CF lock requests

The formula is:

```
actual global contention = CLASS-3 GLOB CONT - ((SUSPENDS-IRLM + SUSPENDS-XES)
* asynch resptime)
```

In our case:

```
actual global contention = 0.007475 - ((53.79 + 11.89) * 62.4/1000000) =
0.006745
```

Therefore, by factoring out the average amount of time spent waiting for lock requests that were converted to be asynchronous, we can see that the average amount of global contention time per thread is actually 6.745 ms in this case.

Tip: If we look at the RMF Structure Activity report for a DB2 lock structure, we can see some interesting information: the number reported on the *right* side of the report, in the `REQ TOTAL` field under `EXTERNAL REQUEST`, is the number of lock or unlock requests sent to XES by IRLM. The number on the *left* side of the report, under `# REQ TOTAL`, is the number of those requests that were actually sent to the Coupling Facility. The difference between the two can be quite large. This difference is accounted for by:

- Batch unlock requests.

  When DB2 wants to release many locks at one time, it sends a single IXLLOCK request to XES, providing a list of locks to be released. Each lock to be released increments the External Request count by one. But many locks can be released by a single request to the Coupling Facility. So, a lot of batch unlock activity (typical of a batch environment) can result in a larger count on the right side of the report than on the left.

- Contention.

  If a lock entry is already known to be in contention when a lock request is received by XES, the request will be passed to the global lock manager for that lock entry, rather than being passed to the Coupling Facility. So, as the level of lock contention increases, you would expect to see a growing difference between the number of lock requests sent to XES (on the right side of the report) and the number sent to the lock structure (on the left side).

When discussing distance, we need to remember that a part of the total time a lock is held for is the amount of time it takes for the CF unlock request to be processed. In this measurement, the overall response time for requests to the DB2 lock structure was 38 microseconds, as shown in the extract from the RMF Structure Activity report in Figure 4-15 on page 75. We arrive at this number by averaging the sync (4329 K requests at 22 mics) and asynch (2844 K requests at 62.4 mics).

Therefore, considering that every 20 km increment is going to increase CF response time from the remote system by at least 200 microseconds, we expect the global contention time to also increase as the distance in the sysplex increases.

Tip: Because it is difficult to predict whether the time that a thread is suspended because it is waiting for a resource will be reported as Lock/Latch time or as global contention, we strongly recommend that you focus less on the differentiation between the different types of contention, and simply treat them as one combined number (delays because of contention) and try to address the cause of the contention. As you will see in our measurements in 6.8, “Measurement at 80 km with index” on page 169, when we addressed the cause of our contention, both Lock/Latch *and* global contention times decreased.

Asynch CF Requests

The `ASYNCH CF REQUESTS` field, as shown in Figure 4-16 on page 77, contains information about the time spent by DB2 threads waiting for read or write access to data into a Global Buffer Pool. For transactions using single-phase commit, writes executed at commit time happen under a different task than the thread’s one; for this reason, the time spent waiting for them is included in the Update Commit time and is not included in the Asynch CF Requests category.

DB2 members in a data sharing group try to keep their local buffer pools full with the data needed by the running threads to proceed. Nevertheless, sometimes they need to get fresh data from a Global Buffer Pool. This can happen if a local copy of the needed page exists but is not valid anymore (because another member updated it and it was cross-invalidated), or

because it simply does not exist in the local buffer pool. In this case, DB2 has to check to see if the GBP contains a valid copy of the latest version of the requested data. If the heuristic algorithm causes XES to issue the read operation asynchronously, this causes a thread suspension which DB2 accounts for in the Asynch CF Requests field.

In order to have accurate reporting of the number and duration of asynchronous GBP requests, ensure you have APAR PK27934 applied.

```
PRIMAUTH: CICSUSER  PLANNAME: TPCCP2NO

ELAPSED TIME DISTRIBUTION                                    CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------  ------------------------------------------------------------------
APPL   |=====> 11%                                           CPU    |======> 13%
DB2    |=============> 26%                                   NOTACC |========> 17%
SUSP   |==============================> 63%                  SUSP   |==================================> 71%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)   CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT   HIGHLIGHTS
------------  ----------  ----------  ----------   --------------------  ------------  --------   --------------------------
ELAPSED TIME    0.043041    0.038216         N/P   LOCK/LATCH(DB2+IRLM)      0.009890      1.28   #OCCURRENCES    :    26764
 NONNESTED      0.043041    0.038216         N/A   SYNCHRON. I/O             0.002039      1.96   #ALLIEDS        :    26764
 STORED PROC    0.000000    0.000000         N/A    DATABASE I/O             0.000497      1.34   #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A    LOG WRITE I/O            0.001541      0.63   #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A   OTHER READ I/O            0.000051      0.19   #DBATS DISTRIB. :        0
                                                   OTHER WRTE I/O            0.000000      0.00   #NO PROGRAM DATA:    26764
CPU TIME        0.006033    0.004818         N/P   SER.TASK SWTCH            0.007373      1.08   #NORMAL TERMINAT:    26764
 AGENT          0.006033    0.004818         N/A    UPDATE COMMIT            0.007312      1.08   #ABNORMAL TERMIN:        0
  NONNESTED     0.006033    0.004818         N/P    OPEN/CLOSE               0.000000      0.00   #CP/X PARALLEL. :        0
  STORED PRC    0.000000    0.000000         N/A    SYSLGRNG REC             0.000000      0.00   #IO PARALLELISM :        0
  UDF           0.000000    0.000000         N/A    EXT/DEL/DEF              0.000061      0.00   #INCREMENT. BIND:        0
  TRIGGER       0.000000    0.000000         N/A    OTHER SERVICE            0.000000      0.00   #COMMITS        :    28716
 PAR.TASKS      0.000000    0.000000         N/A   ARC.LOG(QUIES)            0.000000      0.00   #ROLLBACKS      :      293
                                                   ARC.LOG READ              0.000000      0.00   #SVPT REQUESTS  :        0
SUSPEND TIME    0.000000    0.026958         N/A   DRAIN LOCK                0.000000      0.00   #SVPT RELEASE   :        0
 AGENT              N/A    0.026958         N/A   CLAIM RELEASE             0.000000      0.00   #SVPT ROLLBACK  :        0
 PAR.TASKS          N/A    0.000000         N/A   PAGE LATCH                0.000126      0.08   MAX SQL CASC LVL:        0
 STORED PROC    0.000000         N/A         N/A   NOTIFY MSGS               0.000000      0.00   UPDATE/COMMIT   :    23.02
 UDF            0.000000         N/A         N/A   GLOBAL CONTENTION         0.007475     12.15   SYNCH I/O AVG.  : 0.001038
                                                   COMMIT PH1 WRITE I/O      0.000000      0.00
NOT ACCOUNT.        N/A    0.006441         N/A   ASYNCH CF REQUESTS        0.000004      0.03
```

Figure 4-16 DB2PE reporting of Asynchronous CF Request

Unfortunately, the ASYNCH CF REQUESTS field does not report how many of the requests were reads, and how many were writes. Reads will typically have better performance because they are not duplexed, and they are not delayed by cross-invalidate processing as writes might be. When we get to the stage of trying to predict the change in the ASYNCH CF REQUESTS time, you will see that the lack of this level of detail limits the accuracy of the predictions.

Another limitation of the ASYNCH CF REQUESTS counter is that it counts all asynchronous CF requests prior to commit. So, if a GBP write request is sent asynchronously to both the primary and secondary GBPs (because of the XES heuristic algorithm), this will actually count as two events in the ASYNCH CF REQUESTS counter. This again impedes our ability to accurately predict the impact of changing GBP response times on thread suspension time when there are asynchronous writes to the GBPs before commit (because those writes will be double-counted in this field).

Note: In our measurements, nearly all the GBP accesses by the transaction we focused on were reads, meaning that the possible double-counting of writes was not an issue for us.

Update Commit time

The `Update Commit` time, as reported in the DB2PE Accounting report (shown in Figure 4-17), includes the time required for the following events:

- If single phase commit is being used, log writes to the primary and secondary DB2 log data sets.

  If two phase commit *and* DB2 V8 or later are being used, this happens during commit phase 1, and will be reported in LOG WRITE I/O. The field COMMIT PH1 WRITE I/O, which you might expect to contain this information, is actually only used for DB2 Large Objects (LOBs).

- If single-phase commit is being used, sending all updated pages included in the scope of the COMMIT to the primary and secondary GBP structures[2].

  If two-phase commit *and* DB2 V8 or later are being used, the GBP writes happen during commit phase 1 and are reported in ASYNCH CF REQUESTS.

- Releasing all locks associated with the unit of work.
- DB2 cleanup associated with that unit of work.

In the sample report shown in Figure 4-17, the Update Commit time was about 7 ms. To get a better insight into how this will change as the distance increases, we break the Update Commit time down into its components and see which of those are likely to increase.

```
PRIMAUTH: CICSUSER  PLANNAME: TPCCP2NO

ELAPSED TIME DISTRIBUTION                                    CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------  ------------------------------------------------------------------------
APPL   |=====> 11%                                           CPU    |======> 13%
DB2    |============> 26%                                    NOTACC |========> 17%
SUSP   |==============================> 63%                  SUSP   |===================================> 71%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.043041    0.038216         N/P    LOCK/LATCH(DB2+IRLM)      0.009890      1.28    #OCCURRENCES    :    26764
 NONNESTED      0.043041    0.038216         N/A    SYNCHRON. I/O             0.002039      1.96    #ALLIEDS        :    26764
 STORED PROC    0.000000    0.000000         N/A     DATABASE I/O             0.000497      1.34    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A     LOG WRITE I/O            0.001541      0.63    #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A    OTHER READ I/O            0.000051      0.19    #DBATS DISTRIB. :        0
                                                    OTHER WRTE I/O            0.000000      0.00    #NO PROGRAM DATA:    26764
CPU TIME        0.006033    0.004818         N/P    SER.TASK SWTCH            0.007373      1.08    #NORMAL TERMINAT:    26764
 AGENT          0.006033    0.004818         N/A     UPDATE COMMIT            0.007312      1.08    #ABNORMAL TERMIN:        0
```

Figure 4-17 Update Commit time in DB2PE report

Log writes

Because Update Commit consists of some writes to the DB2 log, we first look at how much time these writes are taking. To see the average response time for the I/Os to write the COMMIT records to the log data sets, we ran an RMF Workload Activity report for the DB2 MSTR address spaces. Most of the I/Os done by the MSTR address space are for logging, so by looking at the I/O rate (reported as SSCHRT) and the response time (reported as RESP) in the RMF report, we can understand the impact of the log write I/Os.

Figure 4-18 on page 79 contains the report for the D8Q1 DB2 MSTR address space.

[2] If the requests to the Primary GBP are sent asynchronously by XES, UPDATE/COMMIT will include the entire time to write to the Primary and Secondary GBPs. However, if the requests to the Primary GBP are sent synchronously by XES, they will not cause suspension and will not be accounted in UPDATE/COMMIT. In this case, the time spent processing the synchronous request to the primary structure will be added to the thread's CPU time.

```
REPORT BY: POLICY=WLMPOL                        REPORT CLASS=RD8Q1MST                                PERIOD=1
                                                HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS SYSSTC

TRANSACTIONS    TRANS.-TIME  HHH.MM.SS.TTT   --DASD I/O--   ---SERVICE----   --SERVICE TIMES--   PAGE-IN RATES    ----STORAGE----
AVG      1.00   ACTUAL                   0   SSCHRT 343.9   IOC     26380    TCB           0.1   SINGLE     0.0   AVG     5242.94
MPL      1.00   EXECUTION                0   RESP     0.6   CPU      4033    SRB          47.1   BLOCK      0.0   TOTAL   5242.92
ENDED       0   QUEUED                   0   CONN     0.2   MSO     47986    RCT           0.0   SHARED     0.0   CENTRAL 5242.92
END/S    0.00   R/S AFFINITY             0   DISC     0.3   SRB      1336K   IIT           0.9   HSP        0.0   EXPAND     0.00
#SWAPS      0   INELIGIBLE               0   Q+PEND   0.1   TOT      1415K   HST           0.0   HSP MISS   0.0
EXCTD       0   CONVERSION               0   IOSQ     0.0   /SEC     2620    IFA           N/A   EXP SNGL   0.0   SHARED     0.00
AVG ENC  0.00   STD DEV                  0                                    APPL% CP      8.9   EXP BLK    0.0
REM ENC  0.00                                              ABSRPTN  2620    APPL% IFACP   0.0   EXP SHR    0.0
MS ENC   0.00                                              TRX SERV 2620    APPL% IFA     N/A
```

Figure 4-18 Workload Activity report for D8Q1 MSTR address space

The corresponding information for the D8Q2 DB2 MSTR address space is shown in Figure 4-19.

```
REPORT BY: POLICY=WLMPOL                        REPORT CLASS=RD8Q2MST                                PERIOD=1
                                                HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS SYSSTC

TRANSACTIONS    TRANS.-TIME  HHH.MM.SS.TTT   --DASD I/O--   ---SERVICE----   --SERVICE TIMES--   PAGE-IN RATES    ----STORAGE----
AVG      1.00   ACTUAL                   0   SSCHRT 340.6   IOC     26044    TCB           0.1   SINGLE     0.0   AVG     2995.55
MPL      1.00   EXECUTION                0   RESP     0.6   CPU      4046    SRB          47.5   BLOCK      0.0   TOTAL   2995.55
ENDED       0   QUEUED                   0   CONN     0.2   MSO     27625    RCT           0.0   SHARED     0.0   CENTRAL 2995.55
END/S    0.00   R/S AFFINITY             0   DISC     0.3   SRB      1348K   IIT           0.8   HSP        0.0   EXPAND     0.00
#SWAPS      0   INELIGIBLE               0   Q+PEND   0.1   TOT      1406K   HST           0.0   HSP MISS   0.0
EXCTD       0   CONVERSION               0   IOSQ     0.0   /SEC     2604    IFA           N/A   EXP SNGL   0.0   SHARED     0.00
AVG ENC  0.00   STD DEV                  0                                    APPL% CP      9.0   EXP BLK    0.0
REM ENC  0.00                                              ABSRPTN  2604    APPL% IFACP   0.0   EXP SHR    0.0
MS ENC   0.00                                              TRX SERV 2604    APPL% IFA     N/A
```

Figure 4-19 Workload Activity report for D8Q2 MSTR address space

You can see that, in this example, both MSTR address spaces were doing a similar amount of logging (based on the SSCHRT values), and that both were seeing an average disk response time of about .6 ms. This includes the additional time needed by PPRC to mirror each write operation. Allowing for writes to both the primary and secondary log data sets, that accounts for roughly 1.2 of the 7 ms. Update Commit time. Remember, though, that the DB2 time includes significant processing in addition to the actual I/O, so the time that DB2 sees for issuing the log writes will, in practice, be higher than 1.2 ms.

GBP writes

The next component of Update Commit time is the time required to write the committed updates to the primary and secondary GBPs. We know that all GBP requests in this category were writes; however, we do not know how many there were. Without knowing the number of GBP requests that are included in the Update Commit time, it is not possible to accurately predict how the GBP component of Update Commit will change with distance.

However, there is another way to approach this, as we explain next.

For all transactions, there are three types of GBP accesses, from the perspective of DB2PE:

- There are asynchronous requests that are issued before the transaction issues a commit.

  These are included in the ASYNCH CF REQUESTS field. We know how many of those there are, but we do not know if they are reads or writes, and there may be an amount of double-counting (if ASYNCH CF REQUESTS includes writes where both the requests to both primary and secondary structures were sent asynchronously).

- There are asynchronous requests that are issued as a part of commit processing.

  If the transaction is using single-phase commit, these are included in the Update Commit time. If the transaction is using two-phase commit, these requests are included in the ASYNCH CF REQUESTS category.

  DB2 does not report anywhere exactly how long these requests took, or how many of them there were.

- And, finally, there are synchronous GBP requests. The time to process them is included in the Class 2 CPU time; however, the number of synchronous requests is not directly reported anywhere.

All these types of requests will cause the thread to be delayed. Fortunately, the GBP section of the DB2 Accounting report, as shown in Figure 4-20, reports *all* the GBP accesses that delayed the thread. It has the added benefit that it only reports on the number or reads and writes to the primary structures, so there is no concern about double-counting asynchronous requests.

So, rather than trying to predict the change in the ASYNCH CF REQUESTS time *and* a part of the Update Commit time, *and* then trying to make some estimate and adjustment for synchronous requests, we recommend simply using this report to determine the total number of GBP requests that delayed the thread.

```
PRIMAUTH: CICSUSER  PLANNAME: TPCCP2NO

GROUP TOT4K              AVERAGE     TOTAL
---------------------  --------  --------
GBP-DEPEND GETPAGES       171.59   4611359
READ(XI)-DATA RETUR        10.06    270481
READ(XI)-NO DATA RT         0.00         1
READ(NF)-DATA RETUR        12.49    335669
READ(NF)-NO DATA RT         1.41     38017
PREFETCH PAGES READ         0.00         8
CLEAN PAGES WRITTEN         0.00         0
UNREGISTER PAGE             0.14      3784
ASYNCH GBP REQUESTS        34.93    938737
EXPLICIT X-INVALID          0.00         0
ASYNCH SEC-GBP REQ          0.00         0
PG P-LOCK LOCK REQ         24.14    648614
 SPACE MAP PAGES            1.72     46177
 DATA PAGES                 0.00         0
 INDEX LEAF PAGES          22.42    602437
PG P-LOCK UNLOCK REQ       24.81    666694
PG P-LOCK LOCK SUSP        21.56    579343
 SPACE MAP PAGES            0.63     16971
 DATA PAGES                 0.00         0
 INDEX LEAF PAGES          20.93    562372
WRITE AND REGISTER          8.12    218337
WRITE & REGISTER MULT       4.40    118307
CHANGED PAGES WRITTEN      37.59   1010163
WRITE TO SEC-GBP            N/A       N/A
```

Figure 4-20 GBP summary section of DB2PE Accounting report - Subsystem D8Q2

Using a combination of this report and an RMF CF report for the GBP structures (shown in Figure 4-21 on page 83 and Figure 4-22 on page 83), we can develop a reasonably accurate picture of the impact of GBP processing related to this thread. If we start with the GBP section, we can extrapolate the following information:

- There were roughly 12.5 write requests to the GBPs (the WRITE AND REGISTER and WRITE & REGISTER MULT fields).
- There were roughly 24 read and unregister requests.
- Combining the read, unregister, and write requests, this gives a total of about 36.5 GBP requests per transaction.

- Of these, just under 35 were asynchronous requests (ASYNCH GBP REQUESTS), meaning that about 1.5 requests were synchronous (on average).

This tells us the total number of GBP requests that caused the thread to be delayed, and how many of those were reads and how many were writes. The next step is to determine the approximate response times for the different types of requests (reads and writes). Unfortunately, RMF does not report on read and write requests separately, so we need to do a little work to arrive at these numbers.

The first thing we want to do is find out which GBPs are used by the transaction in question. If nearly all the requests go to a single GBP, then we are lucky, and can focus on just that structure. If the requests are spread around a number of GBPs, then we need to get the weighted average response time for those structures.

Note: When using duplexed GBP structures (as we hope all installations are), all reads are sent to the primary structure, and writes are sent to both structures (in parallel).

For a write, DB2 sends an asynchronous request to the secondary structure, followed by a synchronous request to the primary (which may get converted to an asynchronous request by the XES heuristic algorithm). The write request may involve a cross-invalidate being sent to other DB2s, which could be in either site.

Which of these write requests takes longer to complete will typically be determined by which structure instance is the furthest from the requesting DB2. The important point is that the thread will not proceed until *both* writes complete.

The difference between read and write behavior, and the difference resulting from the different distances between the DB2s and the primary and secondary structures, mean that we have four different response times to calculate:

- For reads from the DB2 that is beside the primary GBP, we recommend using the response time for that structure when all the components are close together. This response time will include writes and cross-invalidates. However, at zero distance, the difference in response time for reads and writes will be minimal.
- For reads from the DB2 that is remote to the primary GBP, use the response time number from RMF for the system that DB2 resides on.
- For writes from the DB2 that is remote to the secondary GBP, use the longer of the response times for the primary and secondary GBPs.
- For writes from the DB2 that is in the same site as the secondary GBP, we recommend using the response time that is being achieved by the DB2 that is remote to the secondary GBP (that is, the *other* DB2). Our reasoning for this is as follows:
  - It is likely that the writes to the primary GBP (in the remote site to this DB2) will be taking longer than the writes to the secondary GBP, which is right beside DB2. And it is the longer of the two writes that we want to focus on.
  - Writes to the primary GBP will actually take longer than the response time reported by RMF (because the RMF number includes both reads and writes).
  - On the other hand, the response time for the secondary GBP *only* includes writes.
  - Conceptually, the performance of a write from a DB2 in site2 to the primary GBP in site1 should be similar to the performance of a write from DB2 in site1 to the secondary GBP in site2.
  - So, by using the RMF numbers for the site 1 DB2 talking to the secondary GBP in site 2, we should be getting a very close estimate of the response time when DB2 in site 2 does a write to the primary GBP in site 1.

We validated this methodology against our actual results and found it to be within +/- 10% of the actual results in nearly every measurement. In Appendix B, “Predicting the impact of distance on DB2 data sharing” on page 193, we use this methodology when we talk about estimating the impact of distance on response times.

While we are on the topic of RMF reports and GBPs, we also want to point out some other interesting fields in the RMF Structure Activity report that are particularly relevant to DB2 GBPs; refer to Figure 4-21 on page 83 and Figure 4-22 on page 83.

```
STRUCTURE NAME = DB8QU_GBP3        TYPE = CACHE  STATUS = ACTIVE PRIMARY
             # REQ    ------------ REQUESTS ------------   -------------- DELAYED REQUESTS -------------
    SYSTEM   TOTAL            #     % OF  -SERV TIME(MIC)-    REASON    #   % OF  ---- AVG TIME(MIC) -----
    NAME     AVG/SEC          REQ    ALL     AVG   STD_DEV           REQ    REQ   /DEL    STD_DEV   /ALL

    #@$2       580K   SYNC   3650     0.3    76.6    101.9     NO SCH    0    0.0    0.0        0.0    0.0
             966.0    ASYNC   576K   50.3   130.5    123.7     PR WT     0    0.0    0.0        0.0    0.0
                      CHNGD     0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                             DUMP      0    0.0    0.0        0.0    0.0

    #@$3       565K   SYNC   4450     0.4    63.7     82.3     NO SCH    3    0.0   40.3       14.6    0.0
             941.8    ASYNC   561K   49.0   127.4    111.5     PR WT     0    0.0    0.0        0.0    0.0
                      CHNGD     3    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                             DUMP      0    0.0    0.0        0.0    0.0

  ------------------------------------------------------------------------------------------------------------------------
    TOTAL      1145K   SYNC   8100     0.7    69.5     91.9     NO SCH    3    0.0   40.3       14.6    0.0   -- DATA ACCESS ---
               1908    ASYNC  1137K   99.3   128.9    117.8     PR WT     0    0.0    0.0        0.0    0.0   READS       784409
                       CHNGD     3    0.0                      PR CMP    0    0.0    0.0        0.0    0.0   WRITES     1222493
                                                              DUMP      0    0.0    0.0        0.0    0.0   CASTOUTS     51803
                                                                                                           XI'S        431309
```

Figure 4-21 RMF Structure Activity report for primary GBP3 structure

```
STRUCTURE NAME = DB8QU_GBP3        TYPE = CACHE  STATUS = ACTIVE SECONDARY
             # REQ    ------------ REQUESTS ------------   -------------- DELAYED REQUESTS -------------
    SYSTEM   TOTAL            #     % OF  -SERV TIME(MIC)-    REASON    #   % OF  ---- AVG TIME(MIC) -----
    NAME     AVG/SEC          REQ    ALL     AVG   STD_DEV           REQ    REQ   /DEL    STD_DEV   /ALL

    #@$2       164K   SYNC      0     0.0     0.0      0.0     NO SCH    0    0.0    0.0        0.0    0.0
             272.7    ASYNC   164K   50.5   254.7    219.4     PR WT     0    0.0    0.0        0.0    0.0
                      CHNGD     0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                             DUMP      0    0.0    0.0        0.0    0.0

    #@$3       161K   SYNC      0     0.0     0.0      0.0     NO SCH    0    0.0    0.0        0.0    0.0
             267.8    ASYNC   161K   49.5   234.7    201.8     PR WT     0    0.0    0.0        0.0    0.0
                      CHNGD     0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                             DUMP      0    0.0    0.0        0.0    0.0

  ------------------------------------------------------------------------------------------------------------------------
    TOTAL      324K    SYNC      0     0.0     0.0      0.0     NO SCH    0    0.0    0.0        0.0    0.0   -- DATA ACCESS ---
               540.5   ASYNC   324K    100   244.8    211.1     PR WT     0    0.0    0.0        0.0    0.0   READS             0
                       CHNGD     0    0.0                      PR CMP    0    0.0    0.0        0.0    0.0   WRITES      1222578
                                                              DUMP      0    0.0    0.0        0.0    0.0   CASTOUTS          0
                                                                                                           XI'S         655853
```

Figure 4-22 RMF Structure Activity report for secondary GBP3 structure

There are a number of interesting things about the structure shown in these figures. First, because there was no distance involved in this measurement, the request rate and response time from each system was fairly similar, with system #@$2 being a little higher because this system was the castout owner[3].

Another interesting point is that the vast majority of requests to the primary GBP from both systems were converted to asynchronous requests. This confirms what we said in 2.2.1, “Synchronous and asynchronous CF requests” on page 9 about the increasing trend towards the majority of CF requests being sent asynchronously. It also correlates with our findings from the DB2PE GBP summary that only a very small percentage of all GBP requests were sent to the CF synchronously.

You might also notice, in Figure 4-21, that the number of Reads and Writes (reported on the right side of the report, under DATA ACCESS) comes to roughly 2,000,000, while the total

[3] Although you cannot really control which member of a data sharing group is the castout owner, we artificially forced the DB2 on system #@$2 to be the castout owner for all measurements, to minimize the number of variables from one run to the next.

number of requests to the GBP structure was only 1,145,000. The reason for the discrepancy is that DB2 V8 introduced the ability to write multiple pages to a GBP structure on a single IXLCACHE request. The numbers you see under DATA ACCESS represent the number of *pages* moved. In contrast, the numbers on the left side of the report under # REQ TOTAL represent the number of CF *requests* sent to the structure.

Note: Do not expect the total number of reads and writes to match the numbers from the DB2PE Accounting Report GBP summary report. For one thing, the summary report is only reporting on one plan (transaction). Typically, many different transactions will be using the same GBPs.

Also, the GBP summary report is only reporting on the GBP requests that resulted from a specific request by the transaction. If DB2 decided to move pages to the GBP because some internal DB2 threshold was reached, that processing would not affect the thread execution, and therefore would not be included in the Accounting report.

You will also see that the response time for the secondary GBP is considerably longer than for the primary GBP. This is partially explained by the fact that DB2 issues long-running CF commands to the secondary GBP to delete pages that have been cast out.

Another factor in the elongated response time of the secondary GBP was that DB2 V8 was actually replicating the directory information to the secondary GBP, meaning that cross-invalidates were being driven from the secondary GBP for every page that was deleted as part of the castout process. This behavior has been changed in DB2 V9 and is expected to have a beneficial impact on secondary GBP response times, especially in configurations where there is a considerable distance between the CF containing the secondary GBP structure and some of the connected DB2s.

Unlock requests

The time to assemble the information about which locks need to be released, and to create and issue the IXLLOCK request, is included in Update Commit Class 3 suspension time, but it is not specifically broken out into a separate field anywhere.

The IXLLOCK request that IRLM issues to release the locks during commit is issued as an asynchronous request, and the thread does not need to wait for the request to complete. As a result, this processing only takes a very small amount of time and will not change as the distance to the lock structure changes.

DB2 cleanup

After all the other commit processing has completed, DB2 then has to do some cleanup, removing control blocks associated with the transaction, and so on. All of this processing happens internally in DB2, so once again it is unaffected by any changes in the distance within the sysplex.

Not Account

The `Not Account` field is intended to be used by DB2 to report elapsed time spent by the transaction in DB2, when it is not consuming CPU or waiting for a known DB2 suspension reason. In DB2 terms, it is simply the difference between the Class 2 Elapsed time, which is the elapsed time spent by the transaction in DB2, and the sum of the Class 2 CPU Time and Class 3 Suspension, which is the total time spent by the transaction waiting for a known DB2 suspension reason. The events that are most likely to contribute to `Not Account` are listed in 4.2.2, “DB2 Class 3 Suspension” on page 59.

Prior to APAR PK27934, the Not Account field incorrectly contained some P-lock contention and some asynchronous GBP requests. Ensure you have this APAR applied to DB2 to improve the accuracy of this field.

4.4.2 Correlating RMF and DB2PE reports

As anyone familiar with performance monitoring will know, it is very unusual to get exactly the same number from two different products that are monitoring the same thing. Part of this may be due to small differences in precisely what is being measured. Part may be due to differences in how they gather the information (sampling versus measuring, for example). And part may be due to small differences in the interval being measured. These differences sometimes make it difficult to use a combination of products to get an accurate and consistent view of what is happening.

In this situation, we are effectively using three different monitors: RMF, DB2PE Accounting reports, and DB2PE Statistics reports. Most of the focus in this book is on how distance affects the response time of transactions in a multisite data sharing environment, so most of our work is with the DB2PE Accounting reports, which report on particular transactions.

In a DB2 environment, most of the work done by DB2 is in direct response to requests from user programs such as logging updates, reading database pages, writing updated pages to the GBPs, and so on. This activity is reported in the DB2PE Accounting report.

But there is also work done by DB2 itself that is asynchronous to transaction activity; an example is casting out changed pages from the GBP to disk. The DB2PE Statistics reports provide information about both of these activities; it reports on the totality of DB2 processing. As a result, it is generally possible to correlate the numbers in the DB2 Statistics report to the numbers reported by RMF.

However, it is generally not possible to correlate most of the numbers in the DB2PE Accounting report to those provided in the RMF Structure Activity report, because the Accounting report only contains information about a subset of DB2 activity. We have also found that the numbers in the DB2PE Statistics report can be less than the sum of those in the Accounting report, especially in subsystems with high CPU utilization, high numbers of threads, or when there is a large number of CPs in the z/OS image.

Figure 4-23 on page 86 shows how the locking numbers in the DB2PE Statistics report match up with the corresponding number in RMF. Although the DB2 numbers do not add up to exactly the same number as RMF provides, keep in mind that the DB2PE interval is unlikely to exactly line up with the RMF interval, so it would be unusual to find an exact match between the numbers. The figure concentrates on the lock requests sent from DB2 (via IRLM) to XES.

Another metric that is contained in the report, but not discussed in the figure, is the level of False Contention. At the time of writing, IRLM is unable to distinguish between a request that gets converted to asynchronous by XES. and one that encounters False Contention. Therefore, to be able to report on False Contention, IRLM issue a call to XES to obtain this information. Because RMF uses the same service, it is to be expected that DB2PE and RMF will present nearly identical False Contention numbers (allowing for the slight differences in intervals).

```
                                   C O U P L I N G   F A C I L I T Y   A C T I V I T Y
                                                                                                        PAGE   8
      z/OS V1R7              SYSPLEX #@$#PLEX          START 06/27/2006-16.33.00          INTERVAL 000.10.00
                             RPT VERSION V1R7 RMF      END   06/27/2006-16.43.00          CYCLE 01.000 SECONDS

   ------------------------------------------------------------------------------------------------------------------------------
    COUPLING FACILITY NAME = FACIL05
   ------------------------------------------------------------------------------------------------------------------------------
                                      COUPLING  FACILITY  STRUCTURE  ACTIVITY
   ------------------------------------------------------------------------------------------------------------------------------

    STRUCTURE NAME = DB8QU_LOCK1       TYPE = LOCK   STATUS = ACTIVE
              # REQ    -------------- REQUESTS -------------   -------------- DELAYED REQUESTS -------------
    SYSTEM    TOTAL               #     % OF  -SERV TIME(MIC)-    REASON     #    % OF  ---- AVG TIME(MIC) -----  EXTERNAL REQUEST
    NAME      AVG/SEC           REQ     ALL     AVG   STD_DEV              REQ     REQ    /DEL     STD_DEV   /ALL   CONTENTIONS

    #@$3       3579K   SYNC   2181K   61.0    22.2       6.6    NO SCH  255    0.0    8.2        7.5    0.0   REQ TOTAL      4314K
               5965    ASYNC  1397K   39.0    63.2      42.1    PR WT     0    0.0    0.0        0.0    0.0   REQ DEFERRED     58K
                       CHNGD      0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0   -CONT            39K
                                                                                                             -FALSE CONT      13K

   ------------------------------------------------------------------------------------------------------------------------------
    TOTAL      3579K   SYNC   2181K   61.0    22.2       6.6    NO SCH  255    0.0    8.2        7.5    0.0   REQ TOTAL      4314K
               5965    ASYNC  1397K   39.0    63.2      42.1    PR WT     0    0.0    0.0        0.0    0.0   REQ DEFERRED     58K
                       CHNGD      0    0.0                      PR CMP    0    0.0    0.0        0.0    0.0   -CONT            39K
                                                                                                            -FALSE CONT     13K

From DB2 Statistics Report:
DATA SHARING LOCKING           /SECOND                       SYNCH.XES - LOCK REQUESTS    +   2318.89 +
                                                             SYNCH.XES - CHANGE REQUESTS +      6.67 +
LOCK REQUESTS (P-LOCKS)        1137.22                       SYNCH.XES - UNLOCK REQUESTS +   2917.01 +
UNLOCK REQUESTS (P-LOCKS)      1136.1                        ASYNCH.XES - RESOURCES      +         0 +
CHANGE REQUESTS (P-LOCKS)         0.02                       SUSPENDS - FALSE CONT&CONV  -   1922.98 -
                                                             SUSPENDS - FALSE CONT.            20.90
SYNCH.XES - LOCK REQUESTS      2318.89
SYNCH.XES - CHANGE REQUESTS       6.67          This gives: TOTAL NUMBER OF XES REQUESTS per sec    7144.65 *
SYNCH.XES - UNLOCK REQUESTS    2917.01
ASYNCH.XES - RESOURCES            0             multiply by RMF interval                                600

SUSPENDS - IRLM GLOBAL CONT      22.69          Total XES REQUESTS according to DB2PE:            4286790
SUSPENDS - XES GLOBAL CONT.       2.4
SUSPENDS - FALSE CONT&CONV     1922.98          RMF reported IXLLOCK requests for interval:  REQ TOTAL  4314K
SUSPENDS - FALSE CONT.           20.9
INCOMPATIBLE RETAINED LOCK        0
```

Figure 4-23 Comparing locking information in RMF and DB2 Statistics reports

We can also look at the DB2 Statistics report for GBP structures and understand the relationship between the reported values in that report and in the RMF Structure Activity report.

GROUP BP3	QUANTITY	/SECOND	/THREAD	/COMMIT
GROUP BP R/W RATIO (%)	64.49	N/A	N/A	N/A
GBP-DEPENDENT GETPAGES	2057.8K	3383.57	39.56	37.60
SYN.READ(XI)-DATA RETURNED	139.6K	229.60	2.68	2.55
SYN.READ(XI)-NO DATA RETURN	0.00	0.00	0.00	0.00
SYN.READ(NF)-DATA RETURNED	251.9K	414.27	4.84	4.60
SYN.READ(NF)-NO DATA RETURN	1876.00	3.08	0.04	0.03
UNREGISTER PAGE	2859.00	4.70	0.05	0.05
CLEAN PAGES SYNC.WRITTEN	0.00	0.00	0.00	0.00
REG.PAGE LIST (RPL) REQUEST	1458.00	2.40	0.03	0.03
NUMBER OF PAGES RETR.FROM GBP	8184.00	13.46	0.16	0.15
PGS READ FRM DASD AFTER RPL	N/A	N/A	N/A	N/A
ASYNC.READ-DATA RETURNED	N/A	N/A	N/A	N/A
PAGES CASTOUT	37055.00	60.93	0.71	0.68
UNLOCK CASTOUT	1108.00	1.82	0.02	0.02
READ CASTOUT CLASS	244.00	0.40	0.00	0.00
READ DIRECTORY INFO	0.00	0.00	0.00	0.00
READ STORAGE STATISTICS	283.00	0.47	0.01	0.01
REGISTER PAGE	6916.00	11.37	0.13	0.13
DELETE NAME	0.00	0.00	0.00	0.00
ASYNCH GBP REQUESTS	574.5K	944.71	11.04	10.50
EXPLICIT X-INVALIDATIONS	0.00	0.00	0.00	0.00
CASTOUT CLASS THRESHOLD	1.00	0.00	0.00	0.00
GROUP BP CASTOUT THRESHOLD	0.00	0.00	0.00	0.00
GBP CHECKPOINTS TRIGGERED	0.00	0.00	0.00	0.00
CASTOUT ENGINE NOT AVAIL.	N/A	N/A	N/A	N/A
WRITE ENGINE NOT AVAILABLE	N/A	N/A	N/A	N/A
READ FAILED-NO STORAGE	N/A	N/A	N/A	N/A
WRITE FAILED-NO STORAGE	0.00	0.00	0.00	0.00
WRITE TO SEC-GBP FAILED	0.00	0.00	0.00	0.00
DELETE NAME LIST SEC-GBP	1113.00	1.83	0.02	0.02
DELETE NAME FROM SEC-GBP	0.00	0.00	0.00	0.00
UNLOCK CASTOUT STATS SEC-GBP	0.00	0.00	0.00	0.00
ASYNCH SEC-GBP REQUESTS	10.00	0.02	0.00	0.00

GROUP BP3 CONTINUED	QUANTITY	/SECOND	/THREAD	/COMMIT
WRITE AND REGISTER	87991.00	144.68	1.69	1.61
WRITE AND REGISTER MULT	75656.00	124.40	1.45	1.38
CHANGED PGS SYNC.WRTN	561.3K	922.92	10.79	10.26
CHANGED PGS ASYNC.WRTN	58554.00	96.28	1.13	1.07
PAGES WRITE & REG MULT	531.9K	874.54	10.22	9.72
READ FOR CASTOUT	423.00	0.70	0.01	0.01
READ FOR CASTOUT MULT	10106.00	16.62	0.19	0.18
WRITE TO SEC-GBP	N/A	N/A	N/A	N/A
CLEAN PAGES ASYNC.WRTN	N/A	N/A	N/A	N/A
CLEAN PGS READ AFT.RPL	N/A	N/A	N/A	N/A
PARTICIPAT.GBP REBUILD				
PAGE P-LOCK LOCK REQ	581.8K	956.55	11.18	10.63
SPACE MAP PAGES	0.00	0.00	0.00	0.00
DATA PAGES	0.00	0.00	0.00	0.00
INDEX LEAF PAGES	581.8K	956.55	11.18	10.63
PAGE P-LOCK UNLOCK REQ	604.0K	993.19	11.61	11.04
PAGE P-LOCK LOCK SUSP	559.9K	920.68	10.76	10.23
SPACE MAP PAGES	0.00	0.00	0.00	0.00
DATA PAGES	0.00	0.00	0.00	0.00
INDEX LEAF PAGES	559.9K	920.68	10.76	10.23
PAGE P-LOCK LOCK NEG	762.00	1.25	0.01	0.01
SPACE MAP PAGES	0.00	0.00	0.00	0.00
DATA PAGES	0.00	0.00	0.00	0.00
INDEX LEAF PAGES	762.00	1.25	0.01	0.01

Figure 4-24 DB2PE Statistics GBP report

The highlighted fields in Figure 4-24 are the counts of requests to the GBP structure from DB2 on system #@$3. Specifically, the fields are:

- SYN.READ(XI)-DATA RETURNED

  Notice that SYN here refers to the fact that DB2 indicated a preference that the request be sent to the CF synchronously. However, it is possible that the XES heuristic algorithm may convert some or all of these to be asynchronous requests and they will still be reported here as SYN. The same consideration applies to the next three fields as well.

- SYN.READ(XI)-NO DATA RETURN
- SYN.READ(NF)-DATA RETURNED
- SYN.READ(NF)-NO DATA RETURN
- READ CASTOUT CLASS
- READ DIRECTORY INFO
- READ STORAGE STATISTICS
- REGISTER PAGE
- WRITE AND REGISTER
- WRITE AND REGISTER MULT
- READ FOR CASTOUT
- READ FOR CASTOUT MULT

The DB2PE report used in this example is for approximately the same interval as the RMF Structure Activity report shown in Figure 4-21 on page 83. In Figure 4-24, the highlighted fields sum to roughly the number contained in the `ASYNCH GBP REQUESTS` field in that report.

Looking back at the RMF report, you can see that over 99% of requests from system #@$3 to the GBP3 structure are asynchronous, and the asynchronous request rate matches nearly exactly with the `ASYNCH GBP REQUESTS` value in the DB2PE report.

In your case, after having identified the DB2PE fields that report on activity to the GBP structure, you will then be in a better position to understand how changes to the response time for either the Primary of Secondary GBP structures may affect you.

4.5 Summary

In this chapter we discuss not only the reporting tools and products that we used for this project, but also the meanings of the fields and how they relate to each other across different products. Understanding where we got the data from will make it much easier for you to follow the discussion in subsequent chapters, where we concentrate on the *numbers* we encountered, and not so much on exactly where those numbers come from.

5

Our configuration

This chapter describes the configuration we used for the measurements described in the remainder of this book. In order to understand the meaning of the charts, tables, and discussions, it is important to have a grasp of the configuration and the naming conventions we used.

5.1 Software configuration

The configuration we used for all the measurement runs reported in this book consisted of three z/OS systems in a sysplex. However, because the configuration was also being used for a concurrent GDPS residency, only two of the systems were involved in running the workloads (we call these our “production” systems); the third system was the GDPS Control System. In addition to the three systems in the sysplex, a fourth system, outside the sysplex, was used to run the TPNS scripts to inject the transactions for the measurements.

All systems in the sysplex were running z/OS 1.7 at service levels that were current at the time of writing. In addition, CICS TS 3.1, DB2 V8.1, and IMS V9.1 were also used.

All products were at current service levels, and DB2 V8 was in New Function Mode with Locking Protocol 2 enabled. The systems, and the software they were running, are listed in Table 5-1.

Table 5-1 System configuration for measurements

System name	z/OS release	CICS TS	DB2	IMS
#@$2	1.7	3.1.0	8.1 + NFM	9.1
#@$3	1.7	3.1.0	8.1 + NFM	9.1
SC77[a]	1.7	None	None	None
SC47[b]	1.6	N/A	N/A	N/A

a. The GDPS Control System
b. The system where TPNS was run

The unusual names of the two production systems (#@$2 and #@$3) relates to the fact that these systems are part of an IBM offering sold to customers, so we needed to be as sure as possible that the system names would not match those used by any customers.

In addition, the following products were used to extract performance and service information for the reporting and analysis of the measurements:

- CICS Performance Analyzer V1.4
- IBM DB2 Performance Expert for z/OS V2.1
- IMS Performance Analysis V3.3
- RMF component of z/OS 1.7

5.1.1 CICS configuration

The CICS configuration consisted of two Terminal Owning Regions (TORs) on each of systems #@$2 and #@$3, and two Application Owning Regions (AORs) on each of systems #@$2 and #@$3. The TORs were in a VTAM® Generic Resource group called DSTCTOR. However, to improve the repeatability of the measurements, the ISTEXCGR exit was used to distribute requests to the TORs in a round robin order, rather than based on goal achievement.

The naming convention for the CICS regions was as follows:

DSTCabcd where:

DSTC This indicates a CICS region used for the Distance measurements.

a This is “P” for a TOR or AOR, or “C” for a CICSPlex® System Manager CMAS region.

b This is “A” for an AOR, “T”, for a TOR, “M” for a CMAS region, or “A” for a CAS address space.

c This is “2” for an AOR or TOR running on system #@$2, or “3” for an AOR or TOR running on system #@$3. For a CMAS region, c and d are $2 for the CMAS running on system #@$2, or $3 for the CMAS running on system #@$3.

d This is the instance.

So, for example, the region called DSTCPA22 is the second AOR running on system #@$2.

Each TOR was able to distribute transactions across all four CICS AORs. CICSPlex System Manager (CP/SM) was used to control the dynamic transaction routing. The CP/SM “queue” algorithm was used to determine where each transaction should be routed. CP/SM was able to route a transaction from any TOR to any AOR, on either system.

5.1.2 DB2 configuration

There was one DB2 subsystem on each of systems #@$2 and #@$3. The DB2 on #@$2 was called D8Q1, and the DB2 on #@$3 was called D8Q2. Both DB2s were members of a data sharing group called D8QG.

The DB2s were both running DB2 Version 8, with Locking Protocol 2 activated. Locking Protocol 2 may reduce DB2 locking in some environments; in those environments, this reduced locking should be especially beneficial for performance over longer distances.

5.2 Hardware configuration

The logical configuration used for this project is shown in Figure 5-1 on page 92. You can see that in Site1, we had one Coupling Facility (CF) called FACIL05 and one z/OS image (system #@$2). In Site2, we had the other CF, called FACIL06, and two z/OS images (systems #@$3 and SC77). Both Coupling Facilities were running CF Level 14 and were connected to both the “local” and the “remote” CF using ISC-3 links.

The “production” z/OS systems each had two dedicated CPs. The CFs each had one dedicated ICF.

Site1 also contained the primary disk (an IBM 2107). The secondary volumes (also an IBM 2107) were in Site2.

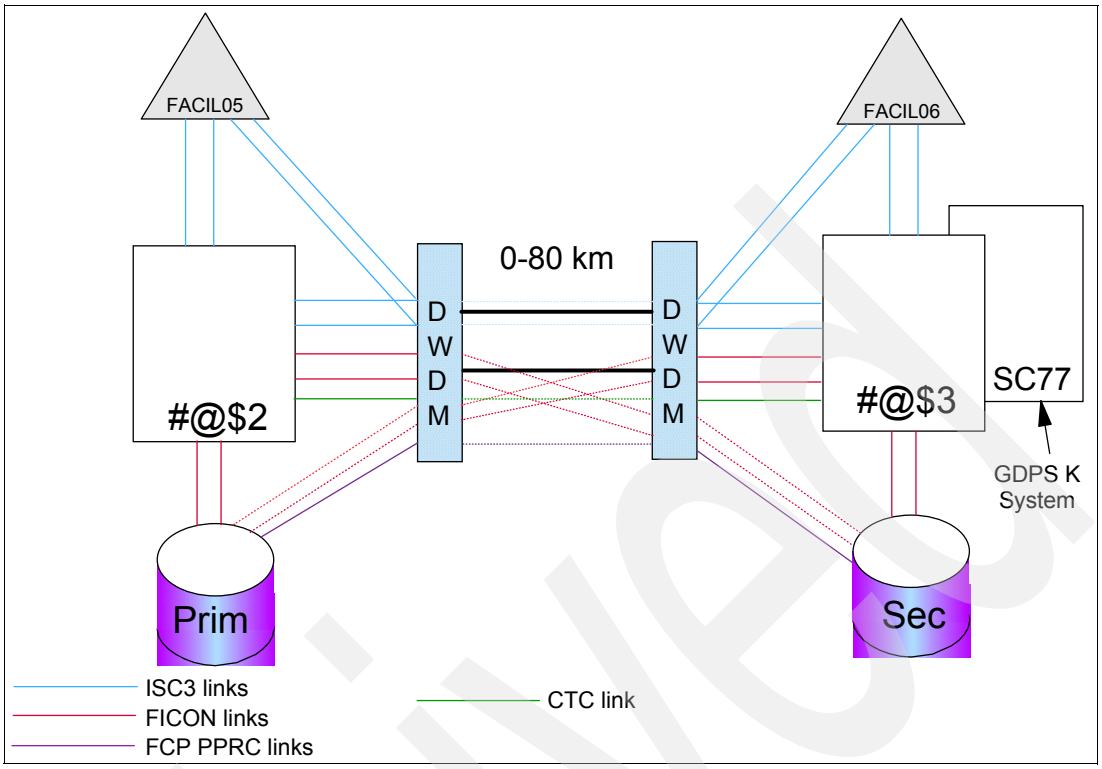


Figure 5-1 Logical configuration used for measurements

The two sites were connected by a pair of IBM 2029 DWDMs. Fibre suitcases were used to allow us to change the distance between the sites in 20 km increments. Measurements were taken at 0 km (bypassing the DWDMs), 20 km, 40 km, 60 km, and 80 km. Testing at 100 km would have required more cable that we had access to, and it would also have required amplifiers, which we also did not have access to. However, we believe that most customers considering cross-site data sharing will be doing so over distances of 80 km or less.

The physical configuration actually consisted of a single z9 S18. The z/OS LPARs were connected to the “local” CFs with two ISC-3 links, and to the “remote” CF also using two ISC-3 links. In most customer situations, more than two cross-site links would be recommended; however, the DWDMs we were using only had support for two ISC-3 2 Gb links. As will be seen in some of the results, the shortage of cross-site ISC-3 links resulted in higher-than-desirable subchannel utilization, especially for the longer distance measurements.

We also had two 1 Gb FICON links connecting each z/OS system to the disk in the other site. In addition to the XCF structures in the CFs, we also had one 1 Gb FICON CTC connecting the two systems that were taking part in data sharing.

Finally, we had one 1 Gb FCP PPRC link connecting the primary and secondary disk.

Important: We stress that the configuration we used is *not* one we would recommend for any production environment. More links are required to provide acceptable performance over distance.

Perhaps more importantly, you must be *especially* sensitive to single points of failure for all cross-site connectivity. Because the links will be running outside the protected machine room environment, it is possible that some event will result in the loss of most or all links running over one of the paths. Therefore, you must configure your connectivity so that acceptable performance and availability can continue to be delivered following the loss of one path.

5.2.1 Disk configuration

The primary and secondary DASD subsystems used for the measurements were IBM 2107-922 storage subsystems. All interfaces were 2 Gb-capable, however the interfaces connected to the DWDMs were only running at 1 Gb. PPRC was active for all measurements, even the one that had zero distance between the “sites”.

Dynamic PAV was enabled for all devices. We consider this to be a vital feature, especially at the longer distances. You will see that the IOSQ time in all measurements was zero (0). However, in one test that we performed at a long distance without PAV (the test is not documented in this book), the IOSQ time was as high as 8 milliseconds. We highly recommend the use of PAV for any configuration involving distances.

6

CICS/DB2 measurements

This chapter discusses the results of the measurements we ran using a CICS/DB2 workload. The only thing that was changed between the first five measurements was the distance between the “sites”. After the fifth run, at 80 km, we made one small change to try to reduce the impact of the distance. We describe the change we made and, more importantly, the reason why it had such a dramatic effect.

6.1 Presentation of measurement results

There are a number of important objectives for this chapter. Obviously, we want to provide you with the results of the measurements we took. We do this to provide you with an insight into the range of results that are possible with a number of different distances. However, it is vital that you understand that these results only reflect *our* mix of transactions, running with *our* hardware, using *our* specific set of data, and tuned in a particular manner.

A more important reason for providing the measurement results is that they allow us to illustrate not only what happened as the distance changed, but more importantly, *why* things happened as they did. It is this understanding of how distance impacts on the different components of application response and elapsed times that will put you in a stronger position to determine how various distances might impact *your* workload.

And, lastly, providing the results of our measurements helps illustrate why it is *not* possible to accurately predict the impact of large distances on application performance. For the components of response time that are directly impacted by distance, we discuss the calculations we used to estimate the impact of distance, and how accurate those estimates proved to be (the details of these calculations are provided in Appendix B, “Predicting the impact of distance on DB2 data sharing” on page 193). And for the components that are impacted in an indirect way, we demonstrate that it is not possible to estimate what these impacts will be.

We start by describing the results of our measurements when there was no distance between the components of the sysplex. Then we briefly present the results of the measurements at 20 km, 40 km, 60 km, and 80 km. Following the 80 km results, we describe the actions we took to identify and reduce the impact of contention.

Important: A typical customer read-to-write ratio would be four reads for each write (measured in terms of DB2 getpages and buffer updates), and this is approximately the ratio in the workload used for these measurements.

However, the CF intensity for data sharing-intensive customers would typically be in the range of 7-to-9 CF requests per MIPS. The CF intensity of our workload was 21 CF requests per MIPS, meaning that each of our transactions spent a lot more time talking to the CF, and a lot less time using CPU than an average customer workload. For this reason, increasing the distance between DB2 and the CF structures it was using had a more severe impact on our transactions that would be expected for an “average” transaction.

6.2 Base measurement - 0 km

The set of measurements for the CICS/DB2 workload commenced with a measurement taken with zero distance between the two “sites”. The Group Buffer Pools were duplexed and the DASD were mirrored using PPRC. However, there was basically no distance between the components of the configuration.

To refresh your memory of our configuration and the names of the systems and subsystems, refer to Chapter 5, “Our configuration” on page 89. The only changes to the configuration as described in that section are:

- Although we had two TORs in each z/OS system, only one TOR in each system was used for this set of measurements. There were also two AORs in each system, and *both* of those were used. CICSPlex/System Manager’s dynamic transaction routing function was used to route transactions from the two TORs to any of the four AORs.

- We had two TPNS virtual terminal networks, each directed to one CICS TOR in each system. The TPNS drivers were run on a system outside this sysplex to avoid any interference between TPNS and the workloads being measured.
- The DWDMs (IBM 2029s) were bypassed for the 0 km measurement, so the measurement represents a configuration where all boxes were in the same machine room, and data sharing is used to distribute the work across the two DB2 subsystems. This is typical of a configuration which uses local PPRC and HyperSwap to protect against component failure events that could affect DASD subsystems.

6.2.1 Transaction response times

As stated in 4.1, “Workload description” on page 54, our CICS/DB2 workload had a mix of transaction types. In this section (and the corresponding sections for the larger distances) we provide information about the overall average response time and the response time of the TPOS transaction, and then go into detail about the response time and response time breakdown for the most DB2-intensive transaction, TPNO.

Table 6-1, Table 6-2, and Table 6-3 show the transaction rate, average response time, and goal achievement, for *all* CICS transactions, for just the TPOS transaction, and for just the TPNO transaction, respectively, across both systems in the data sharing group. The numbers in these reports were extracted from the RMF Workload Activity reports.

Note that the response times are as seen by the AORs (EXECUTION times, in RMF terms); there may be small differences between these and the response times we discuss later, which will be as seen by DB2. The response times observed by the TORs (which are called ACTUAL times, in the RMF reports) were typically within a few milliseconds of the AOR response times. There may also be small differences as a result of the RMF and DB2PE intervals not lining up exactly.

Table 6-1 Information for all CICS transactions on both systems

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	278				
Resp time (milliseconds)	17				
WLM Goal achievement %	100				

Table 6-2 Information for TPOS transactions on both systems

TPOS	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	9.8				
Resp time (milliseconds)	4.0				
WLM Goal achievement %	100				

Table 6-3 Information for TPNO transactions on both systems

TPNO	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	97				
Resp time (milliseconds)	41				

TPNO	0 km (base)	20 km	40 km	60 km	80 km
WLM Goal achievement %	100				

Table 6-4 and Table 6-5 show the average response times for all transactions for each of the #@$2 and #@$3 systems, respectively. The numbers are nearly identical across the two systems, as you would expect. However, these results show the start of an effect we saw in all measurements, where system #@$3 was always a little slower than system #@$2. We were never able to identify the reason for this difference; however, it is sufficiently small not to have a significant effect on the results.

Note that for all measurements, all CICS transactions had a WLM goal of 90% of transactions to complete in .5 seconds. Although we were tempted to lower the goal, because the observed performance exceeded it by so much, the rationale of WLM is that you should specify goals that reflect your SLAs, *not* what the system is currently capable of. On that basis, we left the target at .5 seconds.

Table 6-4 Information for all CICS transactions on system #@$2

All transactions on #@$2	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139				
Resp time (milliseconds)	16				
WLM Goal achievement %	100				

Table 6-5 Information for all CICS transactions on system #@$3

All transactions on #@$3	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139				
Resp time (milliseconds)	17				
WLM Goal achievement %	100				

Notice that the number of transactions completed, and the response times, are very even across both systems. This is what you would expect, given that both systems were the same distance from the DASD and both Coupling Facilities.

Similarly, the number of TPOS transactions executed on each system and the response times are very even, as shown in Table 6-6 and Table 6-7 on page 99.

Table 6-6 Information for TPOS transaction on system #@$2

TPOS transactions on #@$2	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9				
Resp time (milliseconds)	4.0				
WLM Goal achievement %	100				

Table 6-7 Information for TPOS transaction on system #@$3

TPOS transactions on #@$3	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9				
Resp time (milliseconds)	4.0				
WLM Goal achievement %	100				

Finally, the number of TPNO transactions executed on each system and the response times are very close, as shown in Table 6-8 and Table 6-9.

Table 6-8 Information for TPNO transaction on system #@$2

All transactions on #@$2	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49				
Resp time (milliseconds)	40				
WLM Goal achievement %	100				

Table 6-9 Information for TPNO transaction on system #@$3

All transactions on #@$3	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49				
Resp time (milliseconds)	42				
WLM Goal achievement %	100				

To make it easier to see the changes in response time as the distance is changed, we have included the TPNO response time by system in a graphical format in Figure 6-1 on page 100. Although it is not very exciting to look at with just one pair of measurements, as the distance increases, it should help to show the relative impact on each system.

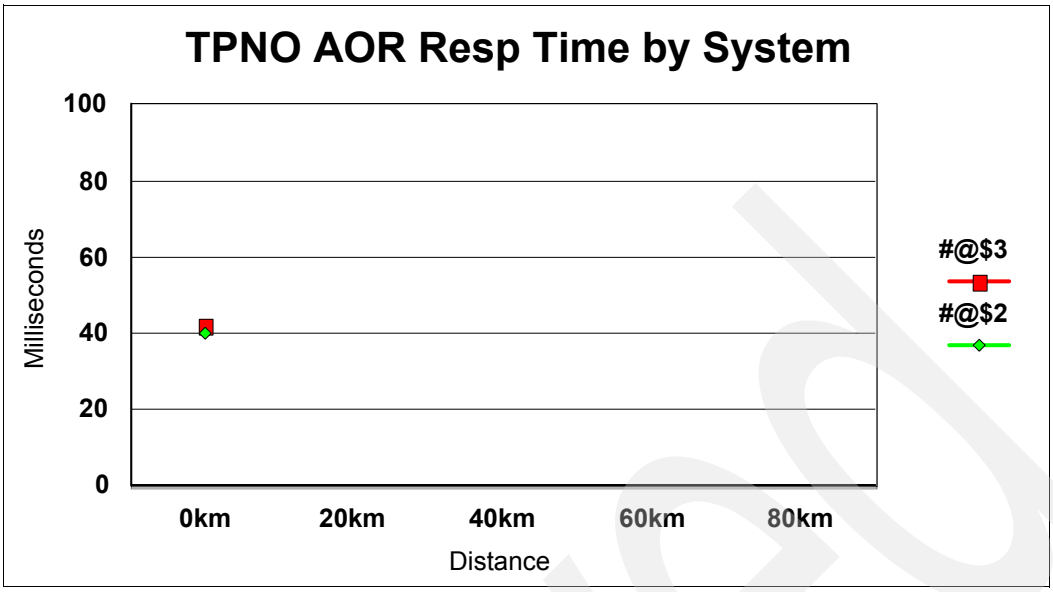


Figure 6-1 TPNO Response time by system for each distance

To verify the numbers we got from RMF, we also ran a pair of CICS PA reports for the TPNO transaction for each system. Table 6-10 and Table 6-11 report the response time from the perspective of both the TOR and the AORs for each of the two systems. As you can see, the response times as seen by CICS very closely match the transaction response time as seen by WLM.

Table 6-10 CICSPA response times (in milliseconds) for TPNO transaction for system #@$2

#@$2 system	0 km	20 km	40 km	60 km	80 km
#@$2 TOR	41.7				
#@$2 AOR	41.2				

Table 6-11 CICS PA response times (in milliseconds) for TPNO transaction for system #@$3

#@$3 system	0 km	20 km	40 km	60 km	80 km
#@$3 TOR	44.1				
#@$3 AOR	43.4				

6.2.2 TPNO response time breakout

Next, we look in more detail at the components of the TPNO response time on each system from the perspective of DB2. To do this, we use the DB2PE Accounting Reports. In order to help you understand where we are getting the data, we have included annotated reports for the two DB2 systems in Figure 6-2 on page 101 and Figure 6-3 on page 102. As we discuss the various components of the response time, we will refer back to these two figures.

```
   LOCATION: DB8Q                          DB2 PERFORMANCE EXPERT (V2)                          PAGE: 1-1
      GROUP: DB8QU                          ACCOUNTING REPORT - LONG                REQUESTED FROM: 08/11/06 17:54:00.00
     MEMBER: D8Q1                                                                             TO: 08/11/06 18:04:00.00
  SUBSYSTEM: D8Q1                         ORDER: PRIMAUTH-PLANNAME                  INTERVAL FROM: 08/11/06 17:54:00.04
DB2 VERSION: V8                               SCOPE: MEMBER                                    TO: 08/11/06 18:03:59.98

PRIMAUTH: CICSUSER  PLANNAME: TPCCP2NO

ELAPSED TIME DISTRIBUTION                                          CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------      ------------------------------------------------------------------------
APPL   |======> 12%                                               CPU    |=======> 14%
DB2    |======> 13%                                               NOTACC |> 1%
SUSP   |====================================> 75%                 SUSP   |========================================> 85%

AVERAGE       APPL(CL.1)   DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS    AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------   ----------  ----------    --------------------   ------------  --------    --------------------------
ELAPSED TIME (9) 0.038356    0.033661        N/P     LOCK/LATCH(DB2+IRLM) (1)   0.009399      1.17    #OCCURRENCES    :    29228
 NONNESTED     0.038356    0.033661        N/A     SYNCHRON. I/O (2)          0.002180      2.40    #ALLIEDS        :    29228
 STORED PROC   0.000000    0.000000        N/A      DATABASE I/O              0.000691      1.83    #ALLIEDS DISTRIB:        0
 UDF           0.000000    0.000000        N/A      LOG WRITE I/O             0.001489      0.57    #DBATS          :        0
 TRIGGER       0.000000    0.000000        N/A     OTHER READ I/O             0.000049      0.19    #DBATS DISTRIB. :        0
                                                   OTHER WRTE I/O             0.000000      0.00    #NO PROGRAM DATA:    29228
CPU TIME      0.005716 (8) 0.004627 (7)    N/P     SER.TASK SWTCH             0.007087      1.00    #NORMAL TERMINAT:    29228
 AGENT        0.005716    0.004627        N/A      UPDATE COMMIT (5)         0.007000      1.00    #ABNORMAL TERMIN:        0
  NONNESTED   0.005716    0.004627        N/P      OPEN/CLOSE                0.000000      0.00    #CP/X PARALLEL. :        0
  STORED PRC  0.000000    0.000000        N/A      SYSLGRNG REC              0.000000      0.00    #IO PARALLELISM :        0
  UDF         0.000000    0.000000        N/A      EXT/DEL/DEF               0.000087      0.00    #INCREMENT. BIND:        0
  TRIGGER     0.000000    0.000000        N/A      OTHER SERVICE             0.000000      0.00    #COMMITS        :    28934
 PAR.TASKS    0.000000    0.000000        N/A     ARC.LOG(QUIES)             0.000000      0.00    #ROLLBACKS      :      294
                                                   ARC.LOG READ               0.000000      0.00    #SVPT REQUESTS  :        0
SUSPEND TIME  0.000000    0.028674        N/A     DRAIN LOCK                 0.000000      0.00    #SVPT RELEASE   :        0
 AGENT            N/A    0.028674        N/A     CLAIM RELEASE              0.000000      0.00    #SVPT ROLLBACK  :        0
 PAR.TASKS        N/A    0.000000        N/A     PAGE LATCH                 0.000055      0.05    MAX SQL CASC LVL:        0
 STORED PROC 0.000000        N/A         N/A     NOTIFY MSGS                0.000000      0.00    UPDATE/COMMIT   :    22.97
 UDF         0.000000        N/A         N/A     GLOBAL CONTENTION (3)      0.006972      7.51    SYNCH I/O AVG.  : 0.000908
                                                   COMMIT PH1 WRITE I/O       0.000000      0.00
NOT ACCOUNT.      N/A    0.000361 (6)    N/A     ASYNCH CF REQUESTS (4)     0.002931     20.71
DB2 ENT/EXIT      N/A      139.84        N/A     TOTAL CLASS 3              0.028674     33.05
EN/EX-STPROC      N/A        0.00        N/A
EN/EX-UDF         N/A        0.00        N/A
DCAPT.DESCR.      N/A         N/A        N/P
LOG EXTRACT.      N/A         N/A        N/P
```

Figure 6-2 DB2PE Accounting Report for D8Q1

```
   LOCATION: DB8Q                          DB2 PERFORMANCE EXPERT (V2)                          PAGE: 1-1
      GROUP: DB8QU                          ACCOUNTING REPORT - LONG                REQUESTED FROM: 08/11/06 17:54:00.00
     MEMBER: D8Q2                                                                            TO: 08/11/06 18:04:00.00
  SUBSYSTEM: D8Q2                          ORDER: PRIMAUTH-PLANNAME                 INTERVAL FROM: 08/11/06 17:54:00.05
DB2 VERSION: V8                               SCOPE: MEMBER                               TO: 08/11/06 18:03:59.95

PRIMAUTH: CICSUSER  PLANNAME: TPCCP2NO

ELAPSED TIME DISTRIBUTION                                          CLASS 2 TIME DISTRIBUTION
-------------------------------------------------------------      ------------------------------------------------------------------------
APPL   |=====> 11%                                                CPU    |======> 12%
DB2    |======> 13%                                               NOTACC |=> 2%
SUSP   |====================================> 76%                 SUSP   |=========================================> 86%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    -------------------- ------------  --------    --------------------------
ELAPSED TIME (9) 0.040546    0.036054         N/P    LOCK/LATCH(DB2+IRLM) (1)    0.009777      1.29    #OCCURRENCES    :    29114
 NONNESTED     0.040546    0.036054         N/A    SYNCHRON. I/O (2)           0.002155      2.48    #ALLIEDS        :    29114
 STORED PROC   0.000000    0.000000         N/A     DATABASE I/O               0.000734      1.90    #ALLIEDS DISTRIB:        0
 UDF           0.000000    0.000000         N/A     LOG WRITE I/O              0.001421      0.58    #DBATS          :        0
 TRIGGER       0.000000    0.000000         N/A    OTHER READ I/O              0.000044      0.19    #DBATS DISTRIB. :        0
                                                   OTHER WRTE I/O              0.000000      0.00    #NO PROGRAM DATA:    29114
CPU TIME      0.005652 (8) 0.004503 (7)     N/P    SER.TASK SWTCH              0.006778      1.00    #NORMAL TERMINAT:    29114
 AGENT        0.005652    0.004503         N/A     UPDATE COMMIT (5)          0.006686      1.00    #ABNORMAL TERMIN:        0
  NONNESTED   0.005652    0.004503         N/P     OPEN/CLOSE                 0.000000      0.00    #CP/X PARALLEL. :        0
  STORED PRC  0.000000    0.000000         N/A     SYSLGRNG REC               0.000000      0.00    #IO PARALLELISM :        0
  UDF         0.000000    0.000000         N/A     EXT/DEL/DEF                0.000092      0.00    #INCREMENT. BIND:        0
  TRIGGER     0.000000    0.000000         N/A     OTHER SERVICE              0.000000      0.00    #COMMITS        :    28856
 PAR.TASKS    0.000000    0.000000         N/A    ARC.LOG(QUIES)              0.000000      0.00    #ROLLBACKS      :      258
                                                   ARC.LOG READ                0.000000      0.00    #SVPT REQUESTS  :        0
SUSPEND TIME  0.000000    0.030956         N/A    DRAIN LOCK                  0.000000      0.00    #SVPT RELEASE   :        0
 AGENT            N/A    0.030956         N/A    CLAIM RELEASE               0.000000      0.00    #SVPT ROLLBACK  :        0
 PAR.TASKS        N/A    0.000000         N/A    PAGE LATCH                  0.000059      0.06    MAX SQL CASC LVL:        0
 STORED PROC 0.000000        N/A         N/A    NOTIFY MSGS                 0.000000      0.00    UPDATE/COMMIT   :    22.99
 UDF         0.000000        N/A         N/A    GLOBAL CONTENTION (3)       0.009580     32.60    SYNCH I/O AVG.  : 0.000868
                                                   COMMIT PH1 WRITE I/O        0.000000      0.00
NOT ACCOUNT.      N/A    0.000596 (6)     N/A    ASYNCH CF REQUESTS (4)      0.002563     20.80
DB2 ENT/EXIT      N/A      139.96         N/A    TOTAL CLASS 3               0.030956     58.42
EN/EX-STPROC      N/A        0.00         N/A
EN/EX-UDF         N/A        0.00         N/A
DCAPT.DESCR.      N/A         N/A         N/P
LOG EXTRACT.      N/A         N/A         N/P
```

Figure 6-3 DB2PE Accounting Report for D8Q2

As the distance between the system and the CF increases, we expect that the elapsed time for each of these components will increase, although not all by the same degree. To help understand exactly why the response times are increasing with distance, it is important to understand what is happening to each of the major components that make up the transaction response time.

Tip: Remember that the DB2PE Accounting report provides information at the *thread* level. In our environment, we had exactly one transaction per thread, so we did not need to make any adjustments. However, if you were to undertake an exercise like this yourself, keep in mind that you may need to adjust the thread times as reported by DB2PE based on the average number of transactions per thread.

Because the TPNO transaction performs most of its work in DB2, we focus on this activity to understand what happened to the response times as the distance changed. Table 6-12 summarizes the role of DB2 processing in the TPNO response time on system #@$2.

Table 6-12 Average DB2 times for TPNO transaction on system #@$2 (in milliseconds)

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention 1	9.4				
Synchronous I/O delays 2	2.2				
Global contention 3	7.0				
Asynchronous CF Requests 4	2.9				

Suspend time component	0 km	20 km	40 km	60 km	80 km
Update/Commit time 5	7.1				
Other miscellaneous DB2 suspends	0.1				
Other Not Account by DB2 6	0.4				
CPU time spent in DB 7	4.6				
Non-DB2 CPU time 8	1.1				
Total as seen by DB2 9	38.4				

For a detailed explanation of these fields and their meaning, refer to 4.2.1, “DB2 and application CPU times” on page 57 and 4.2.2, “DB2 Class 3 Suspension” on page 59. All the numbers in Table 6-12 on page 102 are extracted (or extrapolated) directly from the DB2PE report shown in Figure 6-2 on page 101, and have *not* been adjusted to reflect the fact that reported Global Contention also includes some amount of time waiting for asynchronous lock requests (this is discussed in “Lock/Latch” on page 71).

Note: The row titled “Other miscellaneous DB2 suspends” contains the sum of other DB2 class 3 suspends, each of which were typically less than one millisecond. These generally have no relevance to the focus of this chapter, so we do not discuss them further.

Figure 6-4 shows the DB2 response components on system #@$2 in graphical format. The difference between the response time shown here (roughly 39 ms) and that reported by CICS (about 41 ms, as shown in Table 6-10 on page 100) is the time spent in CICS from transaction arrival, through any queuing in CICS, and all processing up to the first SQL call, and then the time from the commit through to when the response is sent by CICS back to the terminal.

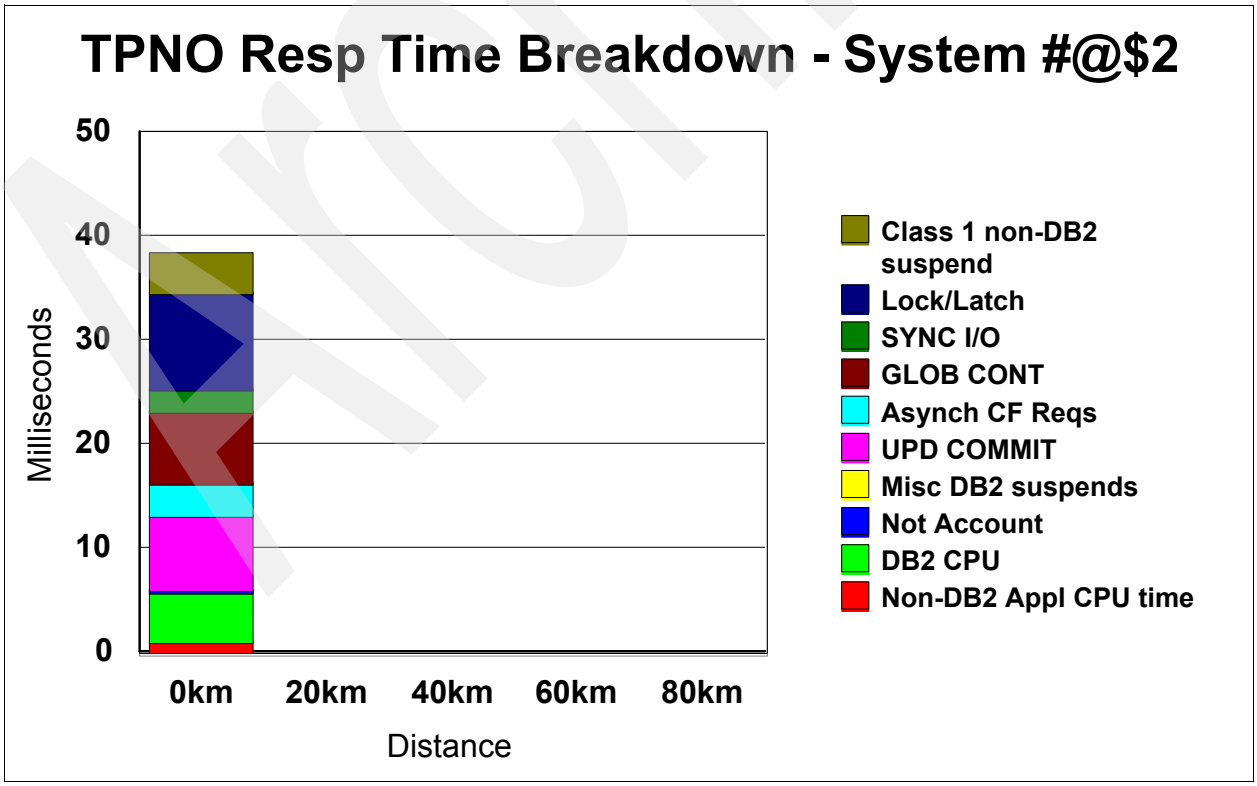


Figure 6-4 TPNO response time breakdown for system #@$2

The breakout of the DB2 Class 1 response time components for system #@$3 is shown in Table 6-13. As you can see, the numbers are very similar to those on system #@$2.

Table 6-13 Average DB2 times for TPNO transaction on system #@$3 (in milliseconds)

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention 1	9.8				
Synchronous I/O delays 2	2.2				
Global contention 3	9.6				
Asynchronous CF Requests 4	2.6				
Update Commit time 5	6.8				
Other miscellaneous DB2 suspends	0.1				
Other Not Account by DB2 6	0.6				
CPU time spent in DB 7	4.5				
Non-DB2 CPU time 8	1.1				
Total as seen by DB2 9	40.5				

Figure 6-5 shows the response time components for system #@$3 in graphical format.

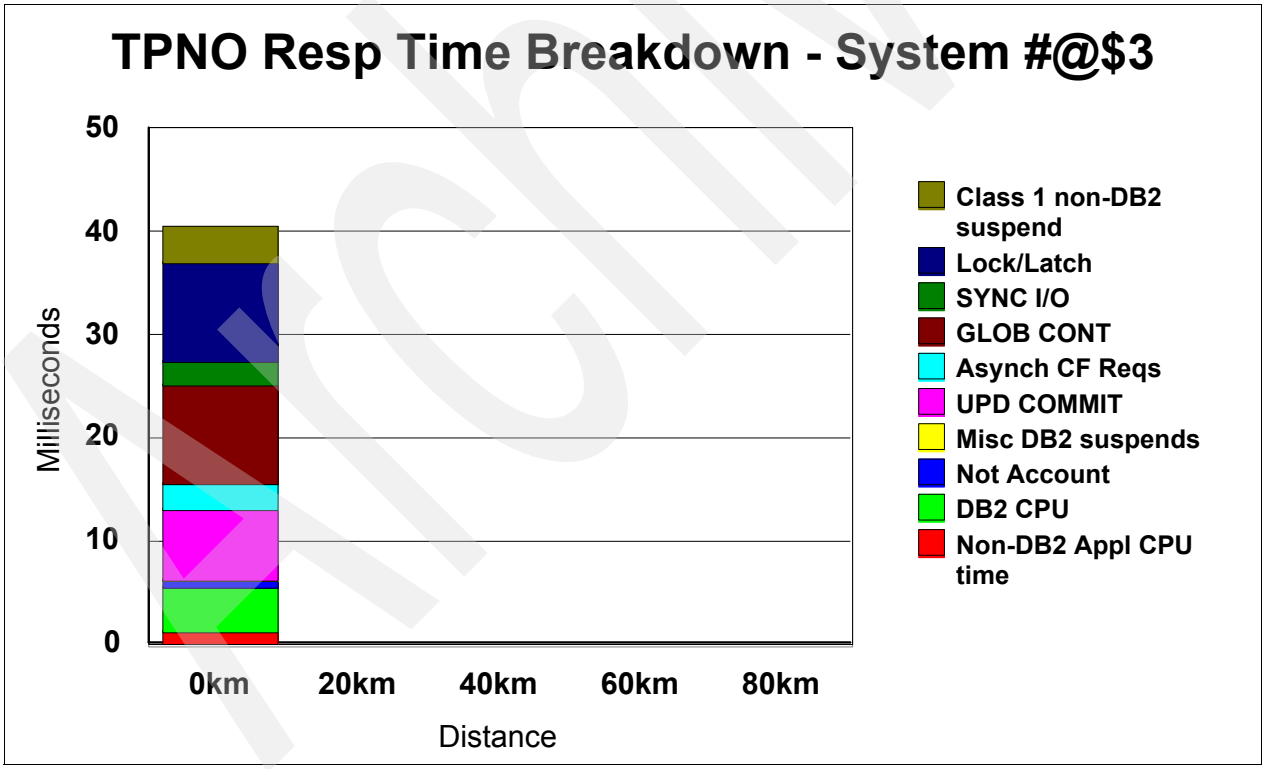


Figure 6-5 TPNO response time breakdown for system #@$3

As you can see in Figure 6-4 on page 103 and Figure 6-5, the response times delivered by the two DB2 subsystems are nearly identical, as is the breakdown of the response times. We expect that this will not remain the case as the distance between the systems is increased.

6.2.3 Lock/Latch

The first component of TPNO response time we will look at is the local contention (shown at 1 in Figure 6-2 on page 101 and Figure 6-3 on page 102). As described in “Lock/Latch” on page 71, this is the time spent by the transaction waiting for resources held by other threads in the same DB2. Although local contention is not directly affected by distance, delays due to the distance can cause transactions to run longer, thereby holding the serialization on resources they are using for a longer time, in turn causing more local contention.

Table 6-14 Average Lock/Latch suspension time and events for TPNO transaction

Lock/Latch suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	9.4				
No. of suspension events - #@$2	1.2				
Total suspension time - #@$3	9.8				
No. of suspension events - #@$3	1.3				

Table 6-14 shows the total amount of time spent on average by a TPNO transaction due to local Lock/Latch contention for each of the two systems, and the number of contention events that caused that delay. Using this information, we can easily calculate that each local contention event took about 7.8 ms (9.4 / 1.2 and 9.8 / 1.3) on both the systems to complete.

There is not much else we can say about Lock/Latch at this time; it is indicative of the elapsed time of other transactions in the same DB2 that are holding resources needed by this transaction. If the elapsed time of those holding transactions increases, we would expect to see Lock/Latch times similarly increase.

6.2.4 Synchronous I/O

The next part of the TPNO response time is time spent waiting for synchronous I/O to complete (2 in Figure 6-2 on page 101 and Figure 6-3 on page 102). Remember that these are I/Os for requested pages that are not found in either the local or global buffer pools, or for the log writes executed prior to the commit.

Table 6-15 Average Sync I/O suspend time and number of events for TPNO transaction

Synchronous I/O suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	2.2				
No. of suspension events - #@$2	2.4				
Total suspension time - #@$3	2.2				
No. of suspension events - #@$3	2.5				

Table 6-15 shows how long, on average, a TPNO transaction has been delayed by synchronous I/O events, and how many synchronous I/O events affected each transaction. On average, each synchronous I/O event (this could be a pair of I/Os if the event is for a log write) took a bit less than one millisecond to complete (2.2 ms / number of events (2.4)).

Although the average disk response time during this test was in the 0.6 milliseconds range[1], the suspension time shown here was mainly due to log writes which are longer than writes

[1] The disk response time comes from the RMF Workload Activity report for the DB2MSTR address space, and therefore reflects the actual disk response times observed for I/Os issued by that address space.

due to the cost of mirroring (log writes accounted for 2/3 of synchronous I/O events, as can be seen in the DB2PE Accounting Report in Figure 6-2 on page 101). Also, the reported time includes other internal work in addition to the DASD writes.

Because the Synchronous I/O time includes elements that are impacted by distance (I/O response time) and elements that are not (internal DB2 processing associated with these I/Os), we would not expect this component to increase completely linearly as the distance increases.

6.2.5 Global contention

The DB2 global contention value includes time spent by transactions waiting for a logical lock related to a GBP-dependent resource, or for a physical lock currently held by another DB2 subsystem (3 in Figure 6-2 on page 101 and Figure 6-3 on page 102).

As described in "Global Contention" on page 73, the introduction of the XES heuristic algorithm means that the reported global contention value may also include other wait times. One of them is the time spent by DB2 waiting for an asynchronous IXLLOCK request to complete. The response time for the IXLLOCK requests will increase as the distance increases, meaning that XES will convert them to be asynchronous requests, resulting in their execution time being reported in this category.

Another event that may be included in this category is the time spent by DB2 waiting for local contention on a GBP-dependent resource. If the heuristic algorithm converts the associated lock request for this resource to be an asynchronous request, this type of event can also end up being reported as global contention. For this reason, we expect to see the number of global contention events and the associated elapsed times increase as the distance increases.

This also means that to get a more realistic view of the actual delay caused by global contention, we need to do some more work. As discussed in "Global Contention" on page 73, we need to determine how many of the *reported* global contention events are *really* global contention situations. We obtain this information in the DB2PE Accounting Report[2].

Table 6-16 Average global contention time and events for TPNO transaction

Global contention	0 km	20 km	40 km	60 km	80 km
Reported global contention time - #@$2	7.0				
Reported number of global contention events - #@$2	7.5				
Actual no. of global contention events - #@$2	0.35				
Reported global contention time - #@$3	9.6				
Reported number of global contention events - #@$3	32.6				
Actual no. of global contention events - #@$3	0.35				

[2] The actual number of global contention events used in the formula is provided by DB2PE in the Data Sharing Locking Section (SMF101 - DSNDQTGA) of the Accounting report.

As shown in Table 6-16 on page 106, the reported global contention delay for the TPNO transaction is 7.0 ms on system #@$1 and 9.6 ms on system #@$3.

As discussed previously, this value includes both the delay due to actual global contention, plus delay due to asynchronous execution of some IXLLOCK requests. So, to determine the amount of delay that was caused by actual global contention, we need to subtract the time spent waiting for the asynchronous lock requests to complete.

The number of asynchronous lock request events is obtained by subtracting the number of actual global contention events from the reported number of global contention events. So, for system #@$2 in Table 6-16 on page 106, we subtract .35 from 7.5, resulting in 7.15 asynchronous lock request events. We then multiply this by the average response time for an asynchronous lock request from system #@$2. We obtain this information from the RMF Structure Activity report, as shown in Table 6-17.

Table 6-17 RMF structure activity report for DB8QU_LOCK1 structure

System	Synch Rate	Synch Serv Time (mics)	Asynch Rate	Asynch Serv Time (mics)
#@$2	5430	21.2	693	75.7
#@$3	3358	21.9	2393[a]	64.5

a. The percentage of lock requests going asynchronously on system #@$3 was significantly higher because the average synchronous response time for that system was right on the threshold for the heuristic algorithm.

In this case, in the Global Contention field in DB2PE for DB2 on system #@$2, we had 7.5 reported global contention events per transaction, with only 0.35 of them being actual global contentions. If we say that the average service time reported by RMF for an asynchronous request to this lock structure is 76 microseconds (from Table 6-17), we can calculate that 7.15 asynchronous requests accounted for 543 microseconds (76 * 7.15). The remaining 6.5 milliseconds is the time spent waiting on the actual 0.35 global contention events. Although the global contention time on #@3 was a little higher (9.0 ms), the number of events was over four times higher, resulting in 7.5 ms of suspension not related to asynchronous lock requests.

As the distance within the sysplex increases, we expect the actual global contention time and the number of real global contention events to increase, because the transactions holding the resources that are in contention will experience longer elapsed times. The global contention time as currently reported by DB2PE will also increase as the percentage and duration of asynch CF lock requests increases with the distance.

Tip: We spent a significant amount of time trying to understand which events get reported by DB2 as local contention (also referred to as Lock/Latch) and which are reported as global contention. However, what is really important is that the transaction was delayed because of *some* contention. The logic that DB2 uses to determine how to report a given contention event is extremely complex, and you need to understand the logic of each transaction, as well as the interrelationships between the transactions, in order to fully understand why a given contention event is reported by DB2 in a specific way.

It is far more important is to focus on the *cause* of the contention, and what can be done to reduce it. You will get a much better return on your time investment if you concentrate on reducing contention, without getting overly concerned about how the event is being reported. As you will see later in this chapter, one change that we implemented to reduce contention improved both the global contention and the lock/latch times.

6.2.6 Asynch CF Requests

Asynch CF Requests refers to the time spent waiting for asynchronous GBP requests issued by the thread prior to commit to complete (4 in Figure 6-2 on page 101 and Figure 6-3 on page 102). It includes the time spent by DB2 waiting for GBP operations that are issued by DB2 as synchronous requests (reads from the GBP, for example), but that are sent to the CF asynchronously by XES, and can include both reads and writes to the GBPs.

The GBP requests that are reported in the Asynch CF Requests field are just one of the three types of GBP requests that cause the thread to be suspended. The other two are synchronous CF requests (the time for these is included in the DB2 CPU time), and requests issued during commit processing (unless the transaction is using two-phase commit, in which case the requests issued during commit are included in Asynch CF Requests).

The execution of the thread is suspended until the requests reported in Asynch CF Requests are complete. So, as the distance between DB2 and the GBP structure increases and the corresponding response time for those requests increases, you will expect the transaction response time to be similarly impacted.

Note that writes to the GBP that are triggered by various DB2 thresholds are done under a different execution unit and are *not* included in this field and do *not* delay the execution of the thread.

Table 6-18 Average Asynch CF Request suspension time and events for TPNO transaction

Asynch CF Requests	0 km	20 km	40 km	60 km	80 km
Total Asynch CF Request suspension time - #@$2	2.9				
Number of events - #@$2	20.7				
Total Asynch CF Request suspension time - #@$3	2.6				
Number of events - #@$3	20.8				

Table 6-18 shows the impact of asynchronous GBP requests on TPNO at 0 km. As the response time for each system to communicate to the remote CF increases with distance, we expect this time to also increase.

Note that GBP requests that are issued by the thread and sent to the CF synchronously are *not* included in this field. In fact, the number of GBP requests that are issued synchronously is not reported directly anywhere in the Accounting Report. However, as the distance between DB2 and the CF increases, it is likely that nearly all GBP requests will be sent asynchronously.

The best way to get an indication of how many additional requests will show up in this field is to look at the RMF CF report to see what percent of GBP requests are being sent synchronously today. As the distance between DB2 and the GBP structure increases, most of those synchronous requests will be converted to asynchronous ones.

6.2.7 Update Commit

Update Commit time refers to the time spent by a transaction waiting for the Commit process to complete (5 in Figure 6-2 on page 101 and Figure 6-3 on page 102). Commit processing consists of writes to the primary and secondary DB2 log data sets, writing all pages that have been changed within the scope of this unit of recovery to the primary and secondary GBPs,

releasing any locks held by the thread, and other cleanup processing. Unfortunately, the Accounting Report does not provide information about how *many* log writes, GBP write requests, or unlock requests are included in this time.

As shown in Table 6-19, the Update Commit time for TPNO was 7.1 ms for system #@$2 and 6.8 ms for system #@$3. The “No. of Suspension Events” reports on the number of commits per thread, not on the number of log I/Os, GBP requests, and so on.

Table 6-19 Average Update Commit suspend time for TPNO transaction (in milliseconds)

Update Commit suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	7.1				
No. of suspension events - #@$2	1				
Total suspension time - #@$3	6.8				
No. of suspension events - #@$3	1				

Log writes

To understand how log write I/Os impacted Update Commit, we ran an RMF Workload Activity report for the DB2 MSTR address spaces. Most of the I/Os done by the MSTR address space are for logging, so by looking at the I/O rate (reported as SSCHRT) and the response time (reported as RESP) in the RMF report, we can understand the impact of the log write I/Os. Figure 6-6 contains the report for the D8Q1 DB2 MSTR address space.

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
2549	7	2454	29267	342.7	0.6	0.2	0.3	0.1	0.0

Figure 6-6 Workload activity report for D8Q1 MSTR address space

The corresponding information for the D8Q2 DB2 MSTR address space is shown in Figure 6-7.

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
2595	8	2499	28902	340.0	0.6	0.2	0.3	0.1	0.0

Figure 6-7 Workload activity report for D8Q2 MSTR address space

You can see that the average response time for each disk I/O for the D8Q1 MSTR address space (in system #@$2) was 0.6 ms, and the equivalent I/O for the D8Q2 MSTR address space (in system #@$3) was also 0.6 ms. If we assume that one log record is written at commit time[3], and allowing for writes to both the primary and alternate log data sets, that accounts for roughly 1.2 of the 7 ms of Update Commit time.

GBP writes

The second component of Update Commit time is the time required to write to the primary and secondary GBPs. Because the Accounting Report does not provide a specific count of how many GBP requests were issued during commit processing, we are presented with two challenges:

- Determining how many requests are included in commit processing

[3] In commit, the log record is written to the DB2 log buffer. The buffer is them immediately written to disk. Depending on how full the buffer was before this log record arrived, this might result in a small I/O (if the buffer was nearly empty), a large I/O, or even more than one I/O (if the buffer was nearly full).

- Determining how long they took

Rather than spend an inordinate amount of time attempting to identify the number of requests that are issued during Update Commit, we determined that it was easier *and* more accurate to use the alternate method described in “GBP writes” on page 79 to assess the impact of increased GBP response times. We refer to this again in Appendix B, “Predicting the impact of distance on DB2 data sharing” on page 193, where we discuss estimating the impact of increased distances.

Releasing locks

The next part of Update Commit processing is releasing the locks held by the thread. This part of Update Commit would typically take very little time. The IXLLOCK request is issued asynchronously by DB2, and the thread does not need to wait for the locks to be released before it can move on to the next phase of commit processing. As a result, we believe that the time spent in this phase of commit processing is brief and unlikely to increase with distance.

Cleanup

The final part of Update Commit is the DB2 cleanup processing. Compared to the other three parts of Update Commit, this should complete very quickly. Also, because it does not involve going outside the processor, it should be unaffected by any changes in the distance within the sysplex.

6.2.8 Not Account

Not Account time is elapsed time spent in DB2, but not covered by any of the other DB2 execution or suspension states (6 in Figure 6-2 on page 101 and Figure 6-3 on page 102). APAR PK27934 significantly decreases the amount of time charged to Not Account by more accurately accounting for it elsewhere; this is discussed in more detail in “Not Account” on page 61. However, there will always be a relatively small amount of processing that cannot be easily accounted for. In this measurement, Not Account was around 1% of the thread elapsed time.

Table 6-20 Average Not Accounted (in milliseconds) for TPNO transaction

Not Accounted	0 km	20 km	40 km	60 km	80 km
Not Accounted - #@$2	0.4				
Not Accounted - #@$3	0.6				

As you can see in Table 6-20, the average Not Accounted time for both systems is not significant. For more details, see “Not Account” on page 84.

6.2.9 DB2 CPU

The DB2 CPU time reported by DB2PE as Class 2 CPU time (7 in Figure 6-2 on page 101 and Figure 6-3 on page 102) reflects the amount of time that DB2 is actually using the CPU, executing instructions on behalf of the transaction. This CPU time is charged back to the allied address space. Table 6-21 shows the average DB2 CPU time for the TPNO transaction on systems #@$2 and #@$3 at 0 km.

Table 6-21 Average DB2 CPU time (in milliseconds) for TPNO transaction

TPNO DB2 CPU time	0 km	20 km	40 km	60 km	80 km
DB2 CPU time - #@$2	4.6				

TPNO DB2 CPU time	0 km	20 km	40 km	60 km	80 km
DB2 CPU time - #@$3	4.5				

6.2.10 All Coupling Facility activity

In addition to the performance of the specific DB2 structures, it is also important to monitor the overall health, performance, and capacity of the Coupling Facility. The metrics that we will specifically monitor in the performance runs are CF CPU utilization and subchannel utilization.

The CPU utilization of each CF can be found in the RMF Coupling Facility report and is shown in Table 6-22. You can see that for this measurement, both CFs were lightly loaded.

Table 6-22 Coupling Facility CPU utilization

	0 km	20 km	40 km	60 km	80 km
FACIL05	14.3%				
FACIL06	4.9%				

There are two main reasons for the unbalanced utilization of the two CFs:

- All the primary GBP structures are located in FACIL05, so all read and all castout requests to any of the GBP structures will be directed to that CF.
- The DB2 lock structure was also in FACIL05. This one structure accounted for nearly 73% of all requests to that CF.

The CF subchannel utilization numbers can be calculated from the values provided in the RMF CF SUBCHANNEL ACTIVITY report, which is provided after the detail structure activity section of the RMF CF Activity report. The subchannel utilization rates for each system for each CF are shown in Table 6-23 and Table 6-24, and are currently well below the IBM-recommended threshold of 30% busy.

Table 6-23 FACIL05 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	3.5%				
From #@$3	3.1%				

Table 6-24 FACIL06 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	1.2%				
From #@$3	1%				

It is important to remember that a subchannel is in use for the full response time, even for an asynchronous request. Therefore, because we expect both the percent of asynchronous requests and the response time for each request to increase as the distance increases, we expect to see increases in CF subchannel utilization.

6.2.11 XCF signaling

As discussed in 2.5.1, “XCF/XES role as global lock manager” on page 20, when DB2 issues a lock or unlock request for a lock entry that is in contention, the request gets passed to the global lock manager using XCF signals. Therefore, as the number of lock contention events increases (which we expect to happen as the transaction response time increases with distance), you would expect an increase in the number of related XCF messages per transaction. In this section we track the number of XCF messages and the distribution over the available CF structure and FICON CTC paths.

As seen in Table 6-25, the number of XCF messages sent from system #@$2 to #@$3 in the DEFAULT transport class was 168.5 per second, most of which went over the XCF signalling structures in the CF.

Table 6-25 XCF messages sent per second from #@$2 to #@$3

	0 km	20 km	40 km	60 km	80 km
CF structures	144.2				
FICON CTC	24.3				

As seen in Table 6-26, the number of XCF messages sent from system #@$3 to #@$2 in the DEFAULT transport class was similar, 170.6 per second, and also in this case, a large number used the XCF signalling structures in the CF.

Table 6-26 XCF messages sent per second from #@$3 to #@$2

	0 km	20 km	40 km	60 km	80 km
CF structures	126.5				
FICON CTC	44.1				

For both systems in this measurement, the average end-to-end delivery time for the messages that went over the XCF signalling structures was faster than for messages delivered over the CTC paths. Considering that XCF will always select the fastest available path, the balance of requests going over the CF compared to over the CTC reflects the better response time that is seen for the CF.

6.2.12 z/OS CPU utilization

Finally, we include overall z/OS CPU utilization and the z/OS capture ratio. The CPU utilization was deliberately kept low for these measurements, in order to ensure that CPU contention would not have an effect on the measured response times. The CPU utilization numbers for each system are included in Table 6-27.

Table 6-27 z/OS CPU utilization

	0 km	20 km	40 km	60 km	80 km
#@$2	45.2%				
#@$3	46.5%				

To determine whether increased distance has any effect on capture ratio, we also included the capture ratios for each distance in Table 6-28. As the distance increases in future measurements, we will monitor to see if the capture ratio changes.

Table 6-28 z/OS capture ratios

	0 km	20 km	40 km	60 km	80 km
#@$2	90%				
#@$3	91.2%				

6.2.13 Summary

No bottlenecks or constraints were observed. The response time of all transactions was well within their SLA target of .5 secs. Really, the objective of showing all these figures was to set the baseline measurement that will be used to identify the changes that are introduced as we start to increase the distance between the components in the sysplex.

6.3 Measurement at 20 km

For the next set of measurements, at 20 km, the configuration was unchanged from that used for the base measurement, with the exception that a pair of DWDMs with 20 km of fiber between them was added to the configuration.

As a general approach, we decided in advance to avoid *any* tuning activity between measurements, even if a solution to the deteriorating performance was apparent. This ensured that all runs could be compared on a like-for-like basis, with the distance in the sysplex being the only factor that changed.

6.3.1 Transaction response times

Table 6-29, Table 6-30, and Table 6-31 show the transaction rate, average response time, and goal achievement, for all CICS transactions, for the TPOS transaction, and for just the TPNO transaction, respectively, across both systems in the data sharing group. The numbers in these tables were extracted from the RMF Workload Activity reports. You will notice a small decline in the number of transactions being processed, and a noticeable increase in the average response times.

Table 6-29 Information for all CICS transactions - sysplex level

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	278	269			
Resp time (milliseconds)	17	42			
Goal achievement %	100	100			

Table 6-30 Information for TPOS transactions - sysplex level

TPOS	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	9.8	9.5			
Resp time (milliseconds)	4.0	5.0			
Goal achievement %	100	100			

Table 6-31 Information for TPNO transactions - sysplex level

TPNO	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	97	94			
Resp time (milliseconds)	41	100			
Goal achievement %	100	100			

Table 6-32 and Table 6-33 show the response times across all transactions for each of the #@$2 and #@$3 systems respectively. Once again, you can see the reduction in the number of transactions being processed.

Table 6-32 Average response times across all CICS transactions on system #@$2 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139	135			
Resp time (milliseconds)	16	34			
Goal achievement %	100	100			

Table 6-33 Average response times across all CICS transactions on system #@$3 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139	133			
Resp time (milliseconds)	17	49			
Goal achievement %	100	100			

You will also notice that we have started to see a divergence in the response time being delivered by the two systems, with the one that is local to the primary DASD and most of the CF structures (system #@$2) already providing better response times. However, this is a blended number, showing the average across all the different transactions. To gain more insight into what is happening, we need to look in more detail into each of the individual transaction types.

Table 6-34 Average AOR response times for TPOS transaction on system #@$2 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9	4.8			
Resp time (milliseconds)	4.0	4.0			
Goal achievement %	100	100			

Table 6-35 Average AOR response times for TPOS transaction on system #@$3 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9	4.7			

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Resp time (milliseconds)	4.0	6.0			
Goal achievement %	100	100			

The response time and transaction rate of the TPOS transaction is shown in Table 6-34 and Table 6-35. You will recall that this is a light transaction that only makes one call to DB2. While the *relative* response time has increased by up to 50%, the *actual* difference is only a maximum of 2 milliseconds, with a difference of just 2 milliseconds between the two systems. In real terms, this number is so small as to be almost meaningless; transaction response times will normally fluctuate by more than this amount in the course of a normal business day.

Table 6-36 and Table 6-37 show the results for the TPNO transactions. The number of TPNO transactions executed on each system is again very even and not significantly reduced from the 0 km run, although the response time has started to drift apart by a much larger amount than was observed for the TPOS transaction.

Table 6-36 Average AOR response times for TPNO transaction on system #@$2 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49	47			
Resp time (milliseconds)	40	83			
Goal achievement %	100	100			

Table 6-37 Average AOR response times for TPNO transaction on system #@$3 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49	47			
Resp time (milliseconds)	42	117			
Goal achievement %	100	100			

To make it easier to see the changes in response time as the distance is changed, we provide the preceding information in a graphic format in Figure 6-8 on page 116. You can see that the transactions running in system #@$3 were more impacted by the distance than those in system #@$2. Because both systems were impacted to the same degree by the cost of PPRCing over 20 km, it will be interesting to see which components of the response time are responsible for the larger increase in response time on system #@$3 than on #@$2.

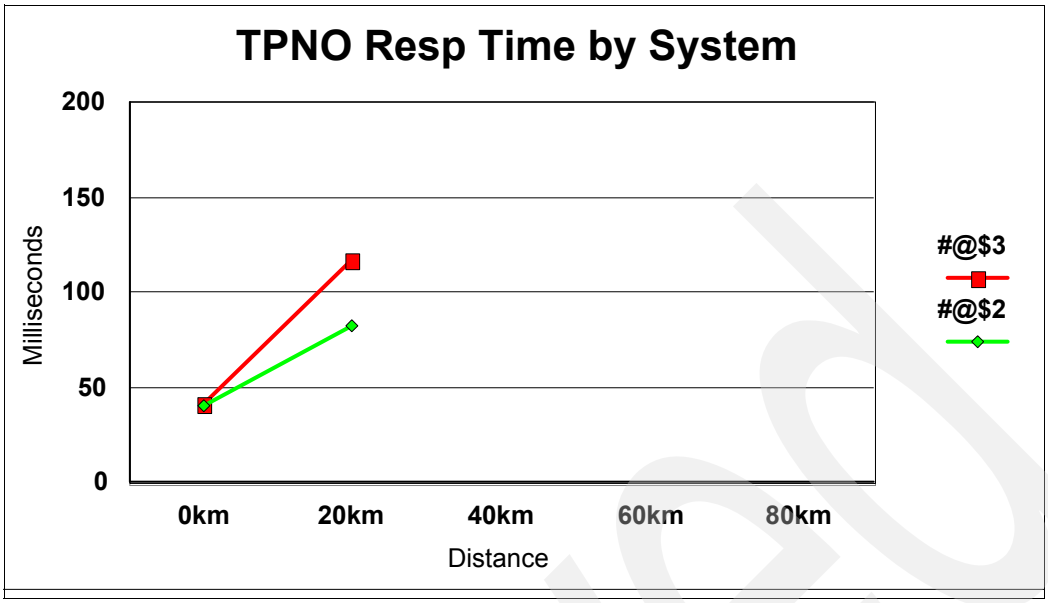


Figure 6-8 TPNO Response time by system for each distance

To verify the validity of the numbers we see in DB2PE and RMF, we also ran a pair of CICS PA reports for the TPNO transaction for each system. One report shows the response time from the perspective of the TORs. The other report shows the response time (and response time breakdown) from the perspective of the AOR.

As you can see in Table 6-38 and Table 6-39, the response time as seen by the AOR continues to closely match the transaction response time as seen by both DB2 and WLM.

Table 6-38 CICSPA response times (in milliseconds) for TPNO transaction for #@$2

	0 km	20 km	40 km	60 km	80 km
#@$2 TOR	41.7	86.5			
#@$2 AOR	41.2	85.2			

Table 6-39 CICSPA response times (in milliseconds) for TPNO transaction for #@$3

	0 km	20 km	40 km	60 km	80 km
#@$3 TOR	44.1	118.3			
#@$3 AORs	43.4	118.5			

As you can see, the response time for the TPNO transactions processed by the CICS AORs on #@$3 is higher than that provided by the TOR on the same system, which may seem unusual (you would normally expect TOR response times to be a little longer than the corresponding AOR ones). This happened because CP/SM has started routing some of the transactions from the #@$3 TOR to the AORs on #@$2 because they are delivering better response times than the AORs on #@$3.

TPNO response time breakout

Next, we look in more detail at the components of the TPNO response time on each system. The following tables summarize some of the data provided by DB2PE Accounting Reports. Table 6-40 on page 117 and Figure 6-9 on page 117 shows the response components on

system #@$2. The numbers in these tables were extracted from the DB2PE Accounting Report.

Table 6-40 Average DB2 times for TPNO transaction on system #@$2*(in milliseconds)*

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.4	31.7			
Synchronous I/O delays	2.2	2.5			
Global contention	7.0	22.4			
Asynchronous CF Requests	2.9	3.1			
Update Commit time	7.1	12.1			
Other miscellaneous DB2 suspends	0.1	0.3			
Other Not Account by DB2	0.4	0.4			
CPU time spent in DB2	4.6	4.8			
Non-DB2 CPU time	1.1	1.2			
Total as seen by DB2	38.4	83.2			

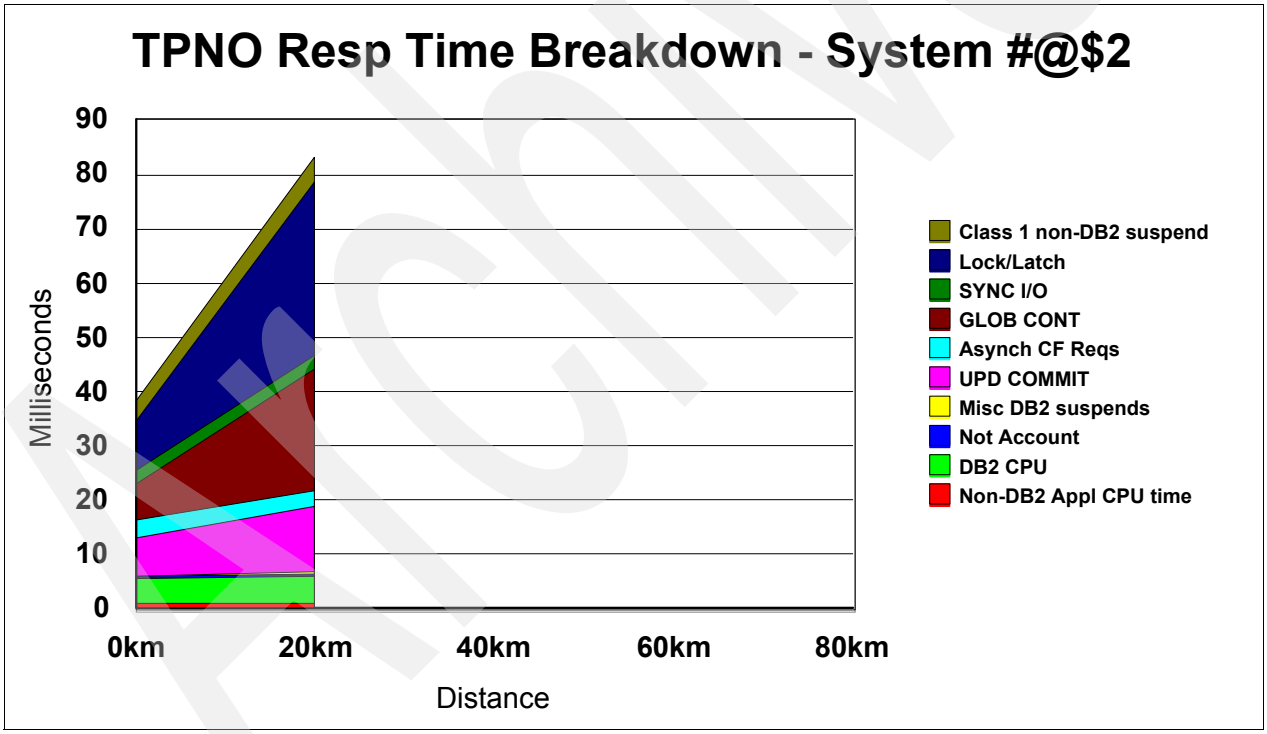


Figure 6-9 TPNO response time breakdown for system #@$2

The equivalent response time breakout for system #@$3 is shown in Table 6-41 on page 117 and Figure 6-10 on page 118.

Table 6-41 Average DB2 times for TPNO transaction on system #@$3 *(in milliseconds)*

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.8	28.5			

Suspend time component	0 km	20 km	40 km	60 km	80 km
Synchronous I/O delays	2.2	3.3			
Global contention	9.6	53.0			
Asynchronous CF Requests	2.6	7.2			
Update Commit time	6.8	11.7			
Other miscellaneous DB2 suspends	0.1	0.4			
Other Not Account by DB2	0.6	2.2			
CPU time spent in DB	4.5	4.6			
Non-DB2 CPU time	1.1	1.2			
Total as seen by DB2	40.5	116.3			

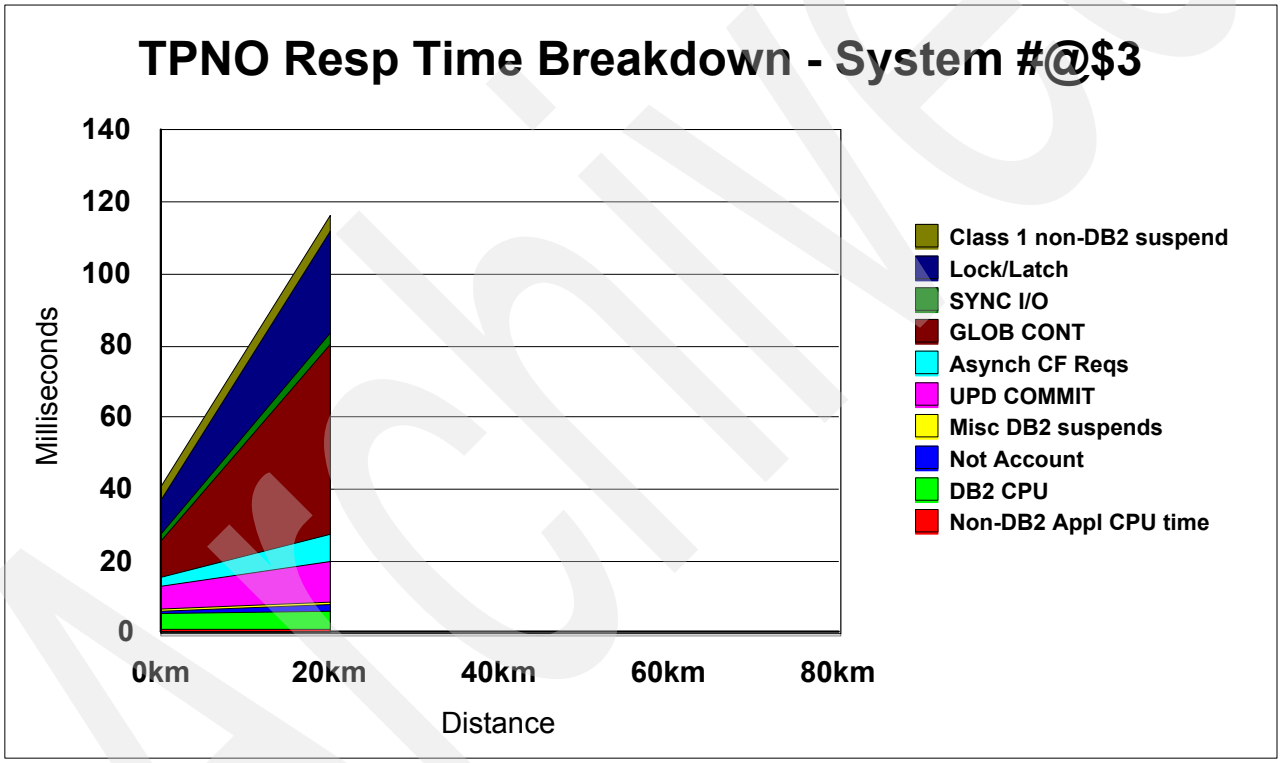


Figure 6-10 TPNO response time breakdown for system #@$3

As you can see in Figure 6-9 on page 117 and Figure 6-10, the response times delivered by the TPNO transaction on the two DB2 subsystems (as reported by DB2PE) have now started to diverge. You can also see that the bulk of the increase is really in a subset of the components; specifically, Lock/Latch, global contention, and Update Commit. Although we review all the components in this section, bear in mind that it is these three that are of most interest.

This is one of the components that saw a large increase when going from 0 km to 20 km.

'Latch

To see the amount of time TPNO is delayed by local contention, refer to Table 6-42. You can see that the average lock/latch suspension time for system #@$2 has increased from 9.4 ms

at 0 km up to 31.7 ms at 20 km, and for system #@$3, from 9.8 ms at 0 km up to 28.5 ms at 20 km.

Table 6-42 Average Lock/Latch suspension time and events for TPNO transaction

Lock/Latch suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	9.4	31.7			
No. of suspension events - #@$2	1.2	1.9			
Total suspension time - #@$3	9.8	28.5			
No. of suspension events - #@$3	1.3	2.2			

We see increases in both suspension time and number of events on both systems. This is not unexpected, given that the transactions are running for longer, meaning that:

- It is more likely that some other transaction will be holding a resource when this transaction wants it, increasing the likelihood of encountering local contention.
- Because all transactions are running for longer, when this transaction has to wait for a prior transaction to release the resource, it will probably have to wait for longer.

The one thing that may be unexpected is that the local contention suspension time is higher on the “local” system (#@$2) than on the remote one. After all, local contention is supposed to be a result of delays in another transaction in the same DB2, and considering that the elapsed time for transactions in #@$2 is less than in #@$3, it is reasonable to expect that local contention time would also be less in #@$2.

However, remember that some events that are reported as local contention may actually be caused by a transaction in another DB2 subsystem. In this case, because the transactions in #@$3 are taking longer, it is possible that some of the increased local contention time in #@$2 is really caused by transactions “in front of us” waiting longer for the transactions in #@$3 to finish and free up the resources we need. On the other hand, a transaction in #@$3 that is being delayed by a transaction in #@$2 will not have to wait quite as long, because the #@$2 transactions are ending in less time.

Once again, though, it is more important to focus on the fact that you are experiencing contention, rather than getting too concerned about whether it is reported as global or local. To understand the total impact of increased contention, we also need to take into account the global contention time.

6.3.3 Synchronous I/O

The next part of the TPNO response time is time spent waiting for synchronous I/O to complete. Remember that these are I/Os:

- For pages needed by this transaction that are not found in either the local or global buffer pools, and
- To write log records prior to commit

To get the amount of time TPNO is waiting for synchronous I/O to complete, refer to Table 6-43 on page 119.

Table 6-43 Average Sync I/O suspend time and events for TPNO transaction

Synchronous I/O suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	2.2	2.5			

Synchronous I/O suspension	0 km	20 km	40 km	60 km	80 km
No. of suspension events - #@$2	2.4	2.3			
Total suspension time - #@$3	2.2	3.3			
No. of suspension events - #@$3	2.5	2.4			

We can see the *average* I/O suspend time for system #@$2 (2.5 ms) has increased by .3 ms from 0 km to 20 km. And the number of events has decreased slightly, from 2.4 to 2.3, meaning that each synchronous I/O took longer at 20 km. Remembering that this includes both database reads (which should have unchanged response times on system #@$2) and log writes, it is likely that the increased average response is due to the impact of PPRC on those write I/Os.

For system #@$3, the average I/O suspend time increased from 2.2 ms at 0 km, up to 3.3 ms at 20 km. Again, the number of I/O suspend events decreased slightly, from 2.5 per thread down to 2.4, again meaning that, on average, each I/O was taking longer. There are two likely reasons for the increased I/O times:

- The response time for disk reads will increase by .1 ms per 10 km between DB2 and the primary disk subsystem. In this measurement, there was 20 km between DB2 and the primary disks, so the synchronous database reads would take .2 ms longer each.
- For the log write I/Os, there is a double impact. First, there is the impact of an additional .2 ms for the I/O to travel from DB2 to the primary subsystem. Then, because these are writes, there is an additional impact of another .2 ms for the write to be mirrored back to the secondary subsystem, 20 km away.

The DB2PE Accounting Report breaks out the number of events that were for database I/O (reads, so they will not have the impact of PPRC), and the number of log write I/Os (writes, so they will have the additional impact of PPRC). Unfortunately, what the report does not provide is information about whether the log write I/Os to the duplex log data sets were issued in parallel or serially.

This component is a good example of the direct effect of distance on response times. It is possible to get a reasonably accurate view of the number of I/Os per thread, and you know that each 10 km will add about .1 ms to the response time for reads, and .2 ms for writes. Using this information, you can make a reasonably accurate projection of how the synchronous I/O suspend time will change with distance.

6.3.4 Global contention

This is another component that saw a large increase when going from 0 km to 20 km.

Table 6-44 on page 121 shows the global contention information for TPNO in both DB2 subsystems at both 0 km and 20 km. The global contention times for both systems for the TPNO transaction started off with similar values at 0 km. However, as the distance increased, the amount of time that TPNO spent waiting on a contention event that was reported as a global contention, as well as the number of reported events, started to diverge quite significantly between the two systems.

As shown in Table 6-44 on page 121, the global contention time on system #@$2 increased from 7 ms up to 22.4 ms. The global contention on system #@$3 increased from 9.6 ms to 53.0 ms.

Table 6-44 Average global contention and events for TPNO transaction

Global contention	0 km	20 km	40 km	60 km	80 km
Reported global contention time - #@$2	7.0	22.4			
Reported number of global contention events - #@$2	7.5	13.0			
Actual no. of global contention Events - #@$2	0.35	0.65			
Reported global contention time - #@$3	9.6	53.0			
Reported number of global contention events - #@$3	32.6	46.2			
Actual no. of global contention events - #@$3	0.35	0.67			

Given that a large part of the cause of global contention is waiting for a task in *another* DB2 subsystem to complete and release the resource, and that the transaction elapsed times were lower on system #@$2, you might expect the transactions on system #@$2 to be more delayed by tasks on system #@$3 than the other way around. Yet what we saw was a much larger increase in global contention on system #@$3 than on system #@$2.

To explain this, we need to look a little closer at the numbers. We see that not only did the total global contention time increase on system #@$3, but the number of global contention events also increased.

Nearly every lock request from system #@$3 will be processed asynchronously and counted by DB2 as a global contention; this is part of the reason for the reported number of events going from 32 up to 46.

If we look at the response time for an average lock request from system #@$3 at 20 km compared to 0 km, we find that it has increased by about 212 microseconds. This is the sort of increase we expected to see (20 km at 10 microseconds per km). Table 6-45 summarizes the behavior of the lock structure for both systems. This information comes from the RMF Structure Activity report.

Table 6-45 RMF structure activity report for DB8QU_LOCK1 structure

	0 km	20 km	40 km	60 km	80 km
#@$2 Sync Rate	5430	4586			
#@$2 Async Rate	693	896			
#@$3 Sync Rate	3358	21			
#@$3 Async Rate	2393	5013			
#@$2 Sync RT	21.2	21.4			
#@$2 Async RT	75.7	69.6			
#@$2 Weighted Average RT	27.1	29.3			
#@$3 Sync RT	21.9	220.7			

	0 km	20 km	40 km	60 km	80 km
#@$3 Async RT	64.5	251.4			
#@$3 Weighted Average RT	39.6	251.2			

So, on system #@$3, we have 45.5 reported global contention events that are really asynchronous converted IXLLOCK requests. Since each of them took an average of 251 microseconds to complete, we can say that for system #@$3, 11.4 ms out of 53 ms are not real contention, while the remaining 41.6 ms are.

Also, to compare the global contention time on the two systems, both the number of reported events (most of which are asynchronous converted IXLLOCK requests) and the weighted average response time are far lower on system #@$2, contributing to the lower global contention value on that system.

The most important point, however, is that there is no metric that we can use to predict the real global contention time. We can make a reasonable guess at the contribution of the converted lock requests; however, they really are only a small part of the total reported global contention time. If we combine the local and global contention times on both systems, they account for well over half (70% on #@$3 and 76% on #@$2) of the total elapsed time for the transaction. We expect that this percentage will increase with distance, but there is no way to predict how large the increase might be.

And we make one final point here about the relationship between local and global contention. Depending on the arrival patterns of the contending transactions, it is possible that contention between a set of transactions might be reported as global contention one time, and local contention another time. Therefore, we strongly advice combining the local contention and global contention delays and treating them as one common problem.

6.3.5 Asynch CF Requests

Asynch CF Requests are the GBP requests that were issued before commit that were sent to the GBP structure asynchronously. For this specific workload, most of those requests were reads, although this is not always the case.

Table 6-46 Average Asynch CF Request suspension time and events for TPNO transaction

Asynch CF Requests	0 km	20 km	40 km	60 km	80 km
Total Asynch CF request suspension time - #@$2	2.9	3.1			
Number of events - #@$2	20.7	20.5			
Total Asynch CF request suspension time - #@$3	2.6	7.2			
Number of events - #@$3	20.8	21.3			

Table 6-46 shows the amount of time TPNO is waiting for Asynch CF Requests. We can see that the average Asynch CF Requests time for system #@$2 has increased marginally, from 2.9 ms at 0 km to 3.1 ms at 20 km. Given that most of the requests included in this counter in our measurement were reads, and all the primary GBPs were in the same site as system #@$2, the small increase was in line with our expectations.

For system #@$3, the response time increased by a larger amount, from 2.6 ms at 0 km up to 7.2 ms at 20 km, which is an increase of a little over 4 ms. You would expect the response time of each read request to increase by roughly 200 microseconds, and system #@$3 had 21.3 events, which brings us very close to the actual increase of 4.6 ms. You can see that Asynch CF Requests is one of the fields that encounters the direct effect of increased distance, and therefore it is relatively easy to predict how it will change with distance[4].

6.3.6 Update Commit

This is the other component that increased significantly with distance.

In the base measurement, the Update Commit time for both DB2 members was about the same, at around 7 ms. If we look at Table 6-47, we see an increase of 5 ms for system #@$2 and a similar increase (4.9 ms) for system #@$3.

Table 6-47 Average Update Commit suspend time for TPNO transaction (in milliseconds)

Update Commit suspensions	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	7.1	12.1			
No. of suspension events - #@$2	1	1			
Total suspension time - #@$3	6.8	11.7			
No. of suspension events - #@$3	1	1			

Log writes

Both systems experienced increased Update Commit times because the increased PPRC distance meant that every write I/O will take longer at this distance than at 0 km. In addition to the PPRC cost, however, system #@$3 had the additional cost of having to communicate over 20 km to send a write to the primary DASD. So, every log write from system #@$3 took twice the impact that was experienced by system #@$2.

To see the average response time for the log write I/Os, we ran an RMF workload activity report for the DB2 MSTR address spaces. Most of the I/Os done by the MSTR address space are for logging, so by looking at the I/O rate (reported as SSCHRT) and the response time (reported as RESP) in the RMF report we can understand the impact of the log write I/Os.

Figure 6-11 contains the report for the D8Q1 DB2 MSTR address space.

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
2732	8	2601	28878	327.1	0.8	0.2	0.5	0.1	0.0

Figure 6-11 Workload activity report for D8Q1 MSTR address space

The corresponding information for the D8Q2 DB2 MSTR address space is shown in Figure 6-12.

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
1866	7	1768	25003	287.5	1.0	0.2	0.5	0.3	0.0

Figure 6-12 Workload activity report for D8Q2 MSTR address space

[4] As mentioned in “Asynch CF Requests” on page 76, due to a reporting bug in DB2, the time spent waiting for asynchronous GBP requests is not always reported in this field. Until the fix for PK27934 is applied, the delays due to asynch CF requests will be added to the Not Accounted field.

You can see that the average response time for each DASD I/O for the D8Q1 MSTR address space (in system #@$2) has increased by .2 ms to 0.8 ms, and the equivalent I/O for the D8Q2 MSTR address space (in system #@$3) has increased by twice that amount (.4 ms) to 1.0 ms. Allowing for writes to both the primary and alternate log data sets, that accounts for roughly 2.0 of the 11.7 ms Update Commit time on system #@$3 and 1.6 ms of 12.1 ms on system #@$2.

GBP writes

As stated in 6.2.6, “Asynch CF Requests” on page 108, because we do not know exactly how many CF requests are generated as a result of the commit, it is not possible to accurately predict how the changed response time for GBP requests will contribute to the increased Update Commit time.

However, we can still make some observations. Most of the requests from the TPNO transaction go to the GBP3 structure, so that is the one we will concentrate on.

Table 6-48 Response time of GBP3 CF requests

System/GBP	0 km	20 km	40 km	60 km	80 km
#@$2 to Primary GBP3	153	221			
#@$2 to Secondary GBP3	279	621			
#@$3 to Primary GBP3	125	321			
#@$3 to Secondary GBP3	220	405			

You will notice in Table 6-48 that the response time for both systems talking to the primary structure has increased. You may have expected the increase from system #@$3, because that system is now 20 km further away from the CF (and its response time has increased by just about 200 microseconds). However, you may not have anticipated the increase for the #@$2 system, given that it has not moved. But remember that when an updated page is written to the primary GBP, and another DB2 has a copy of that page in its local buffer pool, the CF must issue a cross-invalidate signal to that other DB2.

In our case, if DB2 on #@$3 has a copy of a page that #@$2 has updated, the cross-invalidate signal must travel over 20 km. And the response to #@$2 will not be sent back until that cross-invalidate has been successfully issued, resulting in increased response time for those writes.

You may also notice that the response time for #@$2 writes to the secondary GBP are taking longer than you may have expected. This is because of a behavior that is unique to DB2 V8, whereby it sends registration information for some pages to the secondary GBP. This results in cross-invalidates being issued from the secondary GBP, increasing the response times for writes to that structure instance. In versions of DB2 prior to, and newer than, DB2 V8, DB2 did not send registration information to the secondary GBP, so these unexpectedly high response times are unique to V8.

Another consideration is that the Update Commit time includes at least one request to the CF for each pageset containing pages updated by this transaction. Although DB2 can write multiple pages in a single GBP write request (which is especially beneficial when there is a large distance between DB2 and the GBP), all the pages must belong to the same pageset. So, if DB2 has updated multiple pagesets in a single GBP, that will result in multiple GBP writes. As the distance between the systems and the CFs increases, this is going to play an increasingly important role in the total Update Commit time for both systems.

If we look at the response times for GBP3 in Table 6-48, at 0 km distance and at 20 km, we find that the response time for the write to the “local” CF has remained fairly constant for system #@$2, but it has increased significantly for system #@$3’s write to the secondary GBP structure. This may seem surprising, given that the distance from system #@$3 to the CF containing the secondary GBP has not changed.

However, keep in mind that a CF request to update a page in the GBP is not complete until any cross-invalidates have been sent. So, while the distance from system #@$3 to CF FACIL06 has not changed, the distance that any cross-invalidates from FACIL06 to system #@$2 must travel *has* changed, resulting in the increased response times we are seeing for the secondary GBP structure from both systems.

Release locks

The next part of Update Commit processing is releasing the locks help by the thread. Although we do not know exactly how long this process takes (unless we run a trace), we do not believe it is as significant as writing to the GBPs and log data sets, so we will not focus on this processing.

Cleanup

The final part of Update Commit is the DB2 cleanup processing. Compared to the other three parts of Update Commit, this should complete very quickly; in one trace we ran, this processing took just two milliseconds. Also, because it does not involve going outside the processor, it should be unaffected by any changes in the distance within the sysplex.

6.3.7 Not Account time

Table 6-49 shows the amount of Not Accounted time for TPNO. We can see that the average Not Account time for system #@$2 is steady at 0.4 ms. For system #@$3, the Not Account time went up from 0.6 ms for 0 km to 2.2 ms for 20 km.

Table 6-49 Average Not Account time (in milliseconds) for TPNO transaction

Not Account time	0 km	20 km	40 km	60 km	80 km
Not Accounted - #@$2	0.4	0.4			
Not Accounted - #@$3	0.6	2.2			

Remember that Not Accounted time contains time that DB2 is unable to classify, so, by definition, we are unable to explain why it is increasing on system #@$3. However you will also recall that Not Accounted time is expected to remain at a fairly consistent percentage of overall transaction response time. Therefore, as the response time of the TPNO transaction on system #@$3 increases, we will expect this time to increase as well.

6.3.8 DB2 CPU time

Table 6-50 contains the DB2 CPU time for TPNO. We can see that the average DB2 CPU time for system #@$2 has increased marginally from 4.6 ms at 0 km up to 4.8 ms at 20 km. For system #@$3, DB2 CPU time stays almost the same: 4.5 ms at 0 km and 4.6 ms at 20 km. Remember that this is only part of the CPU time spent by DB2. It is actually the CPU spent directly on behalf of the transaction (Class 2 CPU Time in DB2 terms), and it is charged back to the requestor (the “allied address space” in DB2 terms).

Table 6-50 Average DB2 CPU time (in milliseconds) for TPNO transaction

DB2 CPU time	0 km	20 km	40 km	60 km	80 km
DB2 CPU time - #@$2	4.6	4.8			
DB2 CPU time - #@$3	4.5	4.6			

6.3.9 Overall Coupling Facility activity

The CPU utilization of each CF can be found in the RMF Coupling Facility report. You can see in Table 6-51 that for this measurement, both CFs were very lightly loaded, and the utilization did not change significantly when moving from 0 km to 20 km.

Table 6-51 Coupling Facility CPU utilization

	0 km	20 km	40 km	60 km	80 km
FACIL05	14.3	14.1			
FACIL06	4.9	5.7			

The CF subchannel utilization numbers, shown in Table 6-52 and Table 6-53, can be calculated from the values provided in the RMF SUBCHANNEL ACTIVITY report, which is provided after the detail structure activity section of the RMF CF Activity report.

Table 6-52 FACIL05 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	3.5%	4.9%			
From #@$3	3.4%	13.8%			

Table 6-53 FACIL06 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	1.2%	3.3%			
From #@$3	1%	1.9%			

As you can see, the utilization of *all* links has increased. When the distance between z/OS and the CF increases, you would expect to see increases in subchannel utilization. However, we also observed increased utilization on the links between z/OS and the CF that is in the same site. There are two reasons for this increase on the “local” links:

- The cross-invalidates to the DB2 in the remote site result in increased response times for the requests to the local CF.
- The increased DB2 contention resulted in more XCF messages, with a significant number of those going through the XCF signalling structures in the CF, thereby increasing CF utilization.

The subchannel utilization is still comfortably below the IBM-recommended threshold of 30% busy. However, at just 20 km, we have seen that the links between #@$3 and FACIL05 are already running at 13.8% busy, so these should be monitored as more distance is injected into the sysplex.

6.3.10 XCF signaling

As discussed in 2.5.1, “XCF/XES role as global lock manager” on page 20, when DB2 issues a lock or unlock request for a lock entry that is in contention, the request gets passed to the global lock manager using XCF signals. Therefore, as the number of lock contention events increases, we see an increase in the number of related XCF messages.

As can be seen in Table 6-54, the number of XCF messages sent from system #@$2 to #@$3 in the DEFAULT transport class increased from 168.5 to 277.3, and almost all of them went over the XCF CF structures.

Table 6-54 XCF messages sent per second from #@$2 to #@$3

	0 km	20 km	40 km	60 km	80 km
CF structures	144.2	277.0			
FICON CTC	24.3	0.3			

Actually, most of them went through a single CF structure, which is the one placed in the CF which was close to #@$3, the receiving system. As explained in 2.4, “CTCs” on page 18, XCF will automatically select the path(s) that are delivering the best end-to-end response times. Because of the way we had our CTCs configured at the time of the measurement, the response time to send XCF messages over the CTC path was about 200 microseconds longer than when using the XCF structures in the CF. The result is the unbalanced utilization that is shown in Table 6-54.

As can be seen in Table 6-55, the number of XCF messages sent from system #@$3 to #@$2 in the DEFAULT transport class also increased from 170.6 to 287.1.

Table 6-55 XCF messages sent per second from #@$3 to #@$2

	0 km	20 km	40 km	60 km	80 km
CF structures	126.5	158			
FICON CTC	44.1	129.1			

You can also see that a far larger percent of requests used the CTC when going in this direction than when going from #@$2 to #@$3. Because of how our CTCs were configured, the response time delivered by the CTC when going from #@$3 to #@$2 was very close to the times delivered by the XCF signalling structures, resulting in the more even usage of CTCs and the signalling structures in the CF.

6.3.11 z/OS CPU utilization

Finally, we include overall z/OS CPU utilization and the z/OS capture ratio. The CPU utilization numbers for each system are included in Table 6-56.

Table 6-56 z/OS CPU utilization

	0 km	20 km	40 km	60 km	80 km
#@$2	45.2%	49.1%			
#@$3	46.5%	48.7%			

To determine whether increased distance has any effect on capture ratio, we also included the capture ratios for each distance in Table 6-57.

Table 6-57 z/OS capture ratios

	0 km	20 km	40 km	60 km	80 km
#@$2	90%	90%			
#@$3	91.2%	90.2%			

The z/OS CPU utilization increased on both systems, as would be expected considering the changes in the CPU usage of the various DB2 address spaces. The capture ratio was unchanged on system #@$2. However, it decreased by 1% on the system that was moved 20 km from the primary DASD and most of the CF structures. It will be interesting to see if this decline continues as the distance increases.

6.4 Measurement at 40 km

For the next set of measurements, at 40 km, the configuration was unchanged from that used for the base measurement, with the exception that a pair of DWDMs with 40 km of fiber between them was added to the configuration.

6.4.1 Transaction response times

Table 6-58, Table 6-59, and Table 6-60 show the transaction rate, average response time, and goal achievement, for all CICS transactions, for the TPOS transaction, and for just the TPNO transaction, respectively, across both systems in the data sharing group. All CICS transactions continued to have a WLM goal of 90% of transactions to complete in .5 seconds, and this was the first measurement where the goal achievement was less than 100%.

Table 6-58 Information for all CICS transactions - sysplex level

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	278	269	249		
Resp time (milliseconds)	17	42	104		
Goal achievement %	100	100	96.6		

Table 6-59 Information for TPOS transactions - sysplex level

TPOS	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	9.8	9.5	8.9		
Resp time (milliseconds)	4.0	5.0	7.0		
Goal achievement %	100	100	100		

Table 6-60 Information for TPNO transactions - sysplex level

TPNO	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	97	94	87		

TPNO	0 km (base)	20 km	40 km	60 km	80 km
Resp time (milliseconds)	41	100	251		
Goal achievement %	100	100	91.8		

Table 6-61 on page 129 shows the response times across all transactions for the #@$2 system.

Table 6-61 Average response times across all CICS transactions on system #@$2 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139	135	127		
Resp time (milliseconds)	16	34	88		
Goal achievement %	100	100	97.5		

Table 6-62 on page 129 shows the response times across all transactions for the #@$3 system.

Table 6-62 Average response times across all CICS transactions on system #@$3 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139	133	122		
Resp time (milliseconds)	17	49	120		
Goal achievement %	100	100	95.7		

Notice that the number of transactions completed is, once again, fairly even across both systems, but the overall average response times are continuing to diverge. However, the transaction rates and response times for the TPOS transaction, as shown in Table 6-63 and Table 6-64, are affected to a far smaller degree than the TPNO transactions, both in relative and absolute terms.

Table 6-63 Average AOR response times for TPOS transaction on system #@$2 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9	4.8	4.5		
Resp time (milliseconds)	4.0	4.0	6.0		
Goal achievement %	100	100	100		

Table 6-64 Average AOR response times for TPOS transaction on system #@$3 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9	4.7	4.4		
Resp time (milliseconds)	4.0	6.0	8.0		
Goal achievement %	100	100	100		

If we look at the TPNO transaction performance on each system, as shown in Table 6-65 and Table 6-66 on page 130, the transaction rate is down by about 15% compared to the measurement at 0 km, but the response time is up by nearly 600% on system #@$3.

Table 6-65 Average AOR response times for TPNO transaction on system #@$2 (in milliseconds)

TPNO transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49	47	44		
Resp time (milliseconds)	40	83	212		
Goal achievement %	100	100	94.3		

Table 6-66 Average AOR response times for TPNO transaction on system #@$3 (in milliseconds)

TPNO transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49	47	43		
Resp time (milliseconds)	42	117	289		
Goal achievement %	100	100	89.4		

To make it easier to see the changes in response time as the distance is changed, we provide the preceding information in a graphic format in Figure 6-13. You can see that the transactions running in system #@$3 were more impacted by the distance than those in system #@$2. Because both systems were impacted to the same degree by the cost of PPRCing, it will be interesting to see which components of the response time increased more for system #@$3 than for #@$2.

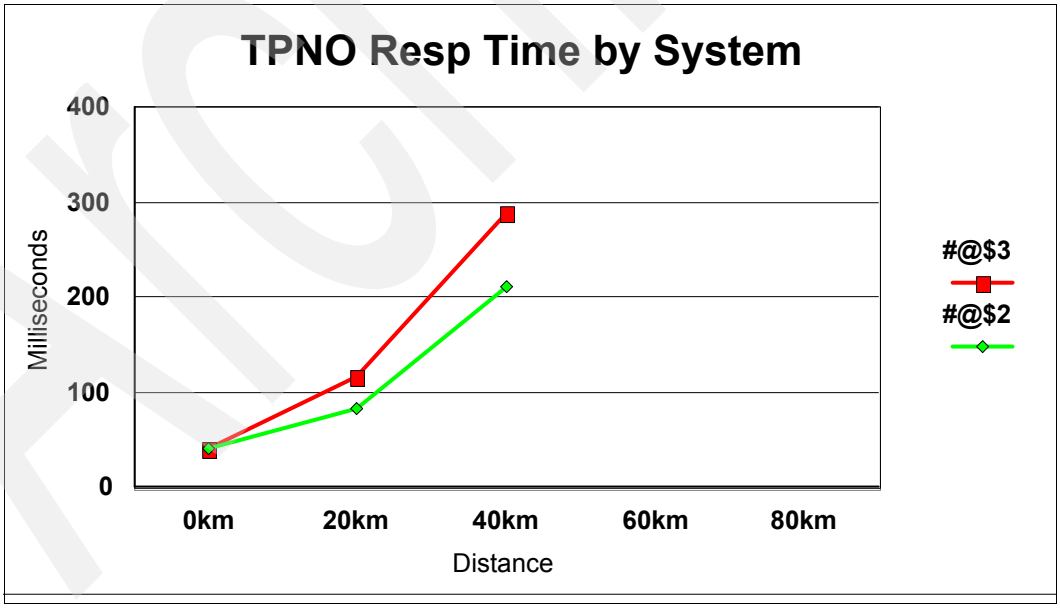


Figure 6-13 TPNO Response time by system for each distance

To verify the numbers we see in DB2PE and RMF, we also ran a pair of CICS PA reports for the TPNO transaction for each system. One report shows the response time from the perspective of the TORs. The other report shows the response time (and response time breakdown) from the perspective of the AOR. As you can see in Table 6-67 and Table 6-68 on

page 131, the response time as seen by the AOR continues to closely match the transaction response time as seen by both DB2 and WLM.

Table 6-67 CICSPA response times (in milliseconds) for TPNO transaction for #@$2

	0 km	20 km	40 km	60 km	80 km
#@$2 TOR	41.7	86.5	215.3		
#@$2 AOR	41.2	85.2	214.6		

Table 6-68 CICSPA response times (in milliseconds) for TPNO transaction for #@$3

	0 km	20 km	40 km	60 km	80 km
#@$3 TOR	44.1	118.3	287.8		
#@$3 AORs	43.4	118.5	290.7		

As you can see, the response time for TPNO transaction processed by the CICS AORs of #@$3 is higher than that provided by the TOR on the same system. This happened because some work coming from #@$3 has been routed for execution by CP/SM on the AORs of #@$2.

TPNO response time breakout

Next, we look in more detail at the components of the TPNO response time on each system. Table 6-69 and Figure 6-14 on page 132 show the response components on system #@$2. The numbers in these tables were extracted from the DB2PE Accounting Report.

Table 6-69 Average DB2 times for TPNO transaction on system #@$2 (in milliseconds)

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.4	31.7	124		
Synchronous I/O delays	2.2	2.5	3.1		
Global contention	7.0	22.4	47.6		
Asynchronous CF Requests	2.9	3.1	3.9		
Update Commit time	7.1	12.1	17.4		
Other miscellaneous DB2 suspends	0.1	0.3	0.9		
Other Not Account by DB2	0.4	0.4	0.6		
CPU time spent in DB2	4.6	4.8	4.9		
Non-DB2 CPU time	1.1	1.2	1.2		
Total as seen by DB2	38.4	83.2	207.7		

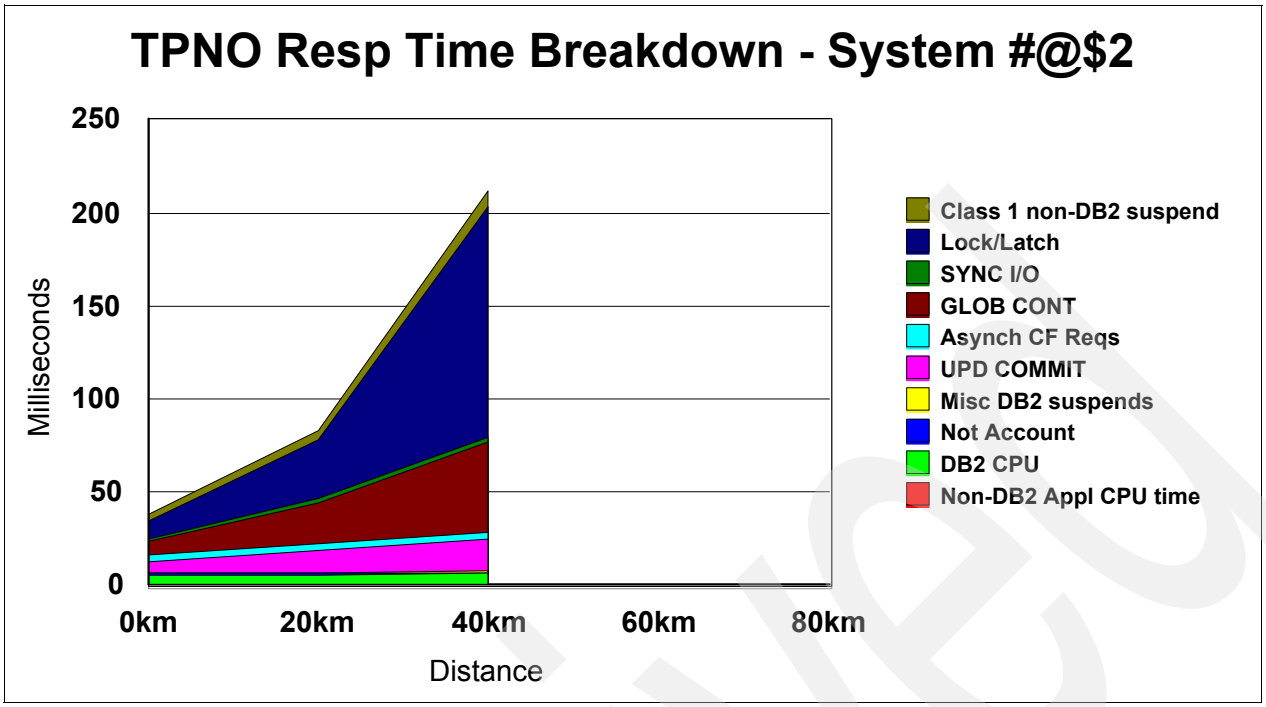


Figure 6-14 TPNO response time breakdown for system #@$2

The equivalent response time breakout for system #@$3 is shown in Table 6-70 on page 132 and Figure 6-15 on page 133.

Table 6-70 Average DB2 times for TPNO transaction on system #@$3*(in milliseconds)*

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.8	28.5	132.5		
Synchronous I/O delays	2.2	3.3	4.4		
Global contention	9.6	53.0	106		
Asynchronous CF Requests	2.6	7.2	11.7		
Update Commit time	6.8	11.7	16.6		
Other miscellaneous DB2 suspends	0.1	0.4	1.3		
Other Not Account by DB2	0.6	2.2	4.3		
CPU time spent in DB2	4.5	4.6	4.9		
Non-DB2 CPU time	1.1	1.2	1.2		
Total as seen by DB2	40.5	116.3	288.3		

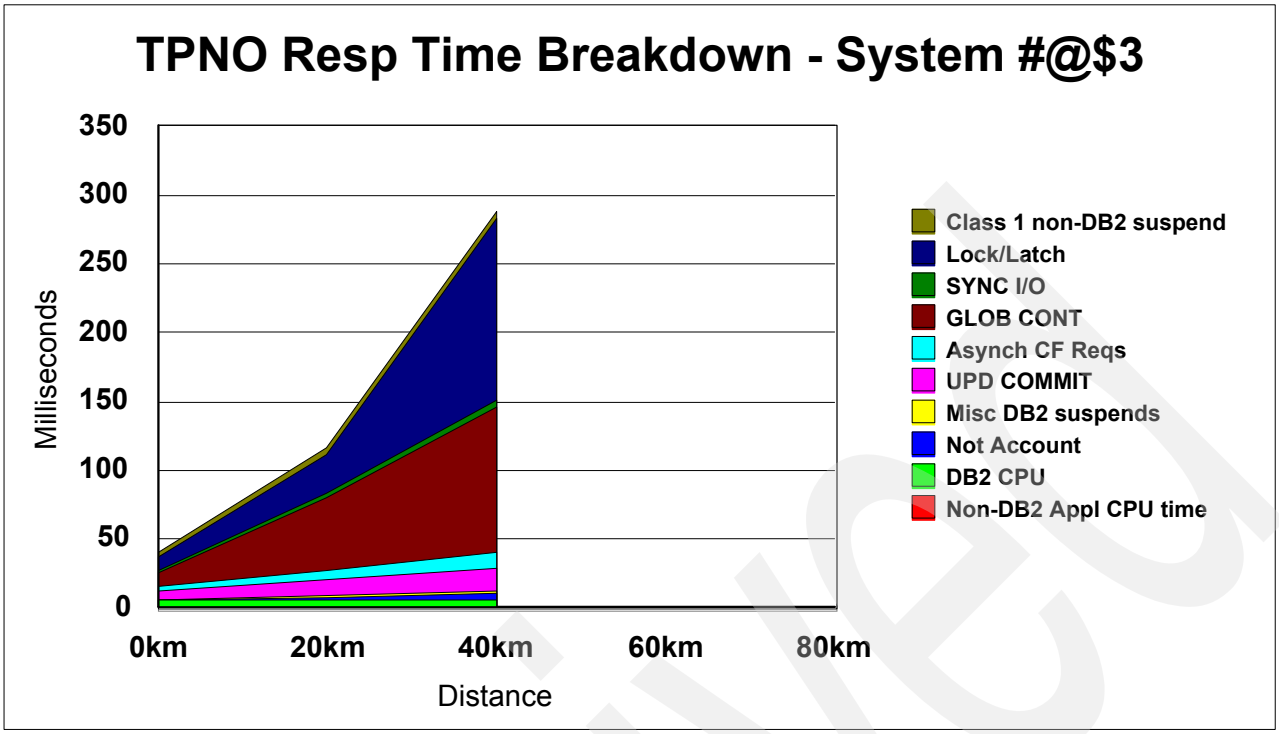


Figure 6-15 TPNO response time breakdown for system #@$3

As you can see in Figure 6-14 on page 132 and Figure 6-15, the response times delivered by the two DB2 subsystems (as reported by DB2PE) continue to diverge. However, we can see clearly now that the overwhelming majority of the TPNO response time on both systems is caused by local and global contention. Although Update Commit is still growing, it is no longer a significant part of the overall response time. Some of the other components have also grown, as you would expect, but none of them are even close to these two contention reasons in terms of impact to the overall response time.

6.4.2 Lock/Latch

In this measurement we see a very significant increase in the duration of local contention events on both systems. There was also an increase in the number of events, but that does not account for all of the increase in local contention time.

Table 6-71 Average Lock/Latch suspension time and events for TPNO transaction

Lock/Latch suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	9.4	31.7	124		
No. of suspension events - #@$2	1.2	1.9	3		
Total suspension time - #@$3	9.8	28.5	132.5		
No. of suspension events - #@$3	1.3	2.2	4.3		

Table 6-71 shows the amount of time TPNO was delayed by local contention. We can see that the average time per Lock/Latch suspend for system #@$2 has increased from 16.7 ms at 20 km up to 41 ms at 40 km. For system #@$3, it has increased from 12.9 ms up to 30.8 ms. As with the 20 km measurement, we see that the local contention delays are longer on system #@$2.

Once again, we observe a significant increase in local contention wait time, but we have no metric to help us predict how many of these events we will encounter, or how long each will last. If Lock/Latch was a trivial part of the overall response time, the fact that we cannot predict it might not matter so much. But at this point, it appears that Lock/Latch is going to be the largest single component of the response time, and nearly half the overall response time. It is becoming apparent that we have a major contention problem, and we address this problem when we provide the results of the 80 km measurement.

6.4.3 Synchronous I/O

The next part of the TPNO response time is time spent waiting for synchronous I/Os to complete. Remember that these are I/Os for database pages that are not in the local or global buffer pools, and I/Os to write log records that are created before the commit.

Table 6-72 Average Sync I/O suspend time and events for TPNO transaction

Synchronous I/O suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	2.2	2.5	3.1		
No. of suspension events - #@$2	2.4	2.3	2.5		
Total suspension time - #@$3	2.2	3.3	4.4		
No. of suspension events - #@$3	2.5	2.4	2.6		

The synchronous I/O suspend times and the number of events are shown in Table 6-72. We can see the average I/O suspend time for system #@$2 has increased from 1.9 ms at 0 km up to 3.1 ms at 40 km. The number of suspend events is pretty consistent throughout the measurements to date, indicating that on average, each I/O is taking a little longer. Given that some of these I/Os are writes (which are affected by the PPRC distance increase), this is what you would expect.

For system #@$3, the average I/O suspend time has increased from 2.2 ms at 0 km up to 4.4 ms at 40 km. Both the database read I/Os and the log write I/Os will be impacted by the increased distance between system #@$3 and the primary DASD. In addition, the log writes carry the cost of PPRC over 40 km.

These results are consistent, and conform to the predicted increase of .4 ms per write I/O on system #@$2, and .4 ms per read I/O and .8 ms per write I/O for system #@$3.

6.4.4 Global contention

Referring to Figure 6-14 on page 132 and Figure 6-15 on page 133, we can see that the two major components of the TPNO response time, on both systems, are local contention (Lock/Latch) and global contention. Table 6-73 shows the global contention suspension times, the *reported* number of global contention events, and the *actual* number of global contention events.

Table 6-73 Average global contention time and events for TPNO transaction

Global contention	0 km	20 km	40 km	60 km	80 km
Reported global contention time - #@$2	7.0	22.4	47.6		
Reported number of global contention events - #@$2	7.5	11.4	14.5		

Global contention	0 km	20 km	40 km	60 km	80 km
Actual no. of global contention events - #@$2	0.35	0.66	.7		
Reported global contention time - #@$3	9.6	53.0	106		
Reported number of global contention events - #@$3	32.6	23.5	45.7		
Actual no. of global contention events - #@$3	0.35	0.74	.9		

Part of the increase in the global contention time can be attributed to increases in the lock response time, shown in Table 6-74, as well as an increase (on both systems) in the number of asynchronous lock requests. However, even allowing for these increases, there was still a sizeable increase in the underlying global contention wait time.

Table 6-74 RMF structure activity report for DB8QU_LOCK1 structure

System	Synch Rate	Synch Serv Time (mics)	Asynch Rate	Asynch Serv Time (mics)
#@$2	4381.2	21.5	825	66.4
#@$3	21.6	416.3	4476.6	455.9

On system #@$2, there were 13.8 (14.5 - .7) asynchronous lock requests, at 66.4 microseconds each. Removing these from the reported 47.6 ms of reported global contention time leaves us with an actual global contention delay time of 46.6 ms on system #@$2.

On system #@$3, there were 44.8 (45.7 - .9) asynchronous lock requests, at 455.9 microseconds each. Removing these from the reported 106 ms of reported global contention time leaves us with an actual global contention delay time of 85.6 ms on that system.

We can make the following observations:

- The actual number of global contention events per transaction is increasing with distance, but slowly; we still have an average of less than one global contention event per transaction.
- When we *do* encounter global contention, the impact is getting greater.
- The only other component of the overall TPNO response time that is growing as quickly as global contention is local contention.
- We do not have *any* methodology for predicting the change in these two components.
- Increases in both of these components reflect the *secondary* effect of distance. The transaction is not encountering more contention because *its* I/Os and CF requests are taking longer. It is encountering increased contention because *other* transactions that share resources that this transaction needs are taking longer.

The one positive piece of news is that the same root cause, contention, is responsible for the two largest components of the transaction response time. So, if we can address the contention, then hopefully both components will be helped.

6.4.5 Asynch CF Requests

The Asynch CF Requests suspension field reports on the number of asynchronous requests (and their elapsed time) that were issued prior to the commit.

Table 6-75 Average Asynch CF Request suspension time and events for TPNO transaction

Asynch CF Requests	0 km	20 km	40 km	60 km	80 km
Total Asynch CF request suspension time - #@$2	2.9	3.1	3.9		
Number of events - #@$2	20.7	20.5	20.4		
Total Asynch CF request suspension time - #@$3	2.6	7.2	11.7		
Number of events - #@$3	20.8	21.3	21.4		

To get the Asynch CF Requests values for TPNO, refer to Table 6-75. We see that the average Asynch CF Request time for system #@$2 has increased from 2.9 ms at 0 km up to 3.9 ms at 40 km. Although this field can potentially contain both reads and writes, in our case, it contains mostly (but not all) reads. The response of the read requests should be largely unchanged from the 0 km measurement. However, the response time of the writes have increased by about .7 ms per request, accounting for most of the increased response time at 40 km compared to 0 km.

Table 6-76 Response time of GBP3 CF requests

System/GBP	0 km	20 km	40 km	60 km	80 km
#@$2 to Primary GBP3	153	221	275		
#@$2 to Secondary GBP3	279	621	987		
#@$3 to Primary GBP3	125	321	525		
#@$3 to Secondary GBP3	220	405	525		

The average Asynch CF Request time for system #@$3 has increased from 2.6 ms at 0 km to 11.7 ms at 40 km. In this case, the response time for the GBP reads will have increased because system #@$3 is now 40 km further from the CF, and the response time for writes to *both* GBP structures has increased. With any version of DB2 other than V8, the writes to the secondary GBP from system #@$3 would be relatively short (because they are both in the same site). However, because of a bug in DB2 V8, the secondary GBP contains registration information, meaning that a write to the secondary GBP may result in a cross-invalidation to system #@$2. This explains why the response time for the #@$3 writes to the secondary GBP are so long, as shown in Table 6-76 on page 136. If we take the number of Asynch CF Requests on system #@$3 (21.4) and multiply that by the response time of either structure (525), that sums to about 11 ms, which is roughly the Asynch CF Request time for that system.

Once again, we see that the amount of time spent waiting for Asynch CF Requests is predictable, and easily calculated. The response times for the secondary GBPs are higher than we expected, but this is primarily as a result of the secondary GBPs issuing the cross-invalidates.

6.4.6 Update Commit time

In the base measurement, the Update Commit time for both DB2 members was about the same, at 7 ms. If we look at the corresponding values for both systems for the 40 km measurement (shown in Table 6-77), we see that both systems have increased by about 10 ms.

Table 6-77 Average Update Commit suspend time for TPNO transaction (in milliseconds)

Update Commit suspensions	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	7.1	12.1	17.4		
No. of suspension events - #@$2	1	1	1		
Total suspension time - #@$3	6.8	11.7	16.6		
No. of suspension events - #@$3	1	1	1		

Log writes

We stated previously that it is valid to assume that one log record will be created as part of commit processing. At 0 km, we saw that the response time for the log writes was about .6 ms on both systems.

For the 40 km measurement, the response time for the log writes would be expected to increase by about .4 ms (the time to mirror the write to the secondary DASD). In Figure 6-16 on page 137 you can see that the response time for the log write I/Os at 40 km was, in fact, 1 ms, exactly meeting the expectation.

For system #@$3, the log write I/Os is expected to increase by .4 ms for the distance between #@$3 and the primary DASD, and another 4 ms to mirror the write back to the secondary DASD. In the RMF workload activity in Figure 6-17 on page 137, we can see that the response time for the log write I/Os from #@$3 have increased to the expected response time of 1.4 ms. Allowing for the log records being written to DASD serially, this means that the log write portion of Update Commit accounts for about 2.8 ms of the 16.6 ms Update Commit time on system #@$3.

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
2680	16	2452	27143	301.4	1.0	0.2	0.7	0.1	0.0

Figure 6-16 Workload activity report for D8Q1 MSTR address space

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
1535	18	1357	18406	215.3	1.4	0.2	0.7	0.5	0.0

Figure 6-17 Workload activity report for D8Q2 MSTR address space

GBP writes

The second component of Update Commit time is the time required to write the updated pages to the primary and secondary GBPs. Although we do not know exactly how many requests were issued during commit processing, there is a way to arrive at the approximate number. However, the process is quite complex, so instead of repeating it over and over for each distance, we will hold off until we are discussing the results of the last measurement, at

80 km. For now, we will simply work on the basis that there were an average of 12 GBP writes on each system during commit processing.

On system #@$2, the total update commit time was 17.4 ms. Using the logic that was introduced in “GBP writes” on page 79, we multiply the number of writes by the response time that system #@$2 was observing for write requests to the secondary GBP (987 microseconds as shown in Table 6-78), giving us about 12 ms spent writing to the GBPs during commit processing.

On system #@$3, the total Update Commit time was 16.6 ms. Using the same logic for determining response times, we again multiply 12 writes by the response time system #@$2 was observing for the secondary structure. This again gives us about 12 ms spent writing to the GBPs.

Table 6-78 Response time of GBP3 CF requests

System/GBP	0 km	20 km	40 km	60 km	80 km
#@$2 to Primary GBP3	153	221	275		
#@$2 to Secondary GBP3	279	621	987		
#@$3 to Primary GBP3	125	321	525		
#@$3 to Secondary GBP3	220	405	525		

Note: In reality, because TPNO writes to a number of GBPs, we should get a weighted average response time across all the GBPs and use that number. However, because we are only using approximate numbers here, it will suffice to use the response times for the busiest GBP (GBP3).

Release locks

The next part of Update Commit processing is releasing the locks held by the thread. Although we do not know exactly how long this process takes (unless we run a trace), we do not believe it is as significant as writing to the GBPs and log data sets, so we will not focus on this processing.

Cleanup

The final part of Update Commit is the DB2 cleanup processing. Compared to the other three parts of Update Commit, this should complete very quickly; we measured this processing (using a trace) at about 2 ms.

Summing the components of Update Commit, we get 2.8 ms (log writes) plus 12 ms (GBP writes) plus 2 ms (Cleanup), giving us about 16.8 ms, which is very close to the DB2-reported numbers of 16.6 ms and 17.4 ms.

6.4.7 Not Account

Table 6-79 shows the amount of Not Accounted time for TPNO. We can see that the average Not Accounted time for system #@$2 has increased marginally, from 0.4 ms at 0 km up to .6 ms at 40 km. On system #@$3, it increased from .6 ms at 0 km up to 4.3 km at 40 km.

Table 6-79 Average Not Accounted (in milliseconds) for TPNO transaction

Not Account time	0 km	20 km	40 km	60 km	80 km
Not Accounted - #@$2	.4	.4	.6		
Not Accounted - #@$3	.6	2.2	4.3		

Remember that Not Accounted time contains time that DB2 is unable to classify so, by definition, we are unable to explain why it is increasing on system #@$3. However, you will also recall that Not Accounted time is expected to remain at a fairly consistent percentage of overall transaction response time. Therefore, as the response time of the TPNO transaction on system #@$3 increases, we will expect this time to increase as well.

6.4.8 DB2 CPU time

Table 6-80 contains the DB2 CPU time for TPNO. We can see that the average DB2 CPU time for system #@$2 has increased marginally from 4.6 ms at 0 km up to 4.9 ms at 40 km. For system #@$3, DB2 CPU time stays almost the same, 4.5 ms at 0 km and 4.9 ms at 40 km. Remember that this is only part of the CPU time spent by DB2.

Table 6-80 Average DB2 CPU time (in milliseconds) for TPNO transaction

DB2 CPU time	0 km	20 km	40 km	60 km	80 km
DB2 CPU time - #@$2	4.6	4.8	4.9		
DB2 CPU time - #@$3	4.5	4.6	4.9		

DB2 subsystem address spaces

6.4.9 Overall Coupling Facility activity

In addition to the performance of the specific DB2 structures, it is also important to monitor the overall health, performance, and capacity of the Coupling Facility. The metrics that we specifically monitored in the performance runs were CF CPU utilization and subchannel utilization.

The CPU utilization of each CF can be found in the RMF Coupling Facility report. As noted, the number of DB2 transactions has been steadily decreasing as the distance has increased. You may also remember that the only structures in the FACIL06 CF were the secondary GBPs and the XCF structures. Additionally, recall that contention has been increasing with distance, and that increased contention results in increased XCF messaging.

Referring to Table 6-81, you see that the utilization in the CF containing the DB2 lock structure and the primary GBPs is declining, and the utilization of the CF that contains some of the XCF signalling structures is increasing.

Table 6-81 Coupling Facility CPU utilization

	0 km	20 km	40 km	60 km	80 km
FACIL05	14.3%	14.1%	13%		
FACIL06	4.9%	5.7%	5.9%		

You may remember that the IBM-recommended threshold for CF subchannel utilization is 30%. The CF subchannel utilization numbers are shown in Table 6-82 and Table 6-83. There are a few interesting points here:

- Even though the utilization of FACIL05 is decreasing, the subchannel utilization is increasing.
- Even though system #@$2 is in the same site as FACIL05, and the distance between the two LPARs has not changed, the subchannel utilization has increased by 50%.
- The subchannel utilization for system #@$3 talking to CF FACIL05 has now increased to 21.7%, nearly a seven-fold increase from the 0 km number, even though the CPU utilization of FACIL05 is actually decreasing.

We continue to closely monitor the CF subchannel utilization as the distance continues to increase.

Table 6-82 FACIL05 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	3.5%	4.9%	5.7%		
From #@$3	3.4%	13.8%	21.7%		

Table 6-83 FACIL06 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	1.2%	3.3%	5.8%		
From #@$3	1%	1.9%	2.6%		

6.4.10 XCF signaling

As discussed in 2.5.1, “XCF/XES role as global lock manager” on page 20, when DB2 issues a lock or unlock request for a lock entry that is in contention, the request gets passed to the global lock manager using XCF signals. Therefore, as the number of lock contention events increases, you would expect an increase in the number of related XCF messages. In this section we track the number of XCF messages, and the distribution over the available CF structure and FICON CTC paths.

As can be seen in Table 6-25 on page 112, the number of XCF messages sent from system #@$2 to #@$3 in the DEFAULT transport class has increased from 168 up to 322, nearly all of which went over the XCF signalling structures.

Table 6-84 XCF messages sent per second from #@$2 to #@$3

	0 km	20 km	40 km	60 km	80 km
CF structures	144.2	277.0	321.2		
FICON CTC	24.3	0.3	1.2		

As can be seen in Table 6-26 on page 112, the number of XCF messages sent from system #@$3 to #@$2 in the DEFAULT transport class also increased, from 170 to 332. In this case, interestingly, we are seeing a trend where a larger percentage of XCF messages are being sent over CTCs as the distance increases. The reason we did not see this on system #@$2 was because of how we had our CTCs configured at the time of these measurements.

Table 6-85 XCF messages sent per second from #@$3 to #@$2

	0 km	20 km	40 km	60 km	80 km
CF structures	126.5	158	88.2		
FICON CTC	44.1	129.1	244.9		

6.4.11 z/OS CPU utilization

Finally, we include overall z/OS CPU utilization and the z/OS capture ratio. The CPU utilization numbers for each system are included in Table 6-86. The dropoff in the number of transactions being processed has begun to be reflected in reduced z/OS CPU utilization.

Table 6-86 z/OS CPU utilization

	0 km	20 km	40 km	60 km	80 km
#@$2	45.2%	49.1%	46.7%		
#@$3	46.5%	48.7%	47.7%		

To determine whether increased distance has any effect on capture ratio, we also included the capture ratios for each distance in Table 6-87.

Table 6-87 z/OS capture ratios

	0 km	20 km	40 km	60 km	80 km
#@$2	90%	90%	90%		
#@$3	91.2%	90.2%	90.1%		

6.5 Measurement at 60 km

For the next set of measurements, at 60 km, the configuration was unchanged from that used for the base measurement, with the exception that a pair of DWDMs with 60 km of fiber between them was added to the configuration.

6.5.1 Transaction response times

Table 6-88, Table 6-89, and Table 6-90 show the transaction rate, average response time, and goal achievement, for all CICS transactions and for just the TPNO transaction, respectively, across both systems in the data sharing group. With the increased distance and increased response times, we are now no longer meeting our WLM objective of 90% of all transactions ending in half a second or less.

Table 6-88 Information for all CICS transactions - sysplex level

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	278	269	249	209	
Resp time (milliseconds)	17	42	104	249	
Goal achievement %	100	100	96.6	79	

Table 6-89 Information for TPOS transactions - sysplex level

TPOS	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	9.8	9.5	8.9	7.6	
Resp time (milliseconds)	4.0	5.0	7.0	6.5	
Goal achievement %	100	100	100	100	

Table 6-90 Information for TPNO transactions - sysplex level

TPNO	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	97	94	87	73	
Resp time (milliseconds)	41	100	251	580	
Goal achievement %	100	100	91.8	45.8	

Table 6-91 and Table 6-92 show the response times across all transactions for each of the #@$2 and #@$3 systems, respectively.

Table 6-91 Average response times across all CICS transactions on system #@$2 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139	135	127	106	
Resp time (milliseconds)	16	34	88	230	
Goal achievement %	100	100	97.5	81.8	

Table 6-92 Average response times across all CICS transactions on system #@$3 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139	133	122	103	
Resp time (milliseconds)	17	49	120	268	
Goal achievement %	100	100	95.7	76.2	

Notice that the number of transactions completed on each system is still fairly level, but system #@$3 continues to have longer response times than #@$2, and the difference in response times across the two systems continues to increase.

Table 6-93 and Table 6-94 show the transaction rates and response times for the TPOS transactions. Interestingly, the response time on system #@$2 has not increased any more when going from 40 km to 60 km. The response time on system #@$3 *did* increase, but only by a very small amount. What is very interesting is that even though TPOS uses some of the same DB2 tables as the TPNO transaction, TPOS (so far) is unaffected by contention, but the TPNO response time is dominated by contention.

Also, TPOS and TPNO are turning out to be excellent examples of how two different transactions, running on the same hardware and software, using the same data, and running over the same distance, can be impacted by the distance in very different ways. This confirms

what we said previously, that is it simply not possible to take a transaction response time and apply some simple formula, and end up with an accurate projection of response times at various distances.

Table 6-93 Average AOR response times for TPOS transaction on system #@$2 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9	4.8	4.5	3.9	
Resp time (milliseconds)	4.0	4.0	6.0	4.0	
Goal achievement %	100	100	100	100	

Table 6-94 Average AOR response times for TPOS transaction on system #@$3 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9	4.7	4.4	3.7	
Resp time (milliseconds)	4.0	6.0	8.0	9.0	
Goal achievement %	100	100	100	100	

The number of TPNO transactions executed on each system and the response times are shown in Table 6-95 and Table 6-96 on page 143.

Table 6-95 Average AOR response times for TPNO transaction on system #@$2 (in milliseconds)

TPNO transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49	47	44	37	
Resp time (milliseconds)	40	83	212	532	
Goal achievement %	100	100	94.3	52.6	

Table 6-96 Average AOR response times for TPNO transaction on system #@$3 (in milliseconds)

TPNO transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49	47	43	36	
Resp time (milliseconds)	42	117	289	628	
Goal achievement %	100	100	89.4	39	

The TPNO response times continue to increase sharply. The increase is not exponential, but it is more than linear. Also, even though system #@$2 is less impacted by the distance than system #@$3, it *is* tracking with system #@$3 and increasing more than we would have expected. To understand this, we need to look at the components of the TPNO response time on each system.

To make it easier to see the changes in response time as the distance is changed, we provide the preceding information in a graphic format in Figure 6-18.

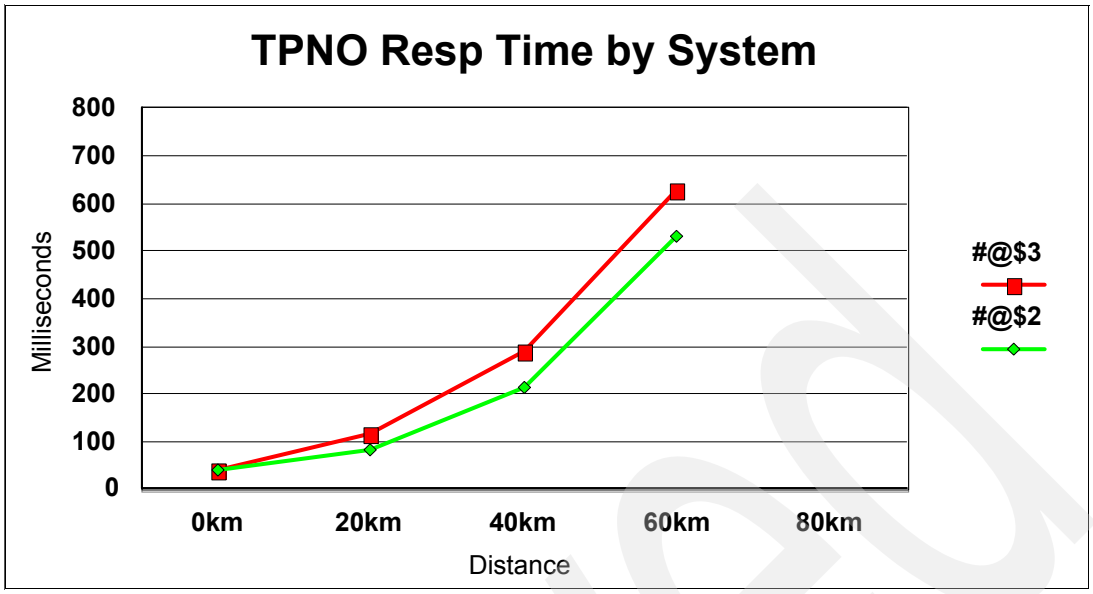


Figure 6-18 TPNO response time by system for each distance

To verify the numbers we see in DB2PE and RMF, we also ran a pair of CICS PA reports for the TPNO transaction for each system. One report shows the response time from the perspective of the TORs. The other report shows the response time (and response time breakdown) from the perspective of the AOR. As you can see in Table 6-97 and Table 6-98 on page 144, the response time as seen by the AOR continues to closely match the transaction response time as seen by both DB2 and WLM.

Table 6-97 CICSPA response times (in milliseconds) for TPNO transaction for #@$2

	0 km	20 km	40 km	60 km	80 km
#@$2 TOR	41.7	86.5	215.3	540	
#@$2 AOR	41.2	85.2	214.6	533.2	

Table 6-98 CICSPA response times (in milliseconds) for TPNO transaction for #@$3

	0 km	20 km	40 km	60 km	80 km
#@$3 TOR	44.1	118.3	287.8	630.7	
#@$3 AORs	43.4	118.5	290.7	629.7	

TPNO response time breakout

Next, we look in more detail at the components of the TPNO response time on each system. Table 6-99 and Figure 6-19 on page 145 show the response time components on system #@$2.

Table 6-99 Average DB2 times for TPNO transaction on system #@$2 *(in milliseconds)*

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.4	31.7	124	414.3	
Synchronous I/O delays	2.2	2.5	3.1	3.4	
Global contention	7.0	22.4	47.6	62.5	

Suspend time component	0 km	20 km	40 km	60 km	80 km
Asynchronous CF Requests	2.9	3.1	3.9	4.5	
Update Commit time	7.1	12.1	17.4	23.8	
Other miscellaneous DB2 suspends	0.1	0.3	0.9	3.1	
Other Not Account by DB2	0.4	0.4	0.6	1.0	
CPU time spent in DB2	4.6	4.8	4.9	5.0	
Non-DB2 CPU time	1.1	1.2	1.2	1.2	
Total as seen by DB2	38.4	83.2	207.7	531.3	

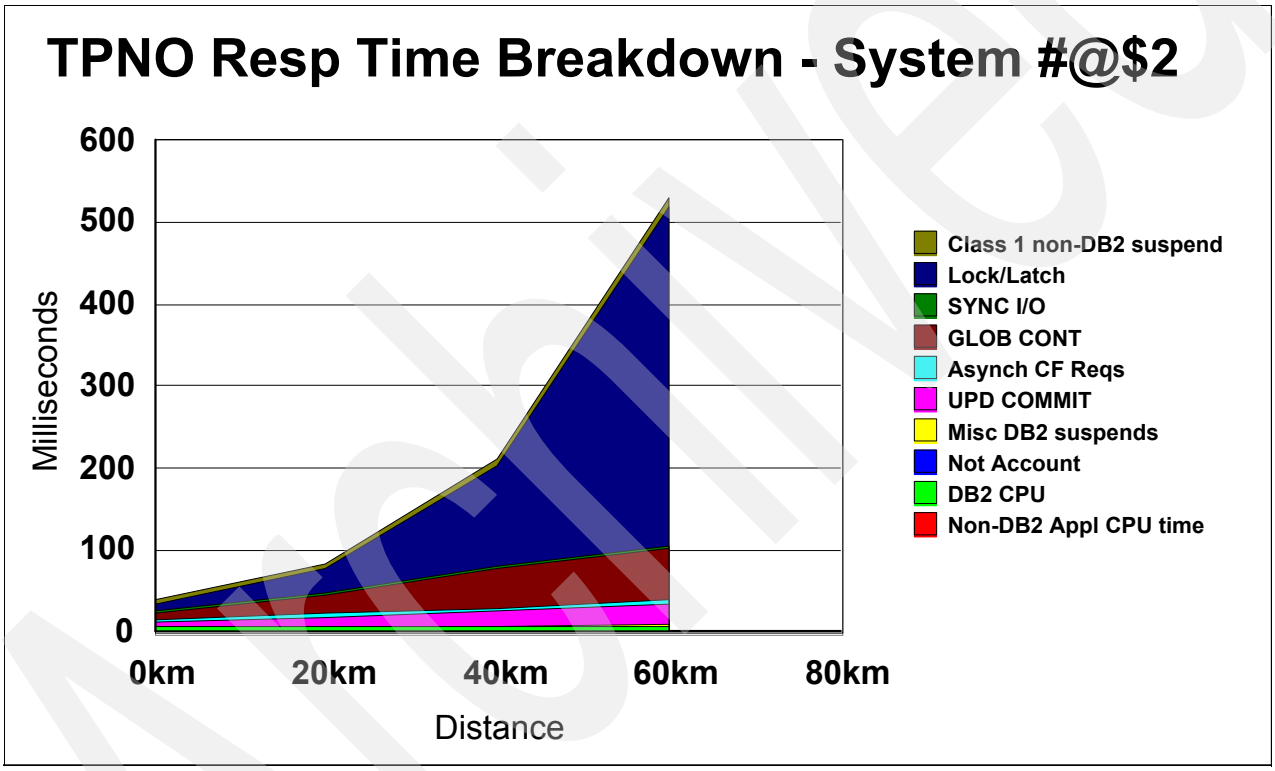


Figure 6-19 TPNO response time breakdown for system #@$2

The corresponding information for system #@$3 is shown in Table 6-100 and Figure 6-20 on page 146.

Table 6-100 Average DB2 times for TPNO transaction on system #@$3 *(in milliseconds)*

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.8	28.5	132.5	415.3	
Synchronous I/O delays	2.2	3.3	4.4	5.6	
Global contention	9.6	53.0	106	150.2	
Asynchronous CF Requests	2.6	7.2	11.7	15.2	
Update Commit time	6.8	11.7	16.6	21.2	

Suspend time component	0 km	20 km	40 km	60 km	80 km
Other miscellaneous DB2 suspends	0.1	0.4	1.3	2.5	
Other Not Account by DB2	0.6	2.2	4.3	6.1	
CPU time spent in DB2	4.5	4.6	4.9	5.1	
Non-DB2 CPU time	1.1	1.2	1.2	1.2	
Total as seen by DB2	40.5	116.3	288.3	627.9	

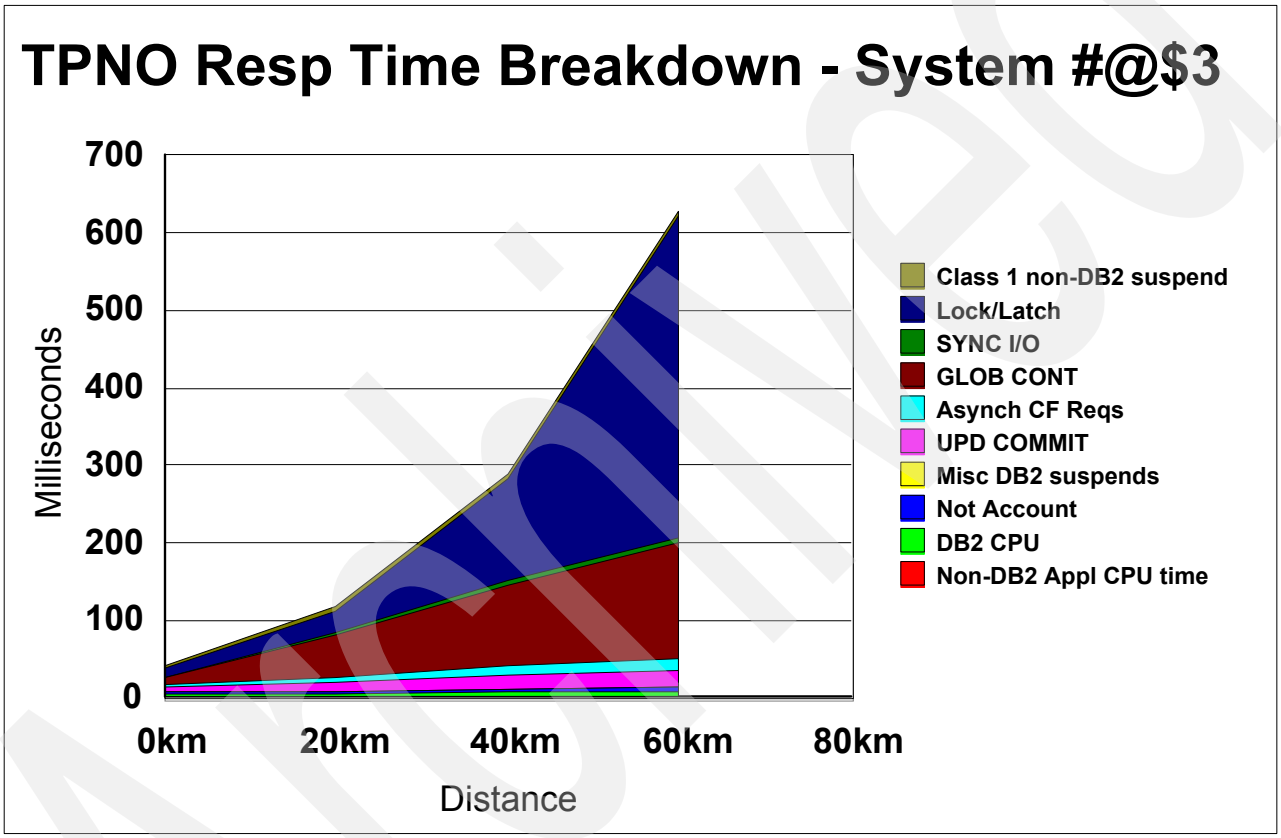


Figure 6-20 TPNO response time breakdown for system #@$3

At this point the two contention-related suspense reasons are really dominating the response time on both systems, with contention delays accounting for nearly 90% of the response time on both systems. In relative terms, all the other components are nearly irrelevant.

6.5.2 Lock/Latch

Table 6-101 shows the Lock/Latch suspend time for both systems.

Table 6-101 Average Lock/Latch suspension time and events for TPNO transaction

Lock/Latch suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	9.4	31.7	124	414.3	
No. of suspension events - #@$2	1.2	1.9	3	4.3	

Lock/Latch suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$3	9.8	28.5	132.5	415.3	
No. of suspension events - #@$3	1.3	2.2	4.3	5.8	

As you can see, the average Lock/Latch time for system #@$2 has increased dramatically, to nearly three and a half times what it was at 40 km. System #@$3 has seen a similar increase. The number of Lock/Latch contention events has also increased on both systems. However, this is to be expected because with every transaction taking so long to complete, the chance of looking for a resource that is already serialized by another transaction is greatly increased.

While it is now obvious that we have a major contention problem, no changes will be made to address this until the full set of measurements have been taken.

6.5.3 Synchronous I/O

The next part of the TPNO response time is time spent waiting for synchronous I/O (database reads and log writes) to complete.

Table 6-102 Average Sync I/O suspend time and events for TPNO transaction

Synchronous I/O suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	2.2	2.5	3.1	3.4	
No. of suspension events - #@$2	2.4	2.3	2.5	2.5	
Total suspension time - #@$3	2.2	3.3	4.4	5.6	
No. of suspension events - #@$3	2.5	2.4	2.6	2.7	

The suspend times and the number of events on both systems are shown in Table 6-102. These values have been very consistent across all the measurements so far, and remain so in this 60 km measurement. The suspend time values conform very closely to the modelling for that distance, and the number of events is basically unchanged across all the distances.

6.5.4 Global contention

The global contention times for both systems are shown in Table 6-103.

Table 6-103 Average global contention and events for TPNO transaction

Global contention	0 km	20 km	40 km	60 km	80 km
Reported global contention time - #@$2	7.0	22.4	47.6	62.5	
Reported number of global contention events - #@$2	7.5	11.4	14.5	19.9	
Actual no. of global contention events - #@$2	0.35	0.66	.7	.6	
Reported global contention time - #@$3	9.6	53	106	150.2	
Reported number of global contention events - #@$3	32.6	23.5	45.7	45	

Global contention	0 km	20 km	40 km	60 km	80 km
Actual no. of global contention events - #@$3	0.35	0.74	.9	.7	

The global contention suspend time has continued to increase on both systems, with both systems seeing roughly a 50% increase in suspend time when we moved from 40 to 60 km. The actual number of global contention events on both systems has actually dropped back a little compared to the 40 km measurement. We believe that this is more a reflection of the reduced number of transactions that are being processed than an indicator of any performance improvement.

On system #@$2, there were 19.3 asynchronous lock requests included in the global contention time, at 63.6 microseconds each (as shown in Table 6-104 on page 148). This sums to just a little over 1 ms, meaning that the suspend time for an actual global contention event is averaging about 61.5 microseconds per transaction. Allowing for the fact that there is still less than one global contention suspend per transaction (.6), the suspend time per global contention event on system #@$2 has now risen to 102.5 ms.

Table 6-104 RMF structure activity report for DB8QU_LOCK1 structure

System	Synch Rate	Synch Serv Time (mics)	Asynch Rate	Asynch Serv Time (mics)
#@$2	3301.7	21.5	885	63.6
#@$3	20	662	3990	662

On system #@$3, there were 44.3 (45 - .7) asynchronous lock requests reported as global contention events, each taking 662 microseconds (as shown in Table 6-104). This totals to 29.3 ms. Removing these from the total reported global contention time leaves us with 120.9 ms of global contention delay, on average, per TPNO transaction. Adjusting for the fact that there were only .7 global contention events per transaction gives us an actual delay time of 172 ms per global contention event on system #@$3.

While a lot smaller than the Lock/Latch time, this is still two to three times the *total* transaction response time at 0 km. Short of tuning the application, there is really very little that can be done to make the contention time “go faster”. And we mentioned at the beginning of this chapter that we would not do any tuning until the measurements of all distances had been completed. Although we hope that we can tune away some of the contention, we are still unable to predict how the global contention time will change between this measurement and the 80 km measurement.

6.5.5 Asynch CF Requests

The Asynch CF Requests value for systems #@$2 and #@$3 are shown in Table 6-105.

Table 6-105 Average Asynch CF Request suspension time and events for TPNO transaction

Asynch CF Requests	0 km	20 km	40 km	60 km	80 km
Total Asynch CF Request suspension time - #@$2	2.9	3.10	3.9	4.5	
Number of events - #@$2	20.7	20.5	20.4	20	
Total Asynch CF Request suspension time - #@$3	2.6	7.2	11.7	15.2	

Asynch CF Requests	0 km	20 km	40 km	60 km	80 km
Number of events - #@$3	20.8	21.3	21.4	20	

You may notice that we are moving towards two types of delays: those that are relatively small, linear, and reasonably predictable, and those that are growing very quickly and are impossible to predict or model. If you look at the Asynch CF Requests Suspend times for both systems, you will notice that the pattern is very similar to that observed for the Synchronous I/O delays.

You might also notice another very interesting comparison. The 0 km suspend times for both Synchronous I/Os and Asynch CF Requests are similar. However, we have about eight times as many Asynch CF Requests as we have Synchronous I/O events. This reflects the much shorter response times for the CF requests.

However, as the distance increases, the Asynch CF Requests from system #@$3 are impacted much more than the Synchronous I/Os from the same system; on system #@$3, the Synchronous I/O suspend time has increased to 5.6 ms, but the Asynch CF Request suspend time has increased to 15.1 ms. This reinforces what was stated earlier in this book, that the *relative* impact of distance on CF requests is worse than the impact on I/Os.

Returning to the numbers, system #@$2 is doing about 20 Asynchronous CF Requests, most of which are reads. We would expect the response time for those read requests to be about 153 microseconds (based on the 0 km numbers for system #@$2 in Table 6-106). The response time for the write (we believe there is about 1 write on average) is about 1284 microseconds (the longer of the writes to the primary and secondary GBPs). If we multiply these out, we get about 4 ms, which is very close to the actual Asynch CF Request Suspend time for system #@$2.

Table 6-106 Response time of GBP3 CF requests

System/GBP	0 km	20 km	40 km	60 km	80 km
#@$2 to Primary GBP3	153	221	275	341	
#@$2 to Secondary GBP3	279	621	987	1284	
#@$3 to Primary GBP3	125	321	525	733	
#@$3 to Secondary GBP3	220	405	525	603	

System #@$3 also had about 20 events. However, that system's read response time will be significantly higher, at about 700 microseconds. The response time for the writes will be slightly longer, but by not a huge amount (when a cross-invalidate is required, the target system (system #@$2) is right beside the CF, so the additional time required is minimal). If we multiply these numbers out, we get about 15 ms, which is nearly exactly the amount being reported by DB2PE.

Based on these calculations, the Asynch CF Requests wait time is growing in a linear, and predictable, manner. There is little that you can do to improve the response time for these requests, especially for the system that is remote to the CF. The bulk of the response time is the time it takes for the signals to travel up and down the fiber. Increasing the speed of the processor at either end will only have a minimal impact, and only one type of CF link (ISC links) currently supports distances greater than 150 meters, so you cannot do anything to make the link faster either.

6.5.6 Update Commit time

Table 6-107 shows the Update Commit time for both systems. You will recall that Update Commit consists of log writes, GBP writes, releasing locks, and some general DB2 transaction cleanup.

Table 6-107 Average Update Commit suspend time for TPNO transaction (in milliseconds)

Update Commit Suspensions	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	7.1	12.1	17.4	23.6	
No. of suspension events - #@$2	1	1	1	1	
Total suspension time - #@$3	6.8	11.7	16.6	21.2	
No. of suspension events - #@$3	1	1	1	1	

If we look at system #@$2 first, its Update Commit time increased by 5 ms when going from 0 to 20 km, a little more than 5 ms when going from 20 km to 40 km, and a little over 6 ms when going from 40 km to 60 km. Given that both the log writes and the GBP writes (to both the primary and secondary GBPs) are impacted by the increased distance, the increases are to be expected, and they are nearly linear.

On system #@$3, the Update Commit time is again increasing in a linear manner, in this case by about 5 ms for every additional 20 km.

Log writes

Because system #@$2 is located in the same site as the primary DASD, there would be no change in the time required to send the I/O to the primary DASD. However, because these are writes, they will be impacted by the increasing PPRC distance to the secondary DASD. At 60 km, the remote copy impact should be about .6 ms. Adding that to the .6 ms we experienced at 0 km gives us 1.2 ms, which is exactly what DB2 is actually experiencing, as shown in Figure 6-21.

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
2363	7	2247	23856	260.8	1.2	0.2	0.9	0.1	0.0

Figure 6-21 Workload activity report for D8Q1 MSTR address space

System #@$3 would also experience the .6 ms remote copy cost. But in addition, there will be a cost of another .6 ms to send the data from #@$3 to the primary DASD (60 km away). This sums to 1.2 ms, in addition to the .6 ms that DB2 was getting at 0 km. As you can see in Figure 6-22, 1.8 ms is in fact exactly what DB2 on system #@$3 is achieving.

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
1154	6	1072	13526	159.1	1.8	0.2	0.9	0.7	0.0

Figure 6-22 Workload activity report for D8Q2 MSTR address space

Based on these measurements and projections, we can see that this part of the Update Commit time is capable of being modeled, and is growing in a nice, linear way.

GBP writes

The second component of Update Commit time is the time required to write the updated pages to the primary and secondary GBPs. Although DB2 does not explicitly report the number of GBP requests that were issued during commit processing, there is a way to arrive at the approximate number. But the process is quite complex, so instead of repeating it over and over for each distance, we will hold off until we discuss the results of the last measurement, at 80 km. For now, we will simply work on the basis that there were about 14 GBP writes on system #@$2 and 11 on system #@$3 during commit processing.

On system #@$2, the total update commit time was 23.6 ms. Using the logic that was introduced in "GBP writes" on page 79, we multiply the number of writes by the response time that system #@$2 was observing for write requests to the secondary GBP (1284 microseconds as shown in Table 6-78 on page 138), giving us about 18 ms spent writing to the GBPs during commit processing.

On system #@$3, the total Update Commit time was 21.2 ms. Using the same logic for determining response times, we multiply 11 writes by the response time system #@$2 was observing for the secondary structure. This gives us about 14 ms spent writing to the GBPs.

Table 6-108 Response time of GBP3 CF requests

System/GBP	0 km	20 km	40 km	60 km	80 km
#@$2 to Primary GBP3	153	221	275	341	
#@$2 to Secondary GBP3	279	621	987	1284	
#@$3 to Primary GBP3	125	321	525	733	
#@$3 to Secondary GBP3	220	405	525	603	

Release locks

The next part of Update Commit processing is releasing the locks help by the thread. Although we do not know exactly how long this process takes (unless we run a trace), we do not believe it is as significant as writing to the GBPs and log data sets, so we will not focus on this processing.

Cleanup

The final part of Update Commit is the DB2 cleanup processing. Compared to the other three parts of Update Commit, this should complete very quickly. We measured this processing (using a trace) at about 2 ms.

Summing the components of Update Commit, we get 3 ms (log writes) plus 18 ms (GBP writes) plus 2 ms (Cleanup), giving us about 23 ms, which is very close to the DB2-reported numbers of 23.6 ms and 21.2 ms.

6.5.7 Not Account

Table 6-109 shows the Not Account time for the two systems.

Table 6-109 Average Not Accounted (in milliseconds) for TPNO transaction

Not Account time	0 km	20 km	40 km	60 km	80 km
Not Accounted - #@$2	.4	.4	.6	1.0	
Not Accounted - #@$3	.6	2.2	4.3	6.1	

The Not Accounted time for system #@$2 continues to increase slowly, moving up from .6 ms at 40 km to 1.0 ms at 60 km. On system #@$3, with its higher response times, the Not Accounted time is a larger value, increasing from 4.3 ms at 40 km up to 6.0 ms at 60 km.

While it would be nice to be able to understand where the additional time is going, in reality, Not Account is only a little over 1% of the total TPNO response time, so putting significant resource into identifying the reasons for the increase would not be a very productive use of time.

6.5.8 DB2 CPU time

Table 6-110 on page 152 contains the DB2 CPU time for TPNO. We can see that the average DB2 CPU time for system #@$2 has increased marginally from 4.6 ms at 0 km up to 5 ms at 60 km. For system #@$3, DB2 CPU time stays has also increased by a small amount, to 5.1 ms. Keep in mind that this is only part of the CPU time spent by DB2.

Table 6-110 Average DB2 CPU time (in milliseconds) for TPNO transaction

DB2 CPU time	0 km	20 km	40 km	60 km	80 km
DB2 CPU time - #@$2	4.6	4.8	4.9	5.0	
DB2 CPU time - #@$3	4.5	4.6	4.9	5.1	

6.5.9 Overall Coupling Facility activity

In addition to the performance of the specific DB2 structures, it is also important to monitor the overall health, performance, and capacity of the Coupling Facility. The metrics that we specifically monitor in the performance runs are CF CPU utilization and subchannel utilization.

The CPU utilization of each CF, as shown in Table 6-111, can be found in the RMF Coupling Facility report. As the transaction rate continues to decrease, the utilization of both CFs is now dropping.

Table 6-111 Coupling Facility CPU utilization

	0 km	20 km	40 km	60 km	80 km
FACIL05	14.3%	14.1%	13%	11.1%	
FACIL06	4.9%	5.7%	5.9%	5.0%	

The CF subchannel utilization numbers shown in Table 6-112 and Table 6-113 can be calculated from the values provided in the SUBCHANNEL ACTIVITY report, which is provided after the detail structure activity section of the RMF CF Activity report. Once again you can see that even for the CFs that have lower utilization, the increased response time more than offsets this, resulting in increases in subchannel utilization for both CFs from both systems. Additionally, the #@$3 subchannels for communicating with CF FACIL05 are now uncomfortably close to the IBM-recommended threshold of 30%. If this were a production environment, additional CF links should be installed between system #@$2 and CF FACIL06 at this point.

Table 6-112 FACIL05 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	3.5%	4.9%	5.7%	5.7%	

	0 km	20 km	40 km	60 km	80 km
From #@$3	3.4%	13.8%	21.7%	27.4%	

Table 6-113 FACIL06 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	1.2%	3.3%	5.8%	6.9%	
From #@$3	1%	1.9%	2.6%	2.9%	

6.5.10 XCF signaling

Table 6-114 on page 153 and Table 6-115 on page 153 contain the XCF message rates for both systems.

Table 6-114 XCF messages sent per second from #@$2 to #@$3

	0 km	20 km	40 km	60 km	80 km
CF structures	144.2	277.0	321.2	243.1	
FICON CTC	24.3	0.3	1.2	1.3	

Table 6-115 XCF messages sent per second from #@$3 to #@$2

	0 km	20 km	40 km	60 km	80 km
CF structures	126.5	158	88.2	67.4	
FICON CTC	44.1	129.1	244.9	180.9	

You will recall that increased contention generally results in increased XCF messaging. However, there is also a point where the decreasing number of transactions is reflected in a declining number of XCF messages. Figure 6-23 shows that even though the number of contention events per transaction continues to increase, the XCF message rate has started to drop off because fewer transactions are being processed (209 per second at 60 km compared to 278 per second at 0 km).

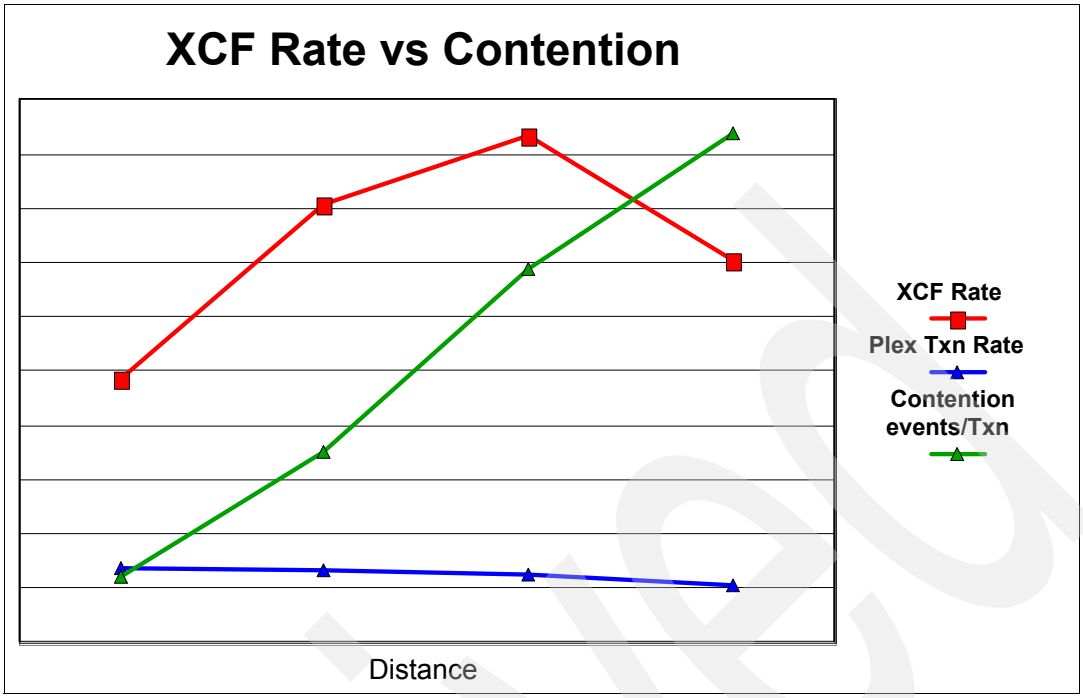


Figure 6-23 XCF message rate versus contention

You will also notice that the proportion of XCF messages going from system #@$3 to #@2 over the CTCs compared to the XCF signalling structure in the CF appears to have stabilized. As we moved from 0 km to 40 km, the percent of messages going over the CTC was increasing. However, going from 40 km to 60 km did not change the balance; roughly 75% of the messages are being sent over the CTCs, and 25% over the signalling structures. Of course, when going from system #@$2 to #@$3, nearly all the messages continue to be sent via the signalling structures because of how we had the CTC control units set up.

6.5.11 z/OS CPU utilization

Finally, we include overall z/OS CPU utilization and the z/OS capture ratio. The CPU utilization numbers for each system are included in Table 6-116. You can see that the utilization on both systems continues the decline that started at 40 km.

Table 6-116 z/OS CPU utilization

	0 km	20 km	40 km	60 km	80 km
#@$2	45.2%	49.1%	46.7%	40.3%	
#@$3	46.5%	48.7%	47.7%	43.7%	

To determine whether increased distance has any effect on capture ratio, we also included the capture ratios for each distance in Table 6-117. The capture ratio on system #@$2 (the one beside the primary DASD and most of the CF structures) remains basically unchanged from 0 km through 60 km. On system #@$3, there continues to be a small decline in the capture ratio as the distance increases.

Table 6-117 z/OS capture ratios

	0 km	20 km	40 km	60 km	80 km
#@$2	90%	90%	90%	90.2%	
#@$3	91.2%	90.2%	90.1%	89.8%	

6.6 Measurement at 80 km

For the next set of measurements, at 80 km, the configuration was unchanged from that used for the base measurement, with the exception that a pair of DWDMs with 80 km of fiber between them was added to the configuration.

6.6.1 Transaction response times

Table 6-118, Table 6-119, and Table 6-120 on page 155 show the transaction rate, average response time, and goal achievement, for all CICS transactions, for the TPOS transaction, and for the TPNO transaction, respectively, across both systems in the data sharing group.

Table 6-118 Information for all CICS transactions - sysplex level

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	278	269	249	209	178
Resp time (milliseconds)	17	42	104	249	418
Goal achievement %	100	100	96.6	79	70.9

Table 6-119 Information for TPOS transactions - sysplex level

TPOS	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	9.8	9.5	8.9	7.6	6.5
Resp time (milliseconds)	4.0	5.0	7.0	6.5	12.0
Goal achievement %	100	100	100	100	100

Table 6-120 Information for TPNO transactions - sysplex level

TPNO	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	97	94	87	73	62
Resp time (milliseconds)	41	100	251	580	842
Goal achievement %	100	100	91.8	45.8	24.1

You can see that, across all CICS transactions, only 70.9% are now meeting the WLM goal of .5 second response time. And of the TPNO transactions, only 24% are achieving the goal.

Table 6-121 and Table 6-122 show the response times across all transactions for each of the #@$2 and #@$3 systems, respectively.

Table 6-121 Average response times across all CICS transactions on system #@$2 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139	135	127	106	89
Resp time (milliseconds)	16	34	88	230	414
Goal achievement %	100	100	97.5	81.8	73.7

Table 6-122 Average response times across all CICS transactions on system #@$3 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	139	133	122	103	89
Resp time (milliseconds)	17	49	120	268	423
Goal achievement %	100	100	95.7	76.2	68.2

Notice that the number of transactions completed continues to drop. But, surprisingly, the disparity between the average response times on system #@$2 and system #@$3 appears to have stopped growing.

Table 6-123 and Table 6-124 on page 156 show the response time and transaction rates for the TPOS transaction.

Table 6-123 Average AOR response times for TPOS transaction on system #@$2 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9	4.8	4.5	3.9	3.3
Resp time (milliseconds)	4.0	4.0	6.0	4.0	11.0
Goal achievement %	100	100	100	100	100

Table 6-124 Average AOR response times for TPOS transaction on system #@$3 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	4.9	4.7	4.4	3.7	3.2
Resp time (milliseconds)	4.0	6.0	8.0	9.0	13.0
Goal achievement %	100	100	100	100	100

What is somewhat surprising (and very interesting) is the trebling of the response time of the TPOS transaction on system #@$2. Up until this run, there had been little change in the response time of TPOS on that system. A little investigation determined that the high average response time was caused by a very small number of TPOS transactions that hit contention with TPNO, resulting in *much* higher than normal response times for these transactions. Further investigation determined that the TPOS average response time on system #@$3 was the result of the increased distance, plus one very long-running transaction.

Because there are so few TPOS transactions and they have such short response times, a small number of long-running transaction can easily skew the average. What makes this

particularly interesting is that until this run, this transaction did not encounter any contention. However, because the TPNO transactions are now running for so long, and holding onto their resources for so long, there is an increased likelihood that a TPOS transaction will need some resource that is already serialized by TPNO. This shows that even transactions that normally do not suffer from contention problems can start to be affected by other transactions that have very long elapsed times.

The measurement results for the TPNO transactions executed on each system are shown in Table 6-125 and Table 6-126.

Table 6-125 Average AOR response times for TPNO transaction on system #@$2 (in milliseconds)

TPNO transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49	47	44	37	31
Resp time (milliseconds)	40	83	212	532	787
Goal achievement %	100	100	94.3	52.6	29.8

Table 6-126 Average AOR response times for TPNO transaction on system #@$3 (in milliseconds)

TPNO transactions	0 km (base)	20 km	40 km	60 km	80 km
Txn rate (per second)	49	47	43	36	31
Resp time (milliseconds)	42	117	289	628	897
Goal achievement %	100	100	89.4	39	18.4

Although the rate of increase of the TPNO transaction response time has slowed with the latest measurement, the response times are likely to be unacceptable, having increased from about 40 ms at 0 km, up to between 787 and 897 ms at 80 km. To make it easier to see the changes in response time as the distance is changed, we provide the preceding information in a graphic format in Figure 6-24 on page 157.

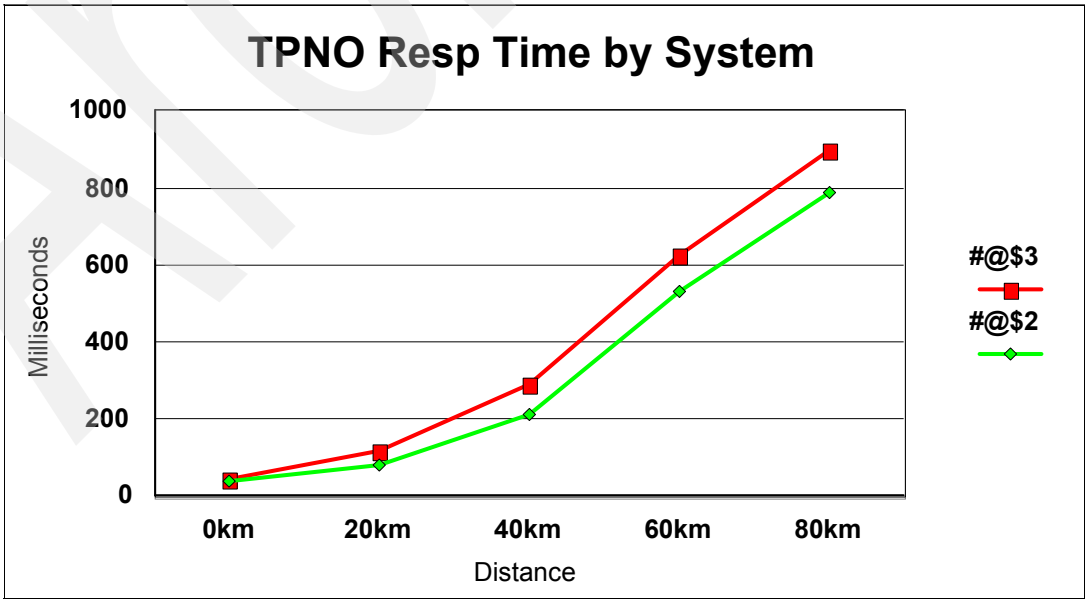


Figure 6-24 TPNO Response time by system for each distance

To verify the numbers we see in DB2PE and RMF, we also ran a pair of CICS PA reports for the TPNO transaction for each system. One report shows the response time from the perspective of the TORs. The other report shows the response time (and response time breakdown) from the perspective of the AOR. As you can see in Table 6-127 and Table 6-128, the response time as seen by the AOR continues to closely match the transaction response time as seen by both DB2 and WLM.

Table 6-127 CICSPA response times (in milliseconds) for TPNO transaction for #@$2

	0 km	20 km	40 km	60 km	80 km
#@$2 TOR	41.7	86.5	215.3	540	795
#@$2 AOR	41.2	85.2	214.6	533.2	789.5

Table 6-128 CICSPA response times (in milliseconds) for TPNO transaction for #@$3

	0 km	20 km	40 km	60 km	80 km
#@$3 TOR	44.1	118.3	287.8	630.7	901.6
#@$3 AORs	43.4	118.5	290.7	629.7	900.3

TPNO response time breakout

Next, we look in more detail at the components of the TPNO response time on each system. Table 6-129 and Figure 6-25 on page 159 show the response components on system #@$2.

Table 6-129 Average DB2 times for TPNO transaction on system #@$2 (in milliseconds)

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.4	31.7	124	414.3	657.1
Synchronous I/O delays	2.2	2.5	3.1	3.4	3.5
Global contention	7.0	22.4	47.6	62.5	67.3
Asynchronous CF Requests	2.9	3.1	3.9	4.5	4.8
Update Commit time	7.1	12.1	17.4	23.8	29.2
Other miscellaneous DB2 suspends	0.1	0.3	0.9	3.1	3.9
Other Not Account by DB2	0.4	0.4	0.6	1.0	.9
CPU time spent in DB2	4.6	4.8	4.9	5.0	5.0
Non-DB2 CPU time	1.1	1.2	1.2	1.2	1.2
Total as seen by DB2	38.4	83.2	207.7	531.3	786.5

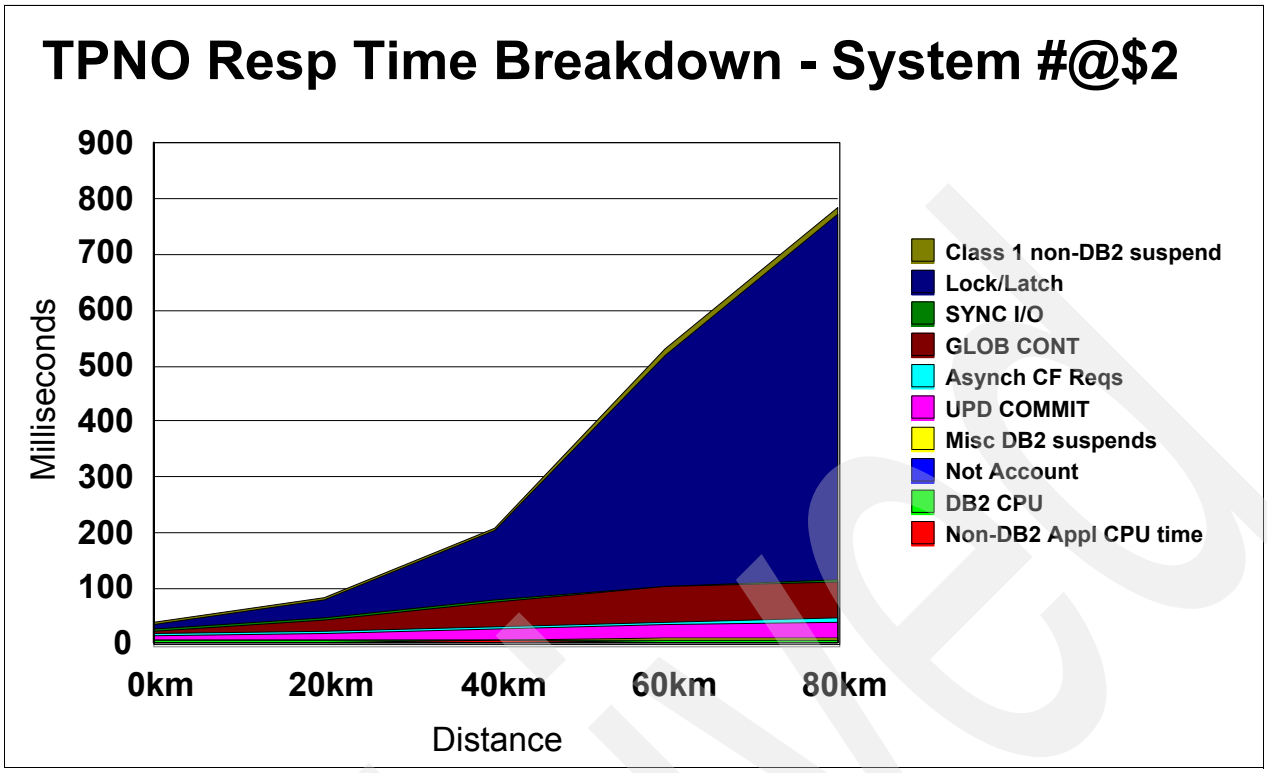


Figure 6-25 TPNO response time breakdown for system #@$2

Table 6-130 and Figure 6-26 on page 160 show the corresponding information for system #@$3.

Table 6-130 Average DB2 times for TPNO transaction on system #@$3 *(in milliseconds)*

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.8	28.5	132.5	415.3	642.2
Synchronous I/O delays	2.2	3.3	4.4	5.6	5.8
Global contention	9.6	53.0	106	150.2	179.9
Asynchronous CF Requests	2.6	7.2	11.7	15.2	19.2
Update Commit time	6.8	11.7	16.6	21.2	25.9
Other miscellaneous DB2 suspends	0.1	0.4	1.3	2.5	3.3
Other Not Account by DB2	0.6	2.2	4.3	6.1	7.0
CPU time spent in DB2	4.5	4.6	4.9	5.1	5.3
Non-DB2 CPU time	1.1	1.2	1.2	1.2	1.2
Total as seen by DB2	40.5	116.3	288.3	627.9	895.9

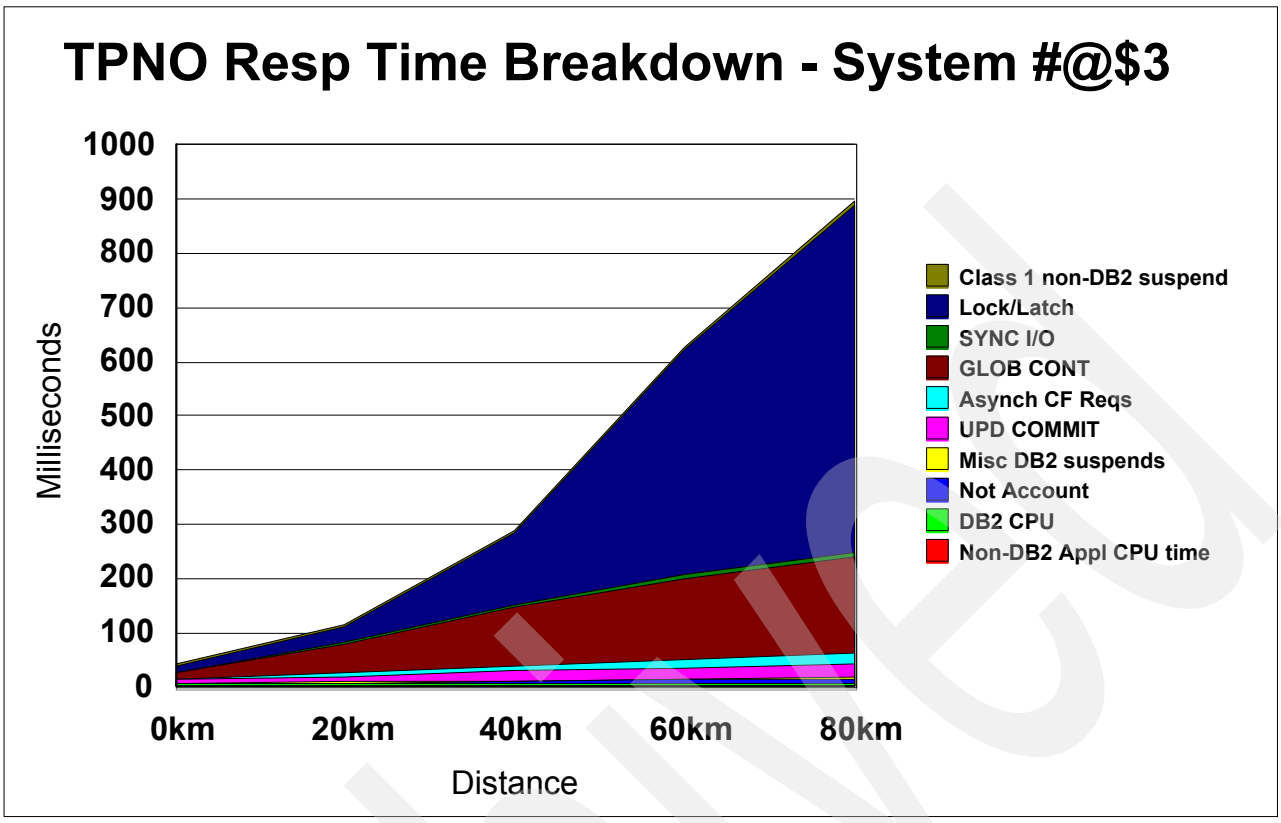


Figure 6-26 TPNO response time breakdown for system #@$3

As you can see in Figure 6-25 on page 159 and Figure 6-26, the response times continue to be dominated by the contention suspends, with contention still growing faster than any of the other response time components.

In this section, we briefly summarize the results of the 80 km measurement, pointing out anything that might be of interest. However, it has been obvious that we really have a problem with contention and unacceptable response times, so in the second part of this section we describe the process and reports that we used to understand exactly what is happening to these transactions. This chapter will then close with the presentation of the results of another measurement run after we made one change to try to reduce the contention.

6.6.2 Lock/Latch

The Lock/Latch delays for both systems are shown in Table 6-131. Lock/Latch is up about 50% on both systems compared to the 60 km run. The number of events is also up by a small amount.

Table 6-131 Average Lock/Latch suspension time and events for TPNO transaction

Lock/Latch suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	9.4	31.7	124	414.3	657.1
No. of suspension events - #@$2	1.2	1.9	3	4.3	4.7
Total suspension time - #@$3	9.8	28.5	132.5	415.3	642.2
No. of suspension events - #@$3	1.3	2.2	4.3	5.8	6.3

6.6.3 Synchronous I/O

The next part of the TPNO response time is time spent waiting for synchronous I/O to complete. The Synchronous I/O suspend times are shown in Table 6-132. Both systems displayed a small reduction in the number of events per transaction. The suspend time per event increased by about .3 ms on system #@$2, compared to the 60 km run, and by about .6 ms per event on system #@$3. Both of these increases are roughly what you would expect given the 20 km distance increase.

Table 6-132 Average Sync I/O suspend time and events for TPNO transaction

Synchronous I/O suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	2.2	2.5	3.1	3.4	3.5
No. of suspension events - #@$2	2.4	1.8	2.5	2.5	2
Total suspension time - #@$3	2.2	3.3	4.4	5.6	5.8
No. of suspension events - #@$3	2.5	1.9	2.6	2.7	2.1

6.6.4 Global contention

The Global contention times for both systems for the TPNO transaction are shown in Table 6-133. The rate of increase of global time has slowed greatly on both systems, and the actual number of global contention events per transaction has actually declined marginally. Nevertheless, the global contention on system #@$3 is more than four times longer than the total TPNO response time at 0 km and needs to be addressed if possible.

Table 6-133 Average global contention and events for TPNO transaction

Global contention	0 km	20 km	40 km	60 km	80 km
Reported global contention time - #@$2	7.0	22.4	47.6	62.5	67.3
Reported number of global contention events - #@$2	7.5	11.4	14.5	19.9	18.4
Actual no. of global contention events - #@$2	0.35	0.66	0.7	0.6	0.5
Reported global contention time - #@$3	9.6	53.0	106	150.2	179.9
Reported number of global contention events - #@$3	32.6	23.5	45.7	45	44.5
Actual no. of global contention events - #@$3	0.35	0.74	.9	.7	.5

6.6.5 Asynch CF Requests

The Asynch CF Requests time for both systems is shown in Table 6-134.

Table 6-134 Average Asynch CF Request suspension time and events for TPNO transaction

Asynch CF Requests	0 km	20 km	40 km	60 km	80 km
Total Asynch CF request suspension time - #@$2	2.9	3.1	3.9	4.5	4.8

Asynch CF Requests	0 km	20 km	40 km	60 km	80 km
Number of events - #@$2	20.7	20.5	20.4	20	20
Total Asynch CF request suspension time - #@$3	2.6	7.2	11.7	15.2	19.2
Number of events - #@$3	20.8	21.3	21.4	20	19.7

The Asynch CF Requests time on system #@$2 has been increasing at a rate of about .5 ms with every 20 km increase in the distance between the two sites. The move from 60 km to 80 km conformed to this pattern.

On system #@$3, the Asynch CF Requests time has been increasing by about 4 ms with every additional 20 km, and the move from 60 km to 80 km again conforms to this trend.

The number of Asynch CF Requests on both systems has remained fairly stable, with a minor decrease on system #@$3.

6.6.6 Update Commit

Table 6-135 shows the Update Commit time for both systems.

Table 6-135 Average Update Commit suspend time for TPNO transaction (in milliseconds)

Update Commit suspensions	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	7.1	12.1	17.4	23.8	29.2
No. of suspension events - #@$2	1	1	1	1	1
Total suspension time - #@$3	6.8	11.5	16.6	21.2	25.9
No. of suspension events - #@$2	1	1	1	1	1

The Update Commit time on system #@$2 has been growing by a little over 5 ms with each additional 20 km, and increasing the distance from 60 km to 80 km continues that trend. On system #@$3, the Update Commit time has been growing by between 4 ms and 5 ms per 20 km, and again, that trend is continued with the latest distance increase.

6.6.7 Not Account

Not Account is the bucket for delays that DB2 is not able to more accurately categorize. As you can see in Table 6-136, on system #@$2, the Not Account time was consistent for the first two measurements, increased a little at 40 km and again at 60 km, then stayed level at 80 km. On system #@$3, the increases have actually been more consistent. In fact, the Not Account time on system #@$3 has stayed at a fairly consistent percentage of the total response time, but s the Not Account time on system #@$2 has been steadily decreasing as a percentage of the total response time.

Table 6-136 Average Not Accounted (in milliseconds) for TPNO transaction

Not Account time	0 km	20 km	40 km	60 km	80 km
Not Accounted - #@$2	.4	.4	.6	1.0	.9
Not Accounted - #@$3	.6	2.2	4.3	6.1	7.0

Without intensive tracing it is not possible to identify what is happening on system #@$3 that is getting categorized as Not Account that is not happening on system #@$2. However, the only real difference between system #@$2 and #@$3 is the distance between the system and the primary DASD and most of the CF structures (both of which result in longer I/O and CF response times), and the fact that all transaction response times are longer on #@$3 than on #@$2. In every other way, the systems are identical.

6.6.8 DB2 CPU time

The CPU time charged against the TPNO transaction on each of the systems is shown in Table 6-137. System #@$3 continues its pattern of growing by about .2 seconds per 20 km. The growth on system #@$2 has been slower, and after the last distance increase it is basically unchanged from the 60 km measurement.

Table 6-137 Average DB2 CPU time (in milliseconds) for TPNO transaction

DB2 CPU time	0 km	20 km	40 km	60 km	80 km
DB2 CPU time - #@$2	4.6	4.8	4.9	5.0	5.0
DB2 CPU time - #@$3	4.5	4.6	4.9	5.1	5.3

6.6.9 Overall Coupling Facility activity

The CPU utilization of each CF can be found in the RMF Coupling Facility report and is shown in Table 6-138. In line with the reduced number of transactions, and the corresponding reduction in the number of XCF messages, the CPU utilization of both CFs has again decreased in this latest measurement.

Table 6-138 Coupling Facility CPU utilization

	0 km	20 km	40 km	60 km	80 km
FACIL05	14.3%	14.1%	13%	11.1%	9.3%
FACIL06	4.9%	5.7%	5.9%	5.0%	4.1%

Despite the constantly reducing CF load, the subchannel utilization for the system that is remote to FACIL05 (system #@$3) continues to increase, driven by the increasing response times for all requests from #@$3 to FACIL05 as shown in Table 6-139, and has now exceeded the IBM-recommended threshold.

On FACIL06, the subchannel utilization for the remote z/OS system (#@$2) is also increasing as shown in Table 6-140. However, because the number of requests being sent to FACIL06 is so low, the subchannel utilization remains well below the recommended threshold.

Table 6-139 FACIL05 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	3.5%	4.9%	5.7%	5.7%	5.5%
From #@$3	3.4%	13.8%	21.7%	27.4%	30.7%

Table 6-140 FACIL06 subchannel utilization

	0 km	20 km	40 km	60 km	80 km
From #@$2	1.2%	3.3%	5.8%	6.9%	7.2%

	0 km	20 km	40 km	60 km	80 km
From #@$3	1%	1.9%	2.6%	2.9%	2.9%

6.6.10 XCF signaling

Table 6-141 and Table 6-142 show the number of XCF messages being sent between the systems. The number of XCF messages started to decline when moving from 40 km to 60 km, and the decline continues with the latest increase to 80 km.

Table 6-141 XCF messages sent per second from #@$2 to #@$3

	0 km	20 km	40 km	60 km	80 km
CF structures	144.2	277.0	321.2	243.1	180.3
FICON CTC	24.3	0.3	1.2	1.3	1.4

As expected, just about all the XCF signals going from system #@$2 to #@$3 are sent over the CF. And for signals going from #@$3 to #@$2, roughly 3/4 continue to be sent over the CTCs.

Table 6-142 XCF messages sent per second from #@$3 to #@$2

	0 km	20 km	40 km	60 km	80 km
CF structures	126.5	158	88.2	67.4	53.5
FICON CTC	44.1	129.1	244.9	180.9	130.8

6.6.11 z/OS CPU utilization

Finally, we include overall the z/OS CPU utilization and the z/OS capture ratio. The CPU utilization numbers for each system are included in Table 6-27 on page 112. As you can see, the utilization of both systems has continued to drop as the number of transactions being executed has fallen. It will be interesting to see how the CPU utilization will change if we are able to reduce the contention.

Table 6-143 z/OS CPU utilization

	0 km	20 km	40 km	60 km	80 km
#@$2	45.2%	49.1%	46.7%	40.3%	34.5%
#@$3	46.5%	48.7%	47.7%	43.7%	38.9%

To determine whether increased distance has any effect on capture ratio, we also included the capture ratios for each distance in Table 6-28 on page 113. You can see that there is a very small downwards trend on system #@$3 as it moved further away. The ratio for system #$@2 was largely unchanged until the latest measurement, when it dropped by a little under 1%.

Table 6-144 z/OS capture ratios

	0 km	20 km	40 km	60 km	80 km
#@$2	90%	90%	90%	90.2%	89.3%
#@$3	91.2%	90.2%	90.1%	89.8%	90.1%

6.7 Addressing contention

It has been obvious since the 20 km measurement that the TPNO transaction has a contention problem. However, at that point, we had no idea that the contention would grow to be as much as nearly 92% of the total elapsed time of the transaction. It is also interesting that at 0 km there was really no indication that the transaction had a contention problem. This indicates that large distances can greatly magnify latent problems in the workload.

As the distance increased, the contention grew to the extent that all the other components of the response time were largely irrelevant. However, we had decided early on that we would not address any performance problems we discovered as the measurements progressed, because that would mean that we were no longer doing a full like-for-like comparison. Now that all measurements have been taken, we are able to attempt to see if we can improve matters.

6.7.1 Identifying the problem

From our work so far, we know that we want to focus on the TPNO transaction. And we know that the component of the TPNO response time that has increased the most is contention. So the next step is to look in more detail into the TPNO transaction to understand what it is doing, and where the contention is being encountered.

Referring to 4.1, “Workload description” on page 54, you can see that the TPNO transaction is reasonably complex, using a number of DB2 tables. In order to be able to proceed, the first thing we had to do was to attempt to identify which tables, or which SQL statements in the transaction, were suffering the most. To do this, we ran an SQL Activity Report using JCL similar to that shown in Example 6-1.

Example 6-1 Sample JCL to create SQL Activity Report

```
//PERSQLTR JOB CLASS=A,MSGCLASS=X,REGION=0M
//PMV110  EXEC PGM=DB2PM
//INPUTDD  DD  DISP=SHR,DSN=MARIO.ZSKO02.GTFTRACE
//SQLWORK  DD  UNIT=SYSDA,SPACE=(CYL,(300,300))
//DPMLOG   DD  SYSOUT=A
//SYSOUT   DD  SYSOUT=A
//JOBSUMDD DD SYSOUT=A
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *
GLOBAL
SQLACTIVITY
           REPORT
           SUMMARIZEBY(STMTNO)
           WORKLOAD (ALL)
           DDNAME(SQTRCDD1)
EXEC
```

The resulting SQL Activity Report is shown in Figure 6-27. The column headed AET/OCCUR reports on the elapsed time for each event. You can see that the CUR_DIST cursor is accounting for most of the time, taking 124 ms per occurrence. You need to check the source of the program to see what that cursor is being used for. In this case, it was used to process the DISTRICT table.

Note: For operational reasons, the reports shown in this chapter are based on a subsequent 20 km test, and not on the 80 km run documented in 6.6, “Measurement at 80 km” on page 155. Although the numbers may be different, all the concepts still apply.

```
PRIMAUTH: CICSUSER     PLANNAME: TPCCP2NO                              THREAD TOTAL:  9413  START AET: 0.003021  STOP AET: 0.000706

       EVENT            COUNT     TOT.ELAPS  TOTAL TCB                                  DETAIL
                                    AET/EVENT  TCB/EVENT
------------------- ----------- ---------- --------- ------------------------------------------------------------------------

DBRM                                          TPCCP2NO

CUR_DIST                 9413 19:37.15028  6.638344 STMTTYPE          COUNT   AET/OCCUR TCB/OCCUR              COMMITS:  9413
                                0.125056  0.000705 FETCH              9413    0.124290  0.000522
                                                   OPEN               9413    0.000114  0.000028
                                                   UPDATE             9413    0.000651  0.000156

CUR_ITEM                 9413   13.466033  6.441889 STMTTYPE          COUNT   AET/OCCUR TCB/OCCUR              COMMITS:  9413
                                0.001431  0.000684 FETCH             93717    0.000139  0.000066
                                                   OPEN               9413    0.000049  0.000028

CUR_STOCK               93716 1:39.422634 27.807799 STMTTYPE          COUNT   AET/OCCUR TCB/OCCUR
                                0.001061  0.000297 CLOSE             93716    0.000043  0.000026
                                                   FETCH             93716    0.000588  0.000118
                                                   OPEN              93716    0.000044  0.000026
                                                   UPDATE            93716    0.000387  0.000127
```

Figure 6-27 Sample SQL Activity Report

The next step is to obtain information about the DISTRICT table to try to understand why it is taking so long to process. Using simple SQL queries against the DB2 system catalog, we see that the DISTRICT table only has 20 rows and that it does not have an index. However, with only 20 rows in the table, you would imagine that the whole table would be resident in DB2's local buffer pools or the Group Buffer Pools in the CF the whole time, so the absence of an index would not make much difference (or so we thought).

Having identified which table we are interested in, the next step is to look at the logic of the TPNO program to understand which tables the program is using, and how it is using them. The logic of TPNO is as follows:

- TPNO reads the Customer and the Warehouse tables for the inputting terminal (the terminal name determines the warehouse).
- It retrieves the requested row from the District table and update that row.
- It inserts a row into the New Order table.
- It inserts a row into the Order table.
- It inserts an average of 10 rows into the Orderline table.
- For each Orderline, it reads the Item table and updates the Stock table.
- It issues a commit at the end of the transaction. All locks are held until the commit is issued.

We are able to make some observations based on this:

- The bulk of the time that TPNO is suspended is because of the DISTRICT table.
- TPNO reads just one row from the DISTRICT table, makes an update to that row (meaning that it gets an X lock), and then holds that lock until the transaction ends.
- TPNO inserts rows into a number of other tables; however, no major contention is observed on those tables

In order to try to understand the contention on DISTRICT, we looked at the number of concurrent transactions on each system (31 x .9 sec - 28) and the number of rows in the table (20).

To get a better understanding of why TPNO is being delayed, we ran a detailed locking trace report (using the LOCK TRACE LEVEL(DETAIL) keywords). An extract from the report is shown in Figure 6-28 on page 167.

```
 LOCATION: DB8Q                          OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4)              PAGE: 1-7
      GROUP: DB8QU                              LOCKING TRACE - DETAIL                    REQUESTED FROM: NOT SPECIFIED
     MEMBER: D8Q1                                                                                    TO: NOT SPECIFIED
   SUBSYSTEM: D8Q1                                                                      ACTUAL FROM: 07-02-15 22:39:20.05
 DB2 VERSION: V8                                    SCOPE: MEMBER                        PAGE DATE: 07-02-15
OPRIMAUTH CORRNAME CONNTYPE
 ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP            --- L O C K   R E S O U R C E ---
 PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                   EVENT SPECIFIC DATA
 ------------------------------ ----------------- -------- --------- ---------------------- ----------------------------------------

 CICSUSER TPCI     CICS         22:39:20.06048765 LOCK     DATAPAGE  DB  =DBCRWW1
 CICSUSER POOL     C029BD065FA2                   AVOIDANCE          OB  =TDIST000
 TPCCP2NO DSTCPA21                                                   PAGE=X'000002'

                               22:39:20.06050657 LOCK     DATAPAGE  DB  =DBCRWW1
                                                 AVOIDANCE          OB  =TDIST000
                                                                    PAGE=X'000003'

                              22:39:20.06052373 LOCK     DATAPAGE  DB  =DBCRWW1
                                                 AVOIDANCE          OB  =TDIST000
                                                                    PAGE=X'000004'

                               22:39:20.06057718 LOCK     DATAPAGE  DB  =DBCRWW1          DURATION=MANUAL   STATE=U     XES PROP=Y
                                                 REQUEST            OB  =TDIST000          RSN CODE= 0       RTNCD= 0    XES FORC=N
                                                                    PAGE=X'000005'         NMODIFY GLOBAL L-LOCK         XES ASYN=N
                                                                                           PARENT  =X'7F5A9A00'
                                                                                           HASH    =X'00046805'

                               22:39:20.06062790 UNLOCK   DATAPAGE  DB  =DBCRWW1          DURATION=MANUAL+1 STATE=U     XES PROP=Y
                               22:39:20.06057718 REQUEST            OB  =TDIST000          RSN CODE= 0       RTNCD= 0    XES FORC=N
                                                                    PAGE=X'000005'         NMODIFY GLOBAL L-LOCK         XES ASYN=N
                                                                                           HASH    =X'00046805'
                               22:39:20.06064636 LOCK     DATAPAGE  DB  =DBCRWW1
                                                 AVOIDANCE          OB  =TDIST000
                                                                    PAGE=X'000006'

                               22:39:20.06066418 LOCK     DATAPAGE  DB  =DBCRWW1
                                                 AVOIDANCE          OB  =TDIST000
                                                                    PAGE=X'000007'

                               22:39:20.06068706 LOCK     DATAPAGE  DB  =DBCRWW1          DURATION=MANUAL   STATE=U     XES PROP=N
                                                 SUSPEND            OB  =TDIST000          ORIG.RSN=LATCH CONT GENER     XES FORC=N
                                                                    PAGE=X'000008'         NMODIFY GLOBAL L-LOCK
                                                                                           PARENT  =X'7F5A9A00'
                                                                                           HASH    =X'00072808'

                               22:39:20.11648490 LOCK     DATAPAGE  DB  =DBCRWW1          SUSP.TIME =0.055798  LOCAL CONTENTION=Y*
                               22:39:20.06068706 RESUME             OB  =TDIST000          DURATION  =MANUAL    LATCH CONTENTION=N
                                                                    PAGE=X'000008'         STATE     =U         IRLM QUEUED REQ =N
                                                                                           RESUME RSN=NORMAL    GLOBAL CONT.    =N
                                                                                           XES PROP  =N         NOTIFY MSG SENT =N
```

Figure 6-28 DB2 lock trace report

This report was very illuminating, and provided great insight into what the TPNO transaction was doing. In order to understand the report and how it helped us, it is vital to understand DB2 locking, an overview of which is contained in 3.4.4, “Levels of DB2 locking” on page 43. You also need to understand the concept of DB2 lock avoidance, as described in 3.4.6, “Lock avoidance” on page 46.

More detailed information is available in *DB2 for z/OS Data Sharing: Planning and Administration*, and in the IBM Redbooks publication *Locking in DB2 for MVS/ESA Environment*, SG24-4725 (this is not a new book, but most of the information it contains is still valid).

The first thing the report highlighted was the fact that TPNO had to look at every row in the DISTRICT table until it found the requested row, despite the fact that the program was doing

a direct access. We know this because the report shows that each successive lock request was for the next sequential page in the table (the DISTRICT table was defined to have just one row per page, and to always use page-level locks). If the program is doing a direct access, why is it having to process each row in the table sequentially?

You may remember that this table does not have an index. So, even though the program is trying to access a given row directly, DB2 has no choice but to evaluate every row in the table sequentially, looking for the requested row. And because every TPNO transaction and one of the other transactions (TPPA) update the DISTRICT table, there is a good chance that the local copy of the required page will have been invalidated by the other DB2, meaning that the transaction suffers an additional wait while the page is refreshed.

The next thing the report highlighted is that DB2 was only able to use Lock Avoidance for a subset of the rows in the table, and you can see this in the EVENT column. The entries that contain LOCK AVOIDANCE are the ones where DB2 was able to use this. However, for the entries that contain LOCK REQUEST, LOCK SUSPEND, LOCK RESUME or UNLOCK REQUEST, DB2 actually had to obtain the stated type of lock for that row.

What is vital to understand here is that the benefit of successful Lock Avoidance, especially at the longer distances, is not the time that was saved by not sending a request to the lock structure in the CF. Instead, and *far* more important, is the fact that if someone else currently held the lock on that row, being able to successfully use Lock Avoidance for that row would mean that TPNO does not have to wait for that lock to be released. This is the main reason why it is *so* important that Lock Avoidance be as successful as possible, especially at longer distances.

And this brings us to the real source of the problem that was delaying TPNO so much. If DB2 is able to exploit Lock Avoidance for a row, it is able to *immediately* evaluate that row (even if another transaction is holding an incompatible lock on the row at that time). And if it is not the requested row, TPNO then moves on to the next row. However, if it *cannot* exploit Lock Avoidance for a particular row, and another transaction holds an incompatible lock on that row, then DB2 must wait for that other lock to be released before it can evaluate that row.

As each transaction runs for longer, it holds onto any locks it holds for longer, delaying any transactions trying to serialize on the same resources. For example, assume Transaction 1 wants to update row 3, and Transaction 2 wants to update row 5. Transaction 1 starts first and gets an X lock on row 3. Transaction 2 starts next, evaluates and passes over rows 1 and 2, then issues a U lock request for row 3 (because for some reason it cannot use lock avoidance for this row).

However, because Transaction 1 already has an incompatible lock on row 3, Transaction 2 must wait for Transaction 1 to complete. And the longer that Transaction 1 runs for, the longer Transaction 2 must wait. When Transaction 1 finally completes and releases the lock, Transaction 2 is able to evaluate row 3, decide it is not the requested one, and then moves on to the next row.

As a result, you can see that the more rows a transaction needs to evaluate when it cannot use Lock Avoidance, the higher is the chance of the transaction being delayed by a row that it is not even interested in. Additionally, as the elapsed time of the transactions increases, the likelihood of lock avoidance being successful decreases, compounding the problem.

However, in this particular situation, the whole question about how often DB2 can successfully use lock avoidance is only an issue because DB2 had to process the DISTRICT table sequentially. If DB2 was able to immediately find the requested row, all the time that DB2 is spending trying to find the requested row would be saved.

Therefore, the one change we made was to add an index to the DISTRICT table. There is no doubt that other changes could also have been made that would have helped, but we believe that making simply this one change should provide a significant benefit.

6.8 Measurement at 80 km with index

For the next set of measurements, at 80 km, the *configuration* was unchanged from that used for the previous 80 km measurement.

However, for this measurement, we *did* make just one “small” change to the system setup. As discussed in 6.7, “Addressing contention” on page 165, we identified that the lack of an index on the DISTRICT table was responsible for a large part of the delays being experienced by TPNO. Therefore, prior to running the 80 km measurements again, we added an index to that table. Everything else, however, was exactly the same as in all prior runs.

6.8.1 Transaction response times

Table 6-145, Table 6-146, and Table 6-147 show the transaction rate, average response time, and goal achievement, for all CICS transactions, for the TPOS transaction, and for just the TPNO transaction, respectively, across both systems in the data sharing group.

Table 6-145 Information for all CICS transactions - sysplex level

All transactions	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	278	269	249	209	178	259
Resp time (milliseconds)	17	42	104	249	418	64
Goal achievement %	100	100	96.6	79	70.9	99.1

Table 6-146 Information for TPOS transactions - sysplex level

TPOS	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	9.8	9.5	8.9	7.6	6.5	9.2
Resp time (milliseconds)	4.0	5.0	7.0	6.5	12.0	9.5
Goal achievement %	100	100	100	100	100	100

Table 6-147 Information for TPNO transactions - sysplex level

TPNO	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	97	94	87	73	62	90
Resp time (milliseconds)	41	100	251	580	842	154
Goal achievement %	100	100	91.8	45.8	24.1	98.1

You can see that adding the index has had a positive effect on the overall average response time, bringing the goal achievement from 70.9% back up to 99.1%. Even more impressive is the improvement in the TPNO transaction.

At the sysplex level, the goal achievement has recovered from 24.1% back up to 98.1%. In fact, looking at the number of transactions and the response times, the performance being delivered by TPNO with the new index at 80 km is better than the performance being achieved *without* the index at just 40 km.

Table 6-148 on page 170 and Table 6-149 on page 170 show the response times across all transactions for each of the #@$2 and #@$3 systems, respectively.

Table 6-148 Average response times across all CICS transactions on system #@$2 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	139	135	127	106	89	132
Resp time (milliseconds)	16	34	88	230	414	48
Goal achievement %	100	100	97.5	81.8	73.7	99.4

Table 6-149 Average response times across all CICS transactions on system #@$3 (in milliseconds)

All transactions	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	139	133	122	103	89	127
Resp time (milliseconds)	17	49	120	268	423	83
Goal achievement %	100	100	95.7	76.2	68.2	98.8

Notice that the number of transactions completed and the response times we experienced improved significantly. Another interesting observation is that there is now a larger gap between the response times being delivered by the two systems. The reason for this is that with a lot of contention removed from the equation, the direct impacts of distance (longer DASD and CF response times) are now playing a larger role in the transaction response times.

Table 6-150 and Table 6-151 show the transaction rates and response times for the TPOS transaction. The TPOS performance on system #@$2 was back to the level seen around 40 km. TPOS on system #@$3, on the other hand, was basically unchanged from the measurement before the index as added.

The reason for this is that nearly all of that response time is made up of the direct impacts of distance (long CF and DASD response times), so reducing the contention did not provide as much benefit for TPOS on system #@$3.

Table 6-150 Average AOR response times for TPOS transaction on system #@$2 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	4.9	4.8	4.5	3.9	3.3	4.7
Resp time (milliseconds)	4.0	4.0	6.0	4.0	11.0	6.0
Goal achievement %	100	100	100	100	100	100

Table 6-151 Average AOR response times for TPOS transaction on system #@$3 (in milliseconds)

TPOS transactions	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	4.9	4.7	4.4	3.7	3.2	4.5
Resp time (milliseconds)	4.0	6.0	8.0	9.0	13.0	13.0
Goal achievement %	100	100	100	100	100	100

The number of TPNO transactions executed on each system, and the response times experienced, improved significantly and were closer to the levels we saw at just 20 km. This is shown in Table 6-152 and Table 6-153.

Perhaps the most impressive number is the recovery in the percent of transactions meeting the WLM goal, from 29.8% and 18.4% without the index, up to 98.6% and 97.5% with the index. Also notice that, in relative terms, there is a larger disparity between the #@$2 and #@$3 response times, presumably because the “smoothing” effect of contention on both systems has been reduced.

Table 6-152 Average AOR response times for TPNO transaction on system #@$2 (in milliseconds)

TPNO transactions	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	49	47	44	37	31	46
Resp time (milliseconds)	40	83	212	532	787	117
Goal achievement %	100	100	94.3	52.6	29.8	98.6

Table 6-153 Average AOR response times for TPNO transaction on system #@$3 (in milliseconds)

TPNO transactions	0 km (base)	20 km	40 km	60 km	80 km	80 km + index
Txn rate (per second)	49	47	43	36	31	44
Resp time (milliseconds)	42	117	289	628	897	193
Goal achievement %	100	100	89.4	39	18.4	97.5

To make it easier to see the changes in response time as the distance is changed, we provide the preceding information in a graphic format in Table 6-29 on page 172. You can see that the transactions running in system #@$3 continue to be more impacted by the distance than those in system #@$2, but the response time improvement for both systems was dramatic.

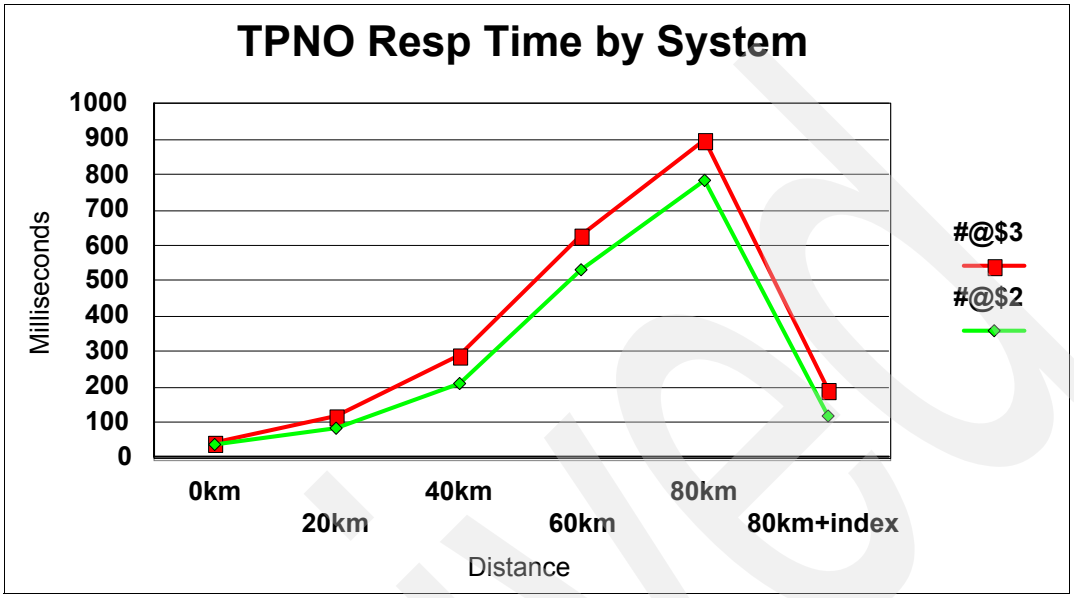


Figure 6-29 TPNO Response time by system for each distance

To verify the numbers reported by DB2PE and RMF, we also ran a pair of CICS PA reports for the TPNO transaction for each system, showing the response time from the perspective of the TORs as well as the AORs. As you can see in Table 6-154 and Table 6-155, the response time as seen by the AOR continues to closely match the transaction response time as seen by both DB2 and WLM.

Table 6-154 CICSPA response times (in milliseconds) for TPNO transaction for #@$2

	0 km	20 km	40 km	60 km	80 km	80 km + index
#@$2 TOR	41.7	86.5	215.3	540	795	122.1
#@$2 AOR	41.2	85.2	214.6	533.2	789.5	119.7

Table 6-155 CICSPA response times (in milliseconds) for TPNO transaction for #@$3

	0 km	20 km	40 km	60 km	80 km	80 km + index
#@$3 TOR	44.1	118.3	287.8	630.7	901.6	194.1
#@$3 AORs	43.4	118.5	290.7	629.7	900.3	196

You can see that CP/SM is still shipping transactions to the AORs on system #@$2 because they are delivering substantially better response times than the AORs on system #@$3.

TPNO response time breakout

Next, we look in more detail at the components of the TPNO response time on each system. Table 6-156 on page 173 and Figure 6-30 on page 173 show the response components on system #@$2.

Table 6-156 Average DB2 times for TPNO transaction on system #@$2 *(in milliseconds)*

Suspend time component	0 km	20 km	40 km	60 km	80 km	80 km + index
Local Lock/Latch contention	9.4	31.7	124	414.3	657.1	28.5
Synchronous I/O delays	2.2	2.5	3.1	3.4	3.5	3.7
Global contention	7.0	22.4	47.6	62.5	67.3	45.1
Asynchronous CF Requests	2.9	3.1	3.9	4.5	4.8	3.9
Update Commit time	7.1	12.1	17.4	23.8	29.2	29.1
Other miscellaneous DB2 Class 3 suspends	0.1	0.3	0.9	3.1	3.9	0.8
Other Not Account by DB2	0.4	0.4	0.6	1.0	.9	0.7
CPU time spent in DB2	4.6	4.8	4.9	5.0	5.0	4.8
Non-DB2 CPU time	1.1	1.2	1.2	1.2	1.2	1.1
Total as seen by DB2	38.4	83.2	207.7	531.3	786.5	122.8

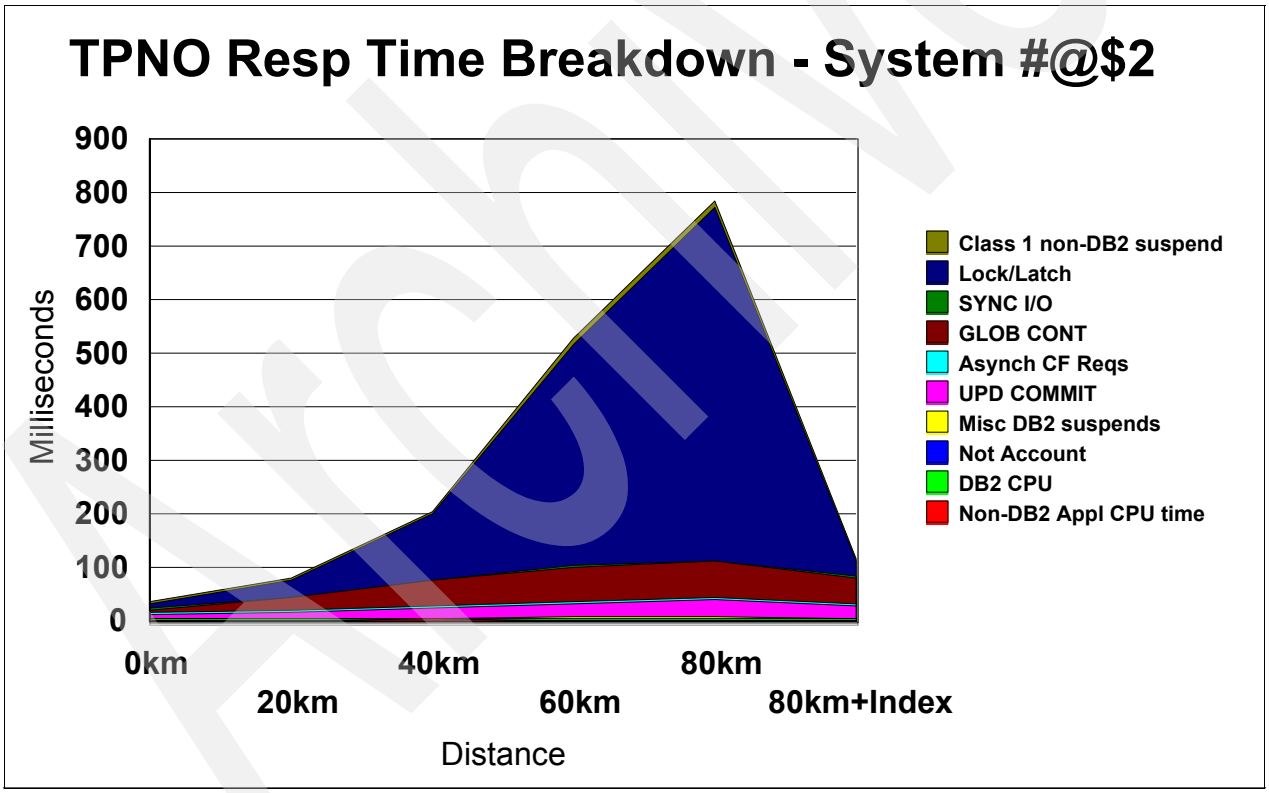


Figure 6-30 TPNO response time breakdown for system #@$2

Table 6-157 on page 174 and Figure 6-31 on page 174 show the components for system #@$3.

Table 6-157 Average DB2 times for TPNO transaction on system #@$3 *(in milliseconds)*

Suspend time component	0 km	20 km	40 km	60 km	80 km	80 km + index
Local Lock/Latch contention	9.8	28.5	132.5	415.3	642.2	17.1
Synchronous I/O delays	2.2	3.3	4.4	5.6	5.8	6.3
Global contention	9.6	53.0	106	150.2	179.9	100.5
Asynchronous CF Requests	2.6	7.2	11.7	15.2	19.2	21.6
Update Commit time	6.8	11.7	16.6	21.2	25.9	25.5
Other miscellaneous DB2 Class 3 suspends	0.1	0.4	1.3	2.5	3.3	1.5
Other Not Account by DB2	0.6	2.2	4.3	6.1	7.0	8.8
CPU time spent in DB2	4.5	4.6	4.9	5.1	5.3	5.6
Non-DB2 CPU time	1.1	1.2	1.2	1.2	1.2	1.3
Total as seen by DB2	40.5	116.3	288.3	627.9	895.9	193.9

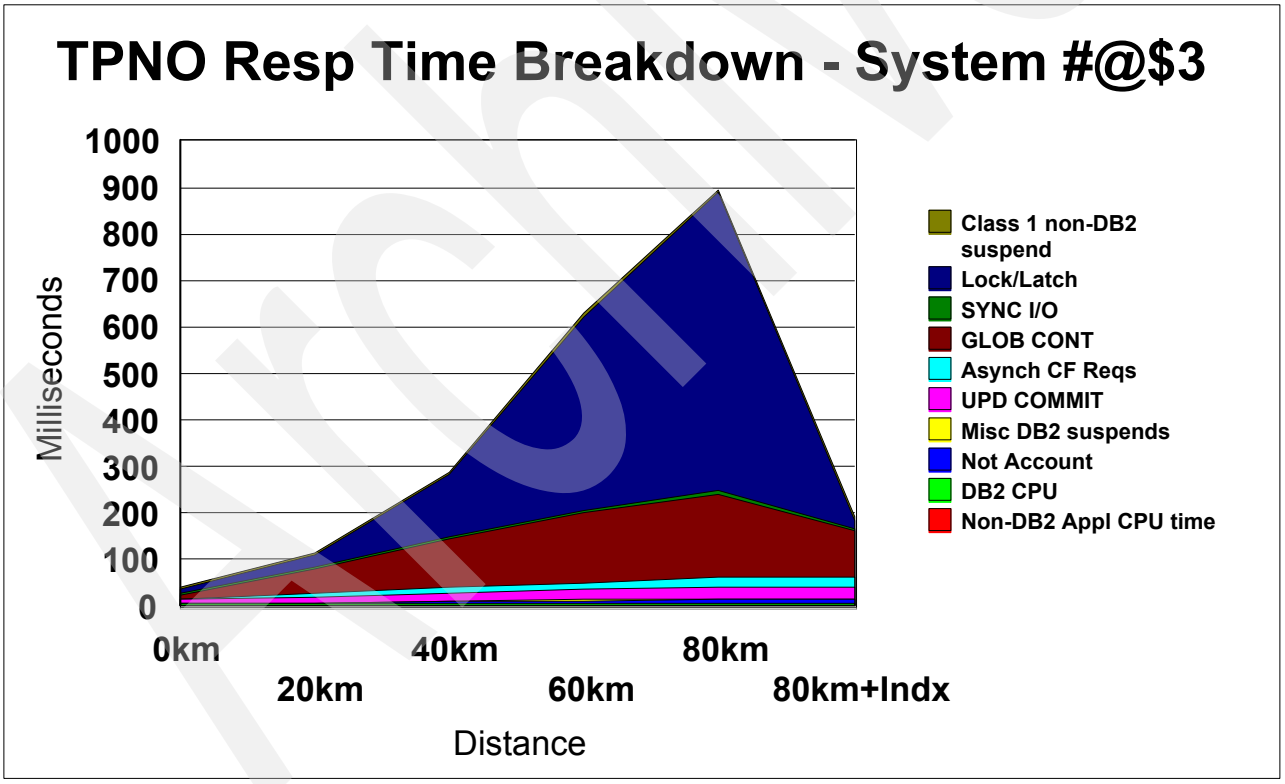


Figure 6-31 TPNO response time breakdown for system #@$3

As you can see in Figure 6-30 on page 173 and Figure 6-31, the TPNO response times delivered by the two DB2 subsystems have improved dramatically, with the bulk of the reduction being in the Lock/Latch category.

6.8.2 Lock/Latch

Table 6-158 contains the Lock/Latch suspend time and the number of suspension events per transaction for both systems.

Table 6-158 Average Lock/Latch suspension time and events for TPNO transaction

Lock/Latch suspension	0 km	20 km	40 km	60 km	80 km	80 km + index
Total suspension time - #@$2	9.4	31.7	124	414.3	657.1	28.5
No. of suspension events - #@$2	1.2	1.9	3	4.3	4.7	1.3
Total suspension time - #@$3	9.8	28.5	132.5	415.3	642.2	17.1
No. of suspension events - #@$2	1.3	2.2	4.3	5.8	6.3	1.8

You can see that adding the index has had a huge positive impact on the amount of time each TPNO transaction is being delayed because of local contention, dropping by over 600 ms on both systems. Both the number of events, and the duration of each event, are below that encountered in the 20 km measurement.

By adding the index to the DISTRICT table, DB2 no longer has to do a sequential search through the table, meaning that the only chance of encountering contention on this table is if another transaction happens to be updating the same row at the same time. Without the index, there is a chance for the transaction to hit contention on *every* row prior to the requested row.

For those not familiar with how DB2 handles locking for rows that are only being evaluated, the need for an index on such a small table may not be obvious. It is for this reason that we *strongly* recommend ensuring that the evaluation team includes a DB2 expert with a solid understanding of how DB2 locking works.

6.8.3 Synchronous I/O

The next part of the TPNO response time is time spent waiting for synchronous I/O to complete. Remember that these are I/Os for requested pages that are not found in either the local or global buffer pools, and I/Os for log records written before the commit. Table 6-159 contains the number of events and the total suspend time for both systems.

Table 6-159 Average Sync I/O suspend time and events for TPNO transaction

Synchronous I/O suspension	0 km	20 km	40 km	60 km	80 km	80 km + index
Total suspension time - #@$2	2.2	2.2	3.1	3.4	3.5	3.7
No. of suspension events - #@$2	2.4	1.8	2.5	2.5	2	2.2
Total suspension time - #@$3	2.2	2.9	4.4	5.6	5.8	6.3
No. of suspension events - #@$2	2.5	1.9	2.6	2.7	2.1	2.3

On system #@$2, the number of events per transaction increased by about 10% compared to the measurement without the index, and the total suspend time increased by a similar amount, meaning that the average suspend time per event remains unchanged.

On system #@$3, the number of events also increased by about 10% compared to the measurement without the index, and the reported suspend time also increased by about 10%, meaning that the suspend time per I/O was flat.

6.8.4 Global contention

Global contention was the second largest contributor to TPNO response times as the distance increased. Table 6-160 shows the contention time and the reported number of global contention events on both systems.

Table 6-160 Average global contention and events for TPNO transaction

Global contention	0 km	20 km	40 km	60 km	80 km	80 km + index
Reported global contention time - #@$2	7.0	18.8	47.6	62.5	67.3	45.1
Reported number of global contention events - #@$2	7.5	11.4	14.5	19.9	18.4	18.3
Actual no. of global contention events - #@$2	0.35	0.66	0.7	0.6	0.5	1
Reported global contention time - #@$3	9.6	41.0	106	150.2	179.9	100.5
Reported number of global contention events - #@$3	32.6	23.5	45.7	45	44.5	45
Actual no. of global contention events - #@$3	0.35	0.74	.9	.7	.5	.7

When we added the index, there was a small increase in the number of actual global contention events. As you would expect, the number of converted lock requests was unchanged, so the decrease that was experienced in total global contention time is a result of the reduced time that the transactions were delayed by a transaction in the other DB2. Just as with the local contention, the fact that DB2 no longer had to sequentially process the table means that there is less opportunity for the transaction to be delayed by contention.

You will recall that the reported global contention time value includes time to process lock requests that were converted to be asynchronous by XES. Table 6-161 shows the response time for the DB2 lock structure for accesses from both systems, and Table 6-160 provides the numbers to let us calculate the number of converted lock requests.

Table 6-161 RMF structure activity report for DB8QU_LOCK1 structure

System	Synch Rate	Synch Serv Time (mics)	Asynch Rate	Asynch Serv Time (mics)
#@$2	4330	21.8	1140	66.5
#@$3	25	808.5	4615	866.7

Using these numbers, we see that system #@$2 spent 1.2 ms (17.3 requests x 66.5 microseconds each) waiting for asynchronous lock requests to complete. On system #@$3

(which was 80 km from the lock structure), the total suspend time for asynchronous lock requests was 38.4 ms (44.3 requests x 866.7 microseconds each).

Based on these numbers, system #@$2 is still encountering about 44 ms of global contention suspend time, and system #@$3 is experiencing about 61.5 ms of global contention time.

Had the time been available, it is possible that we could have reduced the global contention time further by more tuning.

6.8.5 Asynch CF Requests

The Asynch CF Requests suspend time field reports on the number of GBP requests that were issued asynchronously. When using single-phase commit (as we were), the suspend time for GBP requests issued as part of commit processing are included in the Update Commit counter, so the Asynch CF Requests counter includes only asynchronous requests issued prior to the commit.

Table 6-162 Average Asynch CF Request suspension time and events for TPNO transaction

Asynch CF Requests	0 km	20 km	40 km	60 km	80 km	80 km + index
Total Asynch CF request suspension time - #@$2	2.9	3.1	3.9	4.5	4.8	3.9
Number of events - #@$2	20.7	20.5	20.4	20	20	20.5
Total Asynch CF request suspension time - #@$3	2.6	7.2	11.7	15.1	19.2	21.6
Number of events - #@$3	20.8	21.3	21.4	20	19.7	21.6

Table 6-162 contains the suspend times for both systems. On system #@$2, the number of events when we added the index was unchanged from the run without an index, but the total suspend time was down a little, by about 1 ms. On system #@$3, the total suspend time increased to 21.6 ms, but the number of events also increased, so the suspend time per event was unchanged.

6.8.6 Update Commit time

The Update Commit Class 3 suspend time, as reported by DB2PE, includes log writes that are issued during commit, plus the GBP requests for the updated pages, plus some time to create the requests to release any locks held by the transaction, and finally, some DB2 transaction cleanup processing. Table 6-163 contains the Update Commit suspend time for both systems.

Table 6-163 Average Update Commit suspend time for TPNO transaction (in milliseconds)

Update Commit suspensions	0 km	20 km	40 km	60 km	80 km	80 km + index
Total suspension time - #@$2	7.0	12	17.3	23.6	28.9	29.1
No. of suspension events - #@$2	1	1	1	1	1	1
Total suspension time - #@$3	6.7	11.5	16.4	20.9	25.6	25.5

Update Commit suspensions	0 km	20 km	40 km	60 km	80 km	80 km + index
No. of suspension events - #@$2	1	1	1	1	1	1

The components of Update Commit consist of CF requests, DASD I/O, and processing in the CPU. Adding an index to the DISTRICT table has no effect on any of these activities. As a result, there was no change in the Update Commit time after the index was added.

6.8.7 Not Account

Not Account time consists of time that the thread was not processing, but DB2 was unable to determine the reason for the delay. A list of possible delay reasons is provided in “Not Account” on page 61.

Table 6-164 Average Not Accounted time (in milliseconds) for TPNO transaction

Not Account time	0 km	20 km	40 km	60 km	80 km	80 km + index
Not Account - #@$2	.4	.4	.6	1.0	.9	.7
Not Account - #@$3	.6	2.2	4.3	6.0	7.0	8.8

The Not Account time on system #@$2 was largely unchanged across the various measurements. In fact, as a percentage of the total transaction time, Not Account on system #@$2 dropped with each increase in distance.

On system #@$3, the Not Account time was consistently around 1% of the total transaction elapsed time, and increased correspondingly with each increase in distance. However, after the index was added and the elapsed time dropped, the Not Account continued to increase. Because our attention was focused on reducing contention, and the Not Account time was only about 1% of the contention time, we did not investigate the reason for this increase.

6.8.8 DB2 CPU time

The DB2 CPU time is the time that DB2 is executing instructions on behalf of the transaction. Table 6-165 contains the CPU time information for both systems.

Table 6-165 Average DB2 CPU time (in milliseconds) for TPNO transaction

DB2 CPU time	0 km	20 km	40 km	60 km	80 km	80 km + index
DB2 CPU time - #@$2	4.6	4.8	4.9	5.0	5.0	4.8
DB2 CPU time - #@$3	4.5	4.6	4.9	5.1	5.3	5.6

As you can see, there appears to be a direct correlation between the elapsed time of the transaction and the amount of DB2 CPU time charged to it. On system #@$2, as the distance increased the CPU time slowly increased, until the last measurement (with the index). In that measurement, both the CPU time and the elapsed time were similar to those for the 20 km measurement.

On system #@$3, the CPU time was similar; a little lower at the shorter distances, and a little higher at the longer distances. The one anomaly was that the CPU time continued to increase

when the index was added, even though the elapsed time decreased. We are not sure if there is a relation between this and the Not Account time, but both behaved in a similar manner, *decreasing* on system #@$2 after the index was added, but *increasing* on system #@$3. Again, because the anomaly is such a tiny portion of the overall elapsed time, we did not spend time investigating this further.

6.8.9 Overall Coupling Facility activity

In addition to the performance of the specific DB2 structures, it is also important to monitor the overall health, performance, and capacity of the Coupling Facility. Metrics that we specifically monitor in the performance runs are CF CPU utilization and subchannel utilization.

The CPU utilization of each CF can be found in the RMF Coupling Facility report. You can see in Table 6-166 that for this measurement, both CFs were very lightly loaded.

Table 6-166 Coupling Facility CPU utilization

	0 km	20 km	40 km	60 km	80 km	80 km + index
FACIL05	14.3%	14.1%	13%	11.1%	9.3%	13.4%
FACIL06	4.9%	5.7%	5.9%	5.0%	4.1%	6.3%

The CF subchannel utilization numbers shown in Table 6-167 and Table 6-168 can be calculated from the values provided in the SUBCHANNEL ACTIVITY report, which is provided after the detail structure activity section of the RMF CF Activity report.

Table 6-167 FACIL05 subchannel utilization

	0 km	20 km	40 km	60 km	80 km	80 km + index
From #@$2	3.5%	4.9%	5.7%	5.7%	5.5%	8.6%
From #@$3	3.4%	13.8%	21.7%	27.4%	30.7%	42.2%

Table 6-168 FACIL06 subchannel utilization

	0 km	20 km	40 km	60 km	80 km	80 km + index
From #@$2	1.2%	3.3%	5.8%	6.9%	7.2%	11.2%
From #@$3	1%	1.9%	2.6%	2.9%	2.9%	5%

The main concern here is the very high subchannel utilization for system #@$3 talking to CF FACIL05. While not displayed here, the number of Path Busy conditions on system #@$3 talking to that CF was 199,000 out of 3.990,000 requests. This is less than the IBM-recommended threshold of 10%, but it is still higher than we would like to see.

The reason for the higher utilizations on CF FACIL06 in the latest measurement is that the CF was still delivering the long response times that it was delivering in the previous (80 km) measurement, but the request rate was up significantly because of the increased number of transactions being processed (50% more transactions were processed in this measurement than in the previous one).

6.8.10 XCF signaling

As can be seen in Table 6-169 on page 180, the number of XCF messages sent from system #@$2 to #@$3 in the DEFAULT transport class increased from 181.7 to 380.9, with nearly all of them being sent via the XCF signalling path structures in the CF. This continues the pattern from earlier measurements, with XCF sending very few messages over the CTCs.

Also, the number of XCF messages has doubled. This reflects both the increased number of transactions that were processed in this measurement, together with more CICS transactions being shipped from #@$3 to #@$2 to take advantage of the better response times being delivered on that system.

Table 6-169 XCF messages sent per second from #@$2 to #@$3

	0 km	20 km	40 km	60 km	80 km	80 km+Indx
CF structures	144.2	277.0	321.2	243.1	180.3	378.4
FICON CTC	24.3	0.3	1.2	1.3	1.4	2.5

As can be seen in Table 6-170, the number of XCF messages sent from system #@$3 to #@$2 in the DEFAULT transport class also increased from 184.3 to 395.3. This measurement also conforms to the pattern of more XCF messages being sent via the CTCs than the signalling structure in the CFs.

Table 6-170 XCF messages sent per second from #@$3 to #@$2

	0 km	20 km	40 km	60 km	80 km	80 km+Indx
CF structures	126.5	158	88.2	67.4	53.5	111.5
FICON CTC	44.1	129.1	244.9	180.9	130.8	283.8

6.8.11 z/OS CPU utilization

Finally, we include overall z/OS CPU utilization and the z/OS capture ratio. The CPU utilization numbers for each system are included in Table 6-171. The CPU utilization on both systems increased, as you would expect with the increase in the number of transactions.

Table 6-171 z/OS CPU utilization

	0 km	20 km	40 km	60 km	80 km	80 km + index
#@$2	45.2%	49.1%	46.7%	40.3%	34.5%	49.1%
#@$3	46.5%	48.7%	47.7%	43.7%	38.9%	56.4%

To determine whether increased distance has any effect on capture ratio, we also included the capture ratios for each distance in Table 6-172. Interestingly, the capture ratio has now improved, back to the levels seen at the original 0 km measurement. This would indicate a possible relationship between capture ratio and levels of contention.

Table 6-172 z/OS capture ratios

	0 km	20 km	40 km	60 km	80 km	80 km + index
#@$2	90%	90%	90%	90.2%	89.3%	90.2%

	0 km	20 km	40 km	60 km	80 km	80 km + index
#@$3	91.2%	90.2%	90.1%	89.8%	90.1%	91.4%

6.9 Summary

There are a number of very important messages to derive from the results presented in this chapter:

- It is not possible to accurately predict what may be the largest impact on your response times (contention) as the distance increases.
- It is not possible to apply any sort of rule of thumb to determine what is the largest distance that sysplex data sharing can be achieved over.

  For example, if you happened to be the customer that had an index on the DISTRICT table, and you could live with a 200 ms response time for TPNO, then the answer might be 80 km.

  On the other hand, if you had exactly the same data, the same data, the same hardware and software, and the same network connections, but no index, then the answer might be somewhere between 25 and 30 km.
- Depending on your applications, your data, your configuration and (especially) how well your applications and data are tuned, you may be able to implement sysplex data sharing over longer distances than you had expected.
- Good, old-fashioned tuning will become important again.

  There has been a growing trend to address performance problems by throwing more hardware at the problem. However, as you get out to extended distances, the amount of time spent sending signals up and down channels and CF links dominates the non-contention part of the response time.

  So doubling the speed of the processor may only yield a small improvement in response times. The only way to effect significant performance improvements is likely to be from tuning the application, maximizing the hit ratio for buffers, increasing I/O blocksizes (to reduce the number of interactions to send a given amount of data), minimizing channel and CF link contention, and tuning the application to minimize the amount of lock contention that the transaction experiences.
- A poorly tuned configuration that may go unnoticed at 0 km could result in unacceptable response times when the distance between the sysplex components increases.

  In the measurements in this chapter, the configuration that was delivering 45 ms response times at 0 km was the same configuration that delivered 900 ms response times at 80 km. However, because everything happens so quickly at 0 km, the poor tuning was not obvious until the distance started increasing.

One of the lessons we learned early in the project was the importance of having the appropriate expertise on the team that is evaluating your experience and results. Understanding all the performance reports, and the relationships between the different fields in each report, is not as simple as you may expect.

6.9.1 Benchmarking

The official IBM position regarding giving advice for any customer considering implementing a multisite data sharing sysplex is that they benchmark the target configuration in advance of

making a final decision. Based on our experiences in this project, we fully support that position.

The fact is that the only way to know how a given distance will impact your production workload is to actually test it. It is possible that one customer may test at 50 km and be delighted with the results, whereas another customer might test at the same distance and deem the results to be unacceptable. Until you actually test it, there is no way to know.

Another critical point about testing is that you must test with a production or production-like workload, with production-level transaction rates. IBM has tested sysplexes up as far as 100 km (at the time of writing), so we know that the technology will work (although it is always prudent to thoroughly test anything that may be new to you). However, this testing cannot determine if the performance that your target configuration can deliver will be acceptable to you.

We know of customers who are considering implementing a multisite sysplex, that were only planning on testing the final configuration with a test sysplex. However, the transaction mix, and certainly the transaction rate, on a test sysplex is very unlikely to be anywhere close to that of a production system, so the results that would be obtained really tell you very little (if anything) about how your production sysplex would behave over the same distance.

If you already have a multisite sysplex, but are planning to increase the distance between the two sites, you are ideally positioned to run your own benchmark by adding extra fiber to the existing links between the two sites. It is possible to buy devices called “fiber suitcases” which contain variable lengths of fiber (these devices are made to order, so you can specify how much fiber you want in each suitcase). You can then daisy-chain these devices to the existing cross-site dark fibers. This is not inexpensive, and the project must be carefully planned and implemented, but it is the best way to determine exactly how your target configuration will perform in your environment.

A

Definition statements for DB2 resources

This appendix provides the DB2 DDL statements that were used to define the table spaces, tables, and indexes that were used for the DB2 workload used in the measurements in this book.

DDL statements

Example A-1 contains the DDL statements used to define the DB2 resources used in this book. Note that these statements include the index on the DISTRICT table - this was only added after the first 80 km measurement.

Example: A-1 DB2 DDL statements

```
**================================================
**================================================
  SET CURRENT SQLID='KYNEF' ;
**================================================
  DROP DATABASE DBCRWW1;
  COMMIT;

**================================================

  CREATE  DATABASE DBCRWW1;
  COMMIT;

**================================================

  CREATE TABLESPACE TWAREOOO IN DBCRWW1
     USING STOGROUP SGWHSMS1 PRIQTY 48 SECQTY 1000
     FREEPAGE 0
     PCTFREE  99
     BUFFERPOOL BP1
     LOCKSIZE PAGE
     CLOSE NO;

  CREATE TABLESPACE TDISTOOO IN DBCRWW1
     USING STOGROUP SGWHSMS1 PRIQTY 96 SECQTY 1000
     FREEPAGE 0
     PCTFREE  99
     BUFFERPOOL BP1
     LOCKSIZE PAGE
     CLOSE NO;


  CREATE TABLESPACE THISTOOO IN DBCRWW1
     USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000
     FREEPAGE 0
     PCTFREE 0
     BUFFERPOOL BP1
     LOCKSIZE PAGE
     CLOSE NO;

  CREATE TABLESPACE TITEMOOO IN DBCRWW1
     USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000
     FREEPAGE 0
     PCTFREE 0
     BUFFERPOOL BP1
     LOCKSIZE PAGE
     CLOSE NO;

  CREATE TABLESPACE TORDROOO IN DBCRWW1
    NUMPARTS 10 (
      PART 1 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
      PART 2 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
```

```
    PART 3 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 4 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 5 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 6 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 7 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 8 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 9 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 10 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 )
   FREEPAGE  40
   PCTFREE   22
   BUFFERPOOL BP5
   LOCKSIZE PAGE
   CLOSE NO;
COMMIT;
CREATE TABLESPACE TODLN000 IN DBCRWW1
  NUMPARTS 10 (
    PART 1 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 2 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 3 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 4 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 5 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 6 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 7 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 8 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 9 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 10 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 )
  FREEPAGE 10
  PCTFREE  45
  BUFFERPOOL BP6
  LOCKSIZE PAGE
  CLOSE NO;


CREATE TABLESPACE TCUST000 IN DBCRWW1
  NUMPARTS 10 (
    PART 1 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 2 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 3 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 4 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 5 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 6 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 7 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 8 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 9 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 ,
    PART 10 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 2000 )
   FREEPAGE 0
   PCTFREE 10
   BUFFERPOOL BP7
   LOCKSIZE PAGE
   CLOSE NO;


CREATE TABLESPACE TNORD000 IN DBCRWW1
  NUMPARTS 10 (
    PART 1 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 2 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 3 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 4 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 5 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
    PART 6 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
```

```
     PART 7 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 8 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 9 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 10 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 )
    FREEPAGE 1
    PCTFREE  22
    BUFFERPOOL BP4
    LOCKSIZE PAGE
    CLOSE NO;

  CREATE TABLESPACE TSTCK000 IN DBCRWW1
    NUMPARTS 10 (
     PART 1 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 2 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 3 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 4 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 5 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 6 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 7 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 8 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 9 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 ,
     PART 10 USING STOGROUP SGWHSMS1 PRIQTY 20000 SECQTY 20000 )
    FREEPAGE 0
    PCTFREE 0
    BUFFERPOOL BP8
    LOCKSIZE PAGE
    CLOSE NO;

  COMMIT;

**=====================================================

  CREATE TABLE WAREHOUSE
     (
      W_ID        SMALLINT      NOT NULL,
      W_NAME      CHAR(10)      NOT NULL,
      W_STREET_1  CHAR(20)      NOT NULL,
      W_STREET_2  CHAR(20)      NOT NULL,
      W_CITY      CHAR(20)      NOT NULL,
      W_STATE     CHAR(2)       NOT NULL,
      W_ZIP       CHAR(9)       NOT NULL,
      W_TAX       DECIMAL(4,4)  NOT NULL,
      W_YTD       DECIMAL(12,2) NOT NULL)
     IN  DBCRWW1.TWARE000;

  CREATE TABLE DISTRICT
     (
      D_ID          SMALLINT      NOT NULL,
      D_W_ID        SMALLINT      NOT NULL,
      D_NAME        CHAR(10)      NOT NULL,
      D_STREET_1    CHAR(20)      NOT NULL,
      D_STREET_2    CHAR(20)      NOT NULL,
      D_CITY        CHAR(20)      NOT NULL,
      D_STATE       CHAR(2)       NOT NULL,
      D_ZIP         CHAR(9)       NOT NULL,
      D_TAX         DECIMAL(4,4)  NOT NULL,
      D_YTD         DECIMAL(12,2) NOT NULL,
      D_NEXT_O_ID   INTEGER       NOT NULL)
     IN  DBCRWW1.TDIST000;
```

```
CREATE TABLE CUSTOMER
   (
    C_ID             SMALLINT      NOT NULL,
    C_D_ID           SMALLINT      NOT NULL,
    C_W_ID           SMALLINT      NOT NULL,
    C_FIRST          CHAR(16)      NOT NULL,
    C_MIDDLE         CHAR(2)       NOT NULL,
    C_LAST           CHAR(16)      NOT NULL,
    C_STREET_1       CHAR(20)      NOT NULL,
    C_STREET_2       CHAR(20)      NOT NULL,
    C_CITY           CHAR(20)      NOT NULL,
    C_STATE          CHAR(2)       NOT NULL,
    C_ZIP            CHAR(9)       NOT NULL,
    C_PHONE          CHAR(16)      NOT NULL,
    C_SINCE          TIMESTAMP     NOT NULL,
    C_CREDIT         CHAR(2)       NOT NULL,
    C_CREDIT_LIM     DECIMAL(12,2) NOT NULL,
    C_DISCOUNT       DECIMAL(4,4)  NOT NULL,
    C_BALANCE        DECIMAL(12,2) NOT NULL,
    C_YTD_PAYMENT    DECIMAL(12,2) NOT NULL,
    C_PAYMENT_CNT    DECIMAL(4)    NOT NULL,
    C_DELIVERY_CNT   DECIMAL(4)    NOT NULL,
    C_DATA           VARCHAR(500)  NOT NULL)
   IN  DBCRWW1.TCUST000;

CREATE TABLE HISTORY
   (
    H_C_ID           SMALLINT      NOT NULL,
    H_C_D_ID         SMALLINT      NOT NULL,
    H_C_W_ID         SMALLINT      NOT NULL,
    H_D_ID           SMALLINT      NOT NULL,
    H_W_ID           SMALLINT      NOT NULL,
    H_DATE           TIMESTAMP     NOT NULL,
    H_AMOUNT         DECIMAL(6,2)  NOT NULL,
    H_DATA           CHAR(24)      NOT NULL)
   IN  DBCRWW1.THIST000;

COMMIT;

CREATE TABLE ORDERS
   (
    O_ID             INT         NOT NULL,
    O_D_ID           SMALLINT    NOT NULL,
    O_W_ID           SMALLINT    NOT NULL,
    O_C_ID           SMALLINT    NOT NULL,
    O_ENTRY_D        TIMESTAMP   NOT NULL,
    O_CARRIER_ID     SMALLINT,
    O_OL_CNT         DECIMAL(2)  NOT NULL,
    O_ALL_LOCAL      SMALLINT    NOT NULL)
   IN  DBCRWW1.TORDR000;

COMMIT;

CREATE TABLE NEWORDERS
   (
    NO_O_ID          INTEGER     NOT NULL,
    NO_D_ID          SMALLINT    NOT NULL,
    NO_W_ID          SMALLINT    NOT NULL)
   IN  DBCRWW1.TNORD000;
```

```
COMMIT;

CREATE TABLE ORDERLINE
   (
    OL_O_ID           INT          NOT NULL,
    OL_D_ID           SMALLINT     NOT NULL,
    OL_W_ID           SMALLINT     NOT NULL,
    OL_NUMBER         SMALLINT     NOT NULL,
    OL_I_ID           INT          NOT NULL,
    OL_SUPPLY_W_ID    SMALLINT     NOT NULL,
    OL_DELIVERY_D     TIMESTAMP,
    OL_QUANTITY       DECIMAL(2)   NOT NULL,
    OL_AMOUNT         DECIMAL(6,2) NOT NULL,
    OL_DIST_INFO      CHAR(24)     NOT NULL)
   IN  DBCRWW1.TODLNO00;

COMMIT;

CREATE TABLE STOCK
   (
    S_I_ID            INTEGER      NOT NULL,
    S_W_ID            SMALLINT     NOT NULL,
    S_QUANTITY        DECIMAL(4)   NOT NULL,
    S_DIST_01         CHAR(24)     NOT NULL,
    S_DIST_02         CHAR(24)     NOT NULL,
    S_DIST_03         CHAR(24)     NOT NULL,
    S_DIST_04         CHAR(24)     NOT NULL,
    S_DIST_05         CHAR(24)     NOT NULL,
    S_DIST_06         CHAR(24)     NOT NULL,
    S_DIST_07         CHAR(24)     NOT NULL,
    S_DIST_08         CHAR(24)     NOT NULL,
    S_DIST_09         CHAR(24)     NOT NULL,
    S_DIST_10         CHAR(24)     NOT NULL,
    S_YTD             DECIMAL(8)   NOT NULL,
    S_ORDER_CNT       DECIMAL(4)   NOT NULL,
    S_REMOTE_CNT      DECIMAL(4)   NOT NULL,
    S_DATA            CHAR(50)     NOT NULL)
   IN  DBCRWW1.TSTCK000;

COMMIT;

CREATE TABLE ITEM
   (
    I_ID              INTEGER      NOT NULL,
    I_NAME            CHAR(24)     NOT NULL,
    I_PRICE           DECIMAL(5,2) NOT NULL,
    I_DATA            CHAR(50)     NOT NULL)
   IN  DBCRWW1.TITEM000;

COMMIT;

CREATE  UNIQUE INDEX XCUSTLAS
   ON CUSTOMER(C_W_ID,C_D_ID,C_LAST,C_FIRST,C_ID)
   USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 10000
   FREEPAGE 0
   PCTFREE 0
   BUFFERPOOL BP2
   CLOSE NO;

COMMIT;
```

```
CREATE  UNIQUE INDEX XITEM000
   ON ITEM(I_ID)
   USING STOGROUP SGWHSMS1 PRIQTY 2000 SECQTY 10000
   CLUSTER
   FREEPAGE 0
   PCTFREE 0
   BUFFERPOOL BP1
   CLOSE NO;

COMMIT;

CREATE  UNIQUE INDEX XODLN000
   ON ORDERLINE(OL_W_ID,OL_D_ID,OL_O_ID,OL_NUMBER)
   CLUSTER (
     PART 1 VALUES (0001,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 2 VALUES (0002,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 3 VALUES (0003,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 4 VALUES (0004,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 5 VALUES (0005,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 6 VALUES (0006,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 7 VALUES (0007,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 8 VALUES (0008,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 9 VALUES (0009,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 10 VALUES (0010,99,999999,99)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 )
   FREEPAGE  10
   PCTFREE  0
   BUFFERPOOL BP3
   CLOSE NO;

COMMIT;
CREATE  INDEX KYNEF.XODLN001
   ON KYNEF.ORDERLINE(OL_AMOUNT,OL_W_ID,OL_D_ID,OL_O_ID,OL_NUMBER)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000
   FREEPAGE  10
   PCTFREE  0
   BUFFERPOOL BP3
   CLOSE NO;

COMMIT;

CREATE  INDEX KYNEF.XODLN002
  ON KYNEF.ORDERLINE(OL_QUANTITY,OL_W_ID,OL_D_ID,OL_O_ID,OL_NUMBER)
           USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000
  FREEPAGE  10
  PCTFREE  0
  BUFFERPOOL BP3
  CLOSE NO;

COMMIT;
```

```
CREATE  UNIQUE INDEX XCUST000
   ON CUSTOMER(C_W_ID,C_D_ID,C_ID)
   CLUSTER (
     PART 1 VALUES (0001,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 2 VALUES (0002,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 3 VALUES (0003,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 4 VALUES (0004,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 5 VALUES (0005,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 6 VALUES (0006,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 7 VALUES (0007,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 8 VALUES (0008,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 9 VALUES (0009,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 10 VALUES (0010,99,9999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 )
   FREEPAGE 0
   PCTFREE 0
   BUFFERPOOL BP2
   CLOSE NO;

COMMIT;

CREATE  UNIQUE INDEX XNORD000
   ON NEWORDERS(NO_W_ID,NO_D_ID,NO_O_ID)
   CLUSTER (
     PART 1 VALUES (0001,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 2 VALUES (0002,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 3 VALUES (0003,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 4 VALUES (0004,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 5 VALUES (0005,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 6 VALUES (0006,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 7 VALUES (0007,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 8 VALUES (0008,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 9 VALUES (0009,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 10 VALUES (0010,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 )
   FREEPAGE 1
   PCTFREE  0
   BUFFERPOOL BP3
   CLOSE NO;

COMMIT;
```

```
CREATE  UNIQUE INDEX XORDR000
   ON ORDERS(O_W_ID,O_D_ID,O_ID)
   CLUSTER (
     PART 1 VALUES (0001,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 2 VALUES (0002,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 3 VALUES (0003,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 4 VALUES (0004,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 5 VALUES (0005,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 6 VALUES (0006,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 7 VALUES (0007,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 8 VALUES (0008,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 9 VALUES (0009,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 ,
     PART 10 VALUES (0010,99,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 10000 SECQTY 20000 )
   FREEPAGE 5
   PCTFREE  0
   BUFFERPOOL BP3
   CLOSE NO;

COMMIT;

CREATE  UNIQUE INDEX XORDRCUS
   ON ORDERS(O_W_ID,O_D_ID,O_C_ID,O_ID DESC)
   USING STOGROUP SGWHSMS1 PRIQTY 1000 SECQTY 10000
   FREEPAGE 0
   PCTFREE  18
   BUFFERPOOL BP9
   CLOSE NO;

COMMIT;

CREATE  INDEX KYNEF.XDIST00
  ON KYNEF.DISTRICT(D_ID,D_W_ID)
           USING STOGROUP SGWHSMS PRIQTY 50   SECQTY 100
  FREEPAGE  10
  PCTFREE  0
  BUFFERPOOL BP3
  CLOSE NO;

COMMIT;
CREATE  UNIQUE INDEX XSTCK000
   ON STOCK(S_W_ID,S_I_ID)
   CLUSTER (
     PART 1 VALUES (0001,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 2 VALUES (0002,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 3 VALUES (0003,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
     PART 4 VALUES (0004,999999)
```

```
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
      PART 5 VALUES (0005,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
      PART 6 VALUES (0006,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
      PART 7 VALUES (0007,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
      PART 8 VALUES (0008,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
      PART 9 VALUES (0009,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 ,
      PART 10 VALUES (0010,999999)
            USING STOGROUP SGWHSMS1 PRIQTY 5000 SECQTY 10000 )
    FREEPAGE 0
    PCTFREE 0
    BUFFERPOOL BP2
    CLOSE NO;

COMMIT;
```

B

Predicting the impact of distance on DB2 data sharing

This appendix describes the methodology we used to try to predict how distance impacts the response time of DB2 transactions. As we step through each component of the response time, you will see that we were able to predict with fair accuracy how some components would change, while others components were completely unpredictable.

Background

Most IBM customers are familiar with the capacity planning and performance projection tools that IBM has used for many years to estimate the affect of hardware and software changes. And based on that experience, it is natural that people would expect something similar for projecting the impact of injecting long distances into a sysplex.

In this appendix, we examine each component of DB2 transaction response times. You will see that although we could accurately project the changes in some parts of the response time, we could not do this for all components. Specifically, we were unable to do it for the two components (local lock/latch contention and global contention) that dominated our response time as the distance increased above 20 km.

To illustrate what can and cannot be modelled, we use the measurements for both systems from the 0 km run, and from the 80 km run. We start by repeating the results of the 0 km run. The results for system #@$2 are shown in Table B-1.

Table B-1 Average DB2 times for TPNO transaction on system #@$2 (in milliseconds)

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.4				657.1
Synchronous I/O delays	2.2				3.5
Global contention	7.0				67.3
Asynchronous CF Requests	2.9				4.8
Update/Commit time	7.1				29.2
Other miscellaneous DB2 suspends	0.1				3.9
Other Not Account by DB2	0.4				.9
CPU time spent in DB	4.6				5.0
Non-DB2 CPU time	1.1				1.2
Total as seen by DB2	**38.4**				**786.5**

The results for system #@$3 are shown in Table B-2.

Table B-2 Average DB2 times for TPNO transaction on system #@$3 (in milliseconds)

Suspend time component	0 km	20 km	40 km	60 km	80 km
Local Lock/Latch contention	9.8				642.2
Synchronous I/O delays	2.2				5.8
Global contention	9.6				179.9
Asynchronous CF Requests	2.6				19.2
Update Commit Time	6.8				25.9
Other miscellaneous DB2 suspends	0.1				3.3
Other Not Account by DB2	0.6				7.0
CPU time spent in DB	4.5				5.3
Non-DB2 CPU time	1.1				1.2

Suspend time component	0 km	20 km	40 km	60 km	80 km
Total as seen by DB2	**40.5**				**895.9**

Lock/Latch and global contention

Lock/Latch represents the time that a transaction is delayed because it is trying to serialize a resource, but another transaction running on the *same* DB2 subsystem requested that serialization first. (There are, of course, some exceptions.)

Global contention is supposed to be the time that a transaction is delayed because it is waiting for serialization on a resource that is held by another transaction running on a *different* DB2 in the data sharing group. (Again, there are exceptions.)

It is very important to note that there could be 1, 2, or 200, transactions “in front” of this one. If the one *immediately* in front of this transaction is on the same DB2, then the contention will be reported as Lock/Latch. If the transaction immediately ahead of this one is in another DB2, then the wait time will be reported as global contention.

This raises two very important points:

- Whether the contention time for a given transaction is reported as Lock/Latch or global contention is somewhat random, depending on the arrival pattern of transactions on each system.
- The time the transaction has to wait mainly depends on the number of transactions and the elapsed time for the transactions that are delaying this one. Unfortunately, there is no way to accurately project how many transactions will be contending for the same resources, or how long they will run for.

With regard to the first point, we recommend that the Lock/Latch and global contention time be combined and treated as simply “contention”. Table B-3 contains the combined suspension times for Lock/Latch and global contention events at 0 km and at 80 km.

Table B-3 Combine Lock/Latch and global contention suspension time for TPNO transaction

Lock/Latch suspension	0 km	20 km	40 km	60 km	80 km
Total contention time - #@$2	16.4				724.4
No. of Lock/Latch suspension events - #@$2	1.2				4.7
No. of global contention suspension events - #@$2	7.5				18.4
No. of actual global contention events - #@$2	0.35				0.5
Total contention time - #@$3	20.4				822.1
No. of Lock/Latch suspension events - #@$3	1.3				6.3
No. of global contention suspension events - #@$3	32.6				44.5
No. of actual global contention events - #@$3	0.35				0.5

The second point means that it will never be possible to accurately project the amount of time the transaction will be delayed because of contention. It is for this reason that IBM recommends that any customer planning to implement a multi-site sysplex should benchmark the planned configuration in advance. It is especially important that the benchmark is run using production-level workloads and workload mixes, because running a test with five transactions a second and only a small subset of the production transactions will tell you very little about how that same workload would perform with 1000 transactions a second and the full mix of production transactions.

However, there is one set of events that are included in global contention that it *is* possible to project, and that is the delay caused by lock requests being converted by XES to be asynchronous requests. Frankly, the real value of this calculation is to establish a baseline that the global contention time is unlikely to drop below. However, it will probably be only a very small part of the total combined Lock/Latch + global contention number (unless you have a *very* rare situation where transactions never encounter contention).

To calculate this value, we need the number of lock requests that were converted to be asynchronous, and we need the asynchronous lock response time. “Global Contention” on page 73 showed how to calculate the number of asynchronous lock requests. And you can get the asynchronous lock response time from the RMF CF reports. Table B-4 shows the response time numbers at 0 km.

Table B-4 RMF structure activity report for DB8QU_LOCK1 structure at 0 km

System	Synch Rate	Synch Serv Time (mics)	Asynch Rate	Asynch Serv Time (mics)
#@$2	5430	21.2	693	75.7
#@$3	3358	21.9	2393	64.5

The corresponding values for the 80 km run are shown in Table B-5.

Table B-5 RMF structure activity report for DB8QU_LOCK1 structure at 80 km

System	Synch Rate	Synch Serv Time (mics)	Asynch Rate	Asynch Serv Time (mics)
#@$2	2845	21.4	681	65.5
#@$3	18	807.0	3478	866.5

For system #@$2, because there was no change in the distance between DB2 and the lock structure, you would expect there to be no significant change in the CF response time between the two measurements (and this turned out to be the case). However, what did change was the percent of lock requests (across *all* transactions) that got converted to asynchronous ones - in the 0 km measurement for system #@$2, 89% of lock requests were synchronous. In the 80 km measurement, that percentage dropped to about 80%. This is also reflected in the number of asynchronous lock requests for the TPNO transaction, which increased from 7.5 to 17.9 per transaction. However, even 17.9 requests at 65 microseconds each is only 1.1 ms.

For system #@$3, we expected the response time for both synchronous and asynchronous requests to increase by about 800 microseconds when going from 0 km to 80 km, and this is very close to what was actually achieved.

From Table B-3 on page 195 we can calculate that the number of asynchronous requests on system #@$3 at 80 km was 44 per transaction (44.5 - .5). If each asynch lock request from

system #@#3 took 866 microseconds, that means that each TPNO transaction on system #@$3 would be delayed by approximately 38.1 ms.

Remember, however, that these are not really global contention events. In a data sharing environment, there will always be a number of lock requests that get sent to the Coupling Facility, and some percentage of these are likely to be converted to asynchronous ones. So, having 44 converted lock requests per TPNO transaction does not necessarily mean that the transaction was encountering a significant amount of real global contention.

Out of a total contention wait time for TPNO on system #@$3 of 822 ms, being able to project 38 of that may not seem very helpful. However, it *is* helpful insofar as it tells you that even if you were able to tune all the global contention away, you would be unlikely to get below a reported global contention wait time of about 38 ms for this particular transaction (obviously other transactions would have a different value).

Synchronous I/O

The next part of the DB2 transaction response time is time spent waiting for synchronous I/Os to complete.

You may recall that there are two types of synchronous I/Os that affect us:

- Non-prefetch database reads. These are where a transaction does a direct (rather than a sequential) read against a table, and the requested page must be retrieved from DASD.
- Log writes. There are basically two types of these:
  - Those that can be done in parallel (so both copies of the duplex log data sets are updated at the same time).
  - Those that must be done in a serial manner, so the second log data set cannot be updated until the write to the first one has completed successfully.

  Unfortunately, DB2 does not tell you how the log writes were handled. However, in some traces we ran, it appeared that most of the writes for our workload were done serially.

Table B-6 shows the suspend time and the number of suspend events for both systems at 0 km and 80 km. In addition to the information provided in Chapter 6, “CICS/DB2 measurements” on page 95, in Table B-6 we have broken out the synchronous I/O waits into waits for database reads, and waits for log writes. This additional level of detail is very helpful, because reads and writes are impacted differently by distance.

Table B-6 Average Sync I/O suspend time and events for TPNO transaction

Synchronous I/O suspension	0 km	20 km	40 km	60 km	80 km
Total suspension time - #@$2	2.2				3.5
No. of suspension events for database reads - #@$2	1.8				1.4
No. of suspension events for log writes- #@$2	0.6				0.6
Total suspension time - #@$3	2.2				5.8
No. of suspension events - #@$3	1.9				1.5
No. of suspension events for log writes- #@$3	0.6				0.6

Irrespective of how far away the other members of the sysplex are, DASD read I/Os from the systems beside the primary DASD (system #@$2) should take about the same time as they do when all systems are together in the same site, but reads from the remote system (system #@$3) would be expected to take about .1 ms per 10 km longer than at 0 km.

As for writes, on system #@$2 they would be expected to take about .8 ms longer. On system #@$3, they should take about 1.6 ms longer (because system #@$3 has the cost of the long distance between it and the primary DASD, plus the same cost again for mirroring back to the secondary DASD).

Taking these expectations, and the numbers from Table B-6 on page 197, we get the following results:

```
Expected increase on System #@$2:
   1.4 Reads  x  0 ms increase +
   0.6 writes to primary log data set x .8 ms increase +
   0.6 writes to secondary log data set x .8 ms increase =
      Increase of about 1 ms

Expected increase on System #@$3:
   1.5 Reads  x .8 ms increase +
   0.6 Writes to primary log data set x 1.6 ms increase +
   0.6 Writes to secondary log data set x 1.6 ms increase =
      Increase of about 3.2 ms
```

Based on these numbers, the expected increase in Synchronous I/O suspend time on system #@$2 was 1 ms, and the actual increase was 1.3 ms, so that system conformed fairly well to the prediction for this delay reason.

On system #@$3, the expected increase in Synchronous I/O suspend time was 3.2 ms, and the actual increase was 3.6 ms, so again, this system was pretty much in line with expectations.

Important: These projections assume no contention on the channels or UCBs, and no increase in LSS channel interface utilization. It is especially important to enable PAV and (if possible) MIDAWs, and to make sure you have enough channel paths when operating at extended distances.

Asynch CF Requests and Update Commit

The *Asynch CF Request* suspend reason reports on the wait time for asynchronous CF requests issued prior to the commit. Unfortunately, it does not provide the required level of detail to help you predict the impact of distance changes, such as how many were reads and how many were writes. Also, if the transaction issues a write to the GBP and (assuming the GBPs are duplexed) both CF requests are issued asynchronously, that one write request may be reported by DB2PE as two Asynchronous CF Request events (if the write to the primary GBP completes before the write to the secondary GBP).

The *Update Commit* suspend reason reports on GBP writes that were issued during the commit[1]. Unfortunately it does not report how *many* GBP requests were issued. It also sums the time for the GBP writes together with an unspecified number of log writes, plus time for releasing the locks, plus some other DB2 cleanup processing. As a result, it is extremely

[1] When using single-phase commit, the GBP writes issued as a result of the commit get reported in the Update Commit category. If the transaction is using two-phase commit, the GBP writes get reported in Asynch CF Requests.

difficult to look at the Update Commit number and make an accurate projection for how an increased distance will impact each part of that number.

And, just to make things even more complex, DB2 does not report the number of GBP requests that were issued synchronously, and the CPU time for those requests is included in the overall Class 2 CPU time for that transaction, and not broken out separately.

All of this suggests that it is not possible to predict how any changes in CF response times that occur because of increased distances will translate to changes in the transaction elapsed time. Fortunately, there is another way to approach this.

While it would be nice to look at each of the wait reasons that DB2 provides (Asynchronous CF Requests and Update Commit, for example), and predict the change for each reason, the real objective of this exercise is to gain some insight into how increases in CF and DASD response times will impact the transaction response time. All GBP requests that are issued directly as a result of the transaction cause the transaction to wait until the request is complete. So, if we know that a CF request is going to take 800 microseconds longer, we can say that the transaction response time will be at least 800 microseconds longer. We might not know if that time will be reflected in increased Asynchronous CF Request time or increased Update Commit time, but that is not so important.

So, if we can identify how many GBP reads and how many GBP writes were issued by the transaction, including both synchronous and asynchronous requests, and if we can isolate just the requests to the primary GBPs (so there is no possibility of double-counting), then we should be well positioned to predict the increase in response time across all the GBP requests, which in turn will be reflected in increased transaction response time. The GBP section of the DB2PE Accounting Report, together with the RMF CF Structure Activity report, helps us achieve all this. Figure B-1 contains a sample DB2PE GBP report (this is the same report that was shown in Figure 4-20 on page 80).

```
PRIMAUTH: CICSUSER  PLANNAME: TPCCP2NO

GROUP TOT4K              AVERAGE     TOTAL
---------------------  --------  --------
GBP-DEPEND GETPAGES       171.59   4611359
READ(XI)-DATA RETUR        10.06    270481
READ(XI)-NO DATA RT         0.00         1
READ(NF)-DATA RETUR        12.49    335669
READ(NF)-NO DATA RT         1.41     38017
PREFETCH PAGES READ         0.00         8
CLEAN PAGES WRITTEN         0.00         0
UNREGISTER PAGE             0.14      3784
ASYNCH GBP REQUESTS        34.93    938737
EXPLICIT X-INVALID          0.00         0
ASYNCH SEC-GBP REQ          0.00         0
PG P-LOCK LOCK REQ         24.14    648614
 SPACE MAP PAGES            1.72     46177
 DATA PAGES                 0.00         0
 INDEX LEAF PAGES          22.42    602437
PG P-LOCK UNLOCK REQ       24.81    666694
PG P-LOCK LOCK SUSP        21.56    579343
 SPACE MAP PAGES            0.63     16971
 DATA PAGES                 0.00         0
 INDEX LEAF PAGES          20.93    562372
WRITE AND REGISTER          8.12    218337
WRITE & REGISTER MULT       4.40    118307
CHANGED PAGES WRITTEN      37.59   1010163
WRITE TO SEC-GBP            N/A       N/A
```

Figure B-1 DB2PE Accounting GBP report

This report provides nearly all the information we need. It tells us the number of reads and writes. If we sum those, we know the total number of GBP requests (this includes both synchronous and asynchronous requests), and we know the number of requests that were

issued asynchronously from the ASYNCH GBP REQUESTS field[2]. If we subtract the number of asynchronous GBP requests from the total, that will tell us how many requests were issued synchronously. As a reality check, examine the RMF CF report to see the percentage of requests to all GBPs that were synchronous - you would expect both of these numbers to be similar.

Now that we know how many, and what type, of GBP requests were issued by the transaction, we need the GBP response times. This is where things get a little more complex. If you are very lucky, the transaction that you are studying might send most of its GBP requests to the same GBP, but that is probably not likely. What you really want is a set of representative GBP response times.

If all the GBPs exhibit similar response times, you could just pick the one that is the most heavily used by your transaction. However, if the transaction uses a number of GBPs at varying levels of intensity, and the GBPs exhibit significantly different response times, then you should get a weighted average response time across all the involved GBP structures. Unfortunately, this is a fairly time-consuming process. But, having done it for one transaction, at least you should be able to reuse it for other transactions that you might want to study.

There is also another “wrinkle” to the process of identifying the GBP response times to use in your calculations. As mentioned, GBP read requests will typically deliver different response times to write requests. And because the transaction must wait for the writes to both copies of a duplex GBP structure to complete, you would typically use the longer of the two response times. However, which one will be longer depends on the location of the writing DB2 in relation to the locations of the primary and secondary structures.

All of this makes the process of identifying which response time to use somewhat complex. The following guidance explains which response time to use, and why:

- For reads from the DB2 that is *beside the primary* GBP, we recommend always using the response time for that structure when all the components are close together. This response time will include writes and cross-invalidates; however at zero distance, the difference in response time for reads and writes will be minimal.
- For reads from the DB2 that is *remote to the primary* GBP, use the response time number from RMF for the system that DB2 resides on.
- For writes from the DB2 that is *remote to the secondary* GBP, use the longer of the response times from that DB2 to the primary and secondary GBPs.
- For writes from the DB2 that is *beside the secondary* GBP, we recommend using the response time that is being achieved by the DB2 that is remote to the secondary GBP (that is, the *other* DB2). Our reasoning for this is as follows:
  - It is likely that the writes to the primary GBP (in the remote site to this DB2) will be taking longer than the writes to the secondary GBP, which is right beside DB2. And it is the longer of the two writes that we want to focus on.
  - Writes to the primary GBP will actually take longer than the response time reported by RMF (because the RMF number includes both reads and writes).
  - On the other hand, the response time for the *secondary* GBP *only* includes writes.
  - Conceptually, the performance of a write from a DB2 in site2 to the primary GBP in site1 should be similar to the performance of a write from DB2 in site1 to the secondary GBP in site2.

[2] Note that the ASYNCH GBP REQUESTS field in the GBP report does *not* report the same things as the ASYNCH CF REQUESTS field in the Class 3 waits part of the report. In terms of the source fields, ASYNCH GBP REQUESTS comes from the QBGAHS field in the DB2 SMF record, but the ASYNCH CF REQUESTS field comes from the QWAXIXLE field.

– So, by using the RMF numbers for the site1 DB2 talking to the secondary GBP in site2, we should be getting a very close to the response time when DB2 in site 2 does a write to the primary GBP in site1.

We should now be able to use this methodology to determine the approximate impact that the increased distance will have on the summed Asynch CF Requests and Update Commit values. Next, we step through an example to see how successful this will be in practice.

Figure B-2 contains the GBP report for system #@$2 at 80 km.

GROUP TOT4K	AVERAGE	TOTAL
GBP-DEPEND GETPAGES	157.36	2945078
READ(XI)-DATA RETUR	7.22	135135
READ(XI)-NO DATA RT	0.00	1
READ(NF)-DATA RETUR	11.88	222333
READ(NF)-NO DATA RT	1.15	21565
PREFETCH PAGES READ	0.00	0
CLEAN PAGES WRITTEN	0.00	0
UNREGISTER PAGE	0.14	2581
ASYNCH GBP REQUESTS	33.74	631458
EXPLICIT X-INVALID	0.00	0
ASYNCH SEC-GBP REQ	0.00	0
PG P-LOCK LOCK REQ	21.78	407686
SPACE MAP PAGES	0.93	17499
DATA PAGES	0.00	3
INDEX LEAF PAGES	20.85	390184
PG P-LOCK UNLOCK REQ	20.11	376361
PG P-LOCK LOCK SUSP	7.63	142731
SPACE MAP PAGES	0.32	6079
DATA PAGES	0.00	1
INDEX LEAF PAGES	7.30	136651
WRITE AND REGISTER	7.15	133756
WRITE & REGISTER MULT	7.66	143431
CHANGED PAGES WRITTEN	31.98	598445
WRITE TO SEC-GBP	N/A	N/A

Figure B-2 GBP report for #@$2 at 80 km

The corresponding numbers for system #@$3 are shown in Figure B-3.

GROUP TOT4K	AVERAGE	TOTAL
GBP-DEPEND GETPAGES	157.41	2921852
READ(XI)-DATA RETUR	7.33	136087
READ(XI)-NO DATA RT	0.02	439
READ(NF)-DATA RETUR	11.84	219746
READ(NF)-NO DATA RT	1.28	23714
PREFETCH PAGES READ	0.00	0
CLEAN PAGES WRITTEN	0.00	0
UNREGISTER PAGE	0.14	2514
ASYNCH GBP REQUESTS	30.53	566615
EXPLICIT X-INVALID	0.00	0
ASYNCH SEC-GBP REQ	0.00	0
PG P-LOCK LOCK REQ	22.39	415573
SPACE MAP PAGES	1.64	30422
DATA PAGES	0.00	3
INDEX LEAF PAGES	20.75	385148
PG P-LOCK UNLOCK REQ	21.07	391137
PG P-LOCK LOCK SUSP	22.26	413185
SPACE MAP PAGES	1.63	30226
DATA PAGES	0.00	3
INDEX LEAF PAGES	20.63	382956
WRITE AND REGISTER	7.19	133376
WRITE & REGISTER MULT	4.33	80424
CHANGED PAGES WRITTEN	33.74	626227
WRITE TO SEC-GBP	N/A	N/A

Figure B-3 GBP report for #@$3 at 80 km

In these figures, the fields that start with READ are the reads to the GBP. The two fields that start with WRITE are the writes to the GBP. The only other field of interest, UNREGISTER PAGE, is conceptually a write (because it changes storage in the CF) but it should display performance that is more like a read request. Table B-7 extracts and summarizes the key numbers from the GBP report.

Table B-7 Summarized GBP activity

	#@$2	#@$3
Reads	20.39	20.61
Writes	14.81	11.52
Total GBP requests	35.2	32.13
Asynch GBP Requests	33.74	30.53
Synch GBP requests	1.46	1.6

Next, we need to obtain the GBP response times that we will use in the calculations. In the interest of brevity, we will use the RMF numbers for the GBP3 structure. In practice, for this workload, the performance of the GBP3 structure was similar to that of all the other GBP structures, so using the GBP3 numbers provided representative results. Figure B-4 and Figure B-5 on page 203 contain the RMF CF reports for the GBP3 structure for the 0 km and 80 km test runs.

```
 STRUCTURE NAME = DB8QU_GBP3        TYPE = CACHE  STATUS = ACTIVE PRIMARY
              # REQ    ------------- REQUESTS -------------   -------------- DELAYED REQUESTS -------------
     SYSTEM   TOTAL              #     % OF  -SERV TIME(MIC)-    REASON    #   % OF  ---- AVG TIME(MIC) -----
     NAME     AVG/SEC            REQ    ALL     AVG   STD_DEV            REQ    REQ   /DEL    STD_DEV   /ALL

     #@$2       629K   SYNC      12K    1.0    41.2     63.1     NO SCH    0    0.0    0.0        0.0    0.0
                1048   ASYNC    617K   51.5   153.2    124.3     PR WT     0    0.0    0.0        0.0    0.0
                       CHNGD       0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                                DUMP      0    0.0    0.0        0.0    0.0

     #@$3       569K   SYNC    3936     0.3    64.7     80.4     NO SCH    1    0.0   41.0        0.0    0.0
              947.8    ASYNC    565K   47.2   125.2    104.5     PR WT     0    0.0    0.0        0.0    0.0
                       CHNGD       1    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                                DUMP      0    0.0    0.0        0.0    0.0

 STRUCTURE NAME = DB8QU_GBP3        TYPE = CACHE  STATUS = ACTIVE SECONDARY
              # REQ    ------------- REQUESTS -------------   -------------- DELAYED REQUESTS -------------
     SYSTEM   TOTAL              #     % OF  -SERV TIME(MIC)-    REASON    #   % OF  ---- AVG TIME(MIC) -----
     NAME     AVG/SEC            REQ    ALL     AVG   STD_DEV            REQ    REQ   /DEL    STD_DEV   /ALL

     #@$2       171K   SYNC       0    0.0     0.0      0.0     NO SCH    0    0.0    0.0        0.0    0.0
              284.6    ASYNC    171K   51.5   279.9    212.7     PR WT     0    0.0    0.0        0.0    0.0
                       CHNGD       0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                                DUMP      0    0.0    0.0        0.0    0.0

     #@$3       161K   SYNC      58    0.0    23.9      2.8     NO SCH    0    0.0    0.0        0.0    0.0
              268.1    ASYNC    161K   48.5   220.5    164.9     PR WT     0    0.0    0.0        0.0    0.0
                       CHNGD       0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                                DUMP      0    0.0    0.0        0.0    0.0
```

Figure B-4 RMF CF report for GBP3 structures at 0 km

```
STRUCTURE NAME = DB8QU_GBP3        TYPE = CACHE  STATUS = ACTIVE PRIMARY
              # REQ    -------------- REQUESTS -------------   -------------- DELAYED REQUESTS -------------
    SYSTEM    TOTAL             #     % OF  -SERV TIME(MIC)-    REASON    #   % OF  ---- AVG TIME(MIC) -----
    NAME      AVG/SEC           REQ    ALL     AVG   STD_DEV            REQ    REQ   /DEL    STD_DEV   /ALL

    #@$2        439K   SYNC   5555     0.7   149.8    393.3     NO SCH    4    0.0  323.8      300.2    0.0
              731.6    ASYNC   433K   53.0   405.0    611.3     PR WT     0    0.0    0.0        0.0    0.0
                       CHNGD     4    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                               DUMP      0    0.0    0.0        0.0    0.0

    #@$3        378K   SYNC   1936     0.2   870.4     89.3     NO SCH   19K   5.2  268.3      262.3   14.1
              630.1    ASYNC   357K   43.7   939.4    371.5     PR WT     0    0.0    0.0        0.0    0.0
                       CHNGD    19K    2.3  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                               DUMP      0    0.0    0.0        0.0    0.0

STRUCTURE NAME = DB8QU_GBP3        TYPE = CACHE  STATUS = ACTIVE SECONDARY
              # REQ    -------------- REQUESTS -------------   -------------- DELAYED REQUESTS -------------
    SYSTEM    TOTAL             #     % OF  -SERV TIME(MIC)-    REASON    #   % OF  ---- AVG TIME(MIC) -----
    NAME      AVG/SEC           REQ    ALL     AVG   STD_DEV            REQ    REQ   /DEL    STD_DEV   /ALL

    #@$2        162K   SYNC      0     0.0     0.0      0.0     NO SCH    2    0.0   65.0       59.4    0.0
              270.3    ASYNC   162K   55.6  1614.5    762.1     PR WT     0    0.0    0.0        0.0    0.0
                       CHNGD     0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                               DUMP      0    0.0    0.0        0.0    0.0

    #@$3        129K   SYNC     30     0.0    25.5      2.9     NO SCH    0    0.0    0.0        0.0    0.0
              215.4    ASYNC   129K   44.3   720.6    773.2     PR WT     0    0.0    0.0        0.0    0.0
                       CHNGD     0    0.0  INCLUDED IN ASYNC   PR CMP    0    0.0    0.0        0.0    0.0
                                                               DUMP      0    0.0    0.0        0.0    0.0
```

Figure B-5 RMF CF report for GBP3 structures at 80 km

Table B-8 contains the response time numbers that we will use for the calculations, extracted from the RMF report using the methodology described above.

Table B-8 GBP response times to be used in calculations

	#@$2	#@$3
Read response time	153.2	939.4
Write response time	1614.5	1614.5

Plugging all these numbers into our formula gives the following results:

```
For system #@$2:
   20.39 Reads x 153.2 microseconds each +
   14.81 Writes x 1614.5 microseconds each =
      27 milliseconds delayed for GBP accesses
For system #@$3:
   20.61 Reads x 939.4 microseconds each +
   11.52 Writes x 1614.5 microseconds each =
      37.98 milliseconds delayed for GBP accesses
```

To these numbers, we need to add something for the log writes and the processing to release the locks, and then perform general cleanup.

Using the RMF Workload Activity reports, we can get the response time for the log write I/Os in each system. These are shown in Figure B-6 on page 204 and Figure B-7 on page 204.

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
2125	8	2011	20882	226.2	1.4	0.2	1.1	0.1	0.0

Figure B-6 Workload activity report for D8Q1 MSTR address space

TOTSRV	CPUSRV	SRBSRV	EXCP	SSCHRT	RESP	CONN	DISC	QPEND	IOSQ
1024	7	943	11054	128.1	2.2	0.2	1.1	0.9	0.0

Figure B-7 Workload activity report for D8Q2 MSTR address space

Assuming that the log records written at commit time are written serially rather than in parallel, this gives us roughly 2.8 ms suspend time on system #@$2, and 4.4 ms suspend time on system #@$3.

Finally, we have some time to release the locks and perform other cleanup. We estimate that this took roughly 2 ms for the TPNO transaction, and this time would be unaffected by any distance in the sysplex.

Table B-9 summarizes all of these numbers for each of the systems.

Table B-9 Summary of suspend time components

	#@$2	#@$3
GBP suspends	27.0	38.0
Log write suspends	2.8	4.4
Release locks and other cleanup	2.0	2.0
Total	31.8	44.4

Comparing the calculated numbers in Table B-9 to the actual measured numbers in Table B-10, you can see that the calculation proved to be quite accurate.

Table B-10 Average Asynch CF Request Suspension time and events for TPNO transaction

Asynch CF Request	0 km	20 km	40 km	60 km	80 km
Total Asynch CF request and Update Commit suspension times - #@$2	9.9				34.0
Total Asynch CF request suspension time - #@$3	8.3				45.6

Not Account delay value

By its very nature, the Not Account delay value is impossible to accurately model because we do not know what activities are contributing to that number. However, based on these measurements we can put forward two observations:

- As a percentage of the total transaction response time, Not Account is very small, and
- Because of this, it is not a major cause of concern.

Table B-11 shows the Not Account times for the 0 km and 80 km runs.

Table B-11 Average Not Accounted (in milliseconds) for TPNO transaction

NOT ACCOUNT time	0 km	20 km	40 km	60 km	80 km
Not Accounted - #@$2	.4				.9
Not Accounted - #@$3	.6				7.0

On the system that was located beside the primary DASD, the lock structure, and the primary GBPs, the Not Account time stayed nearly flat, in absolute terms, across all measurements. On the system that was in the “remote” site, the Not Account time grew, but it remained steady as a percentage of the overall TPNO transaction response time.

DB2 Class 2 CPU time

The Class 2 CPU time is the time spent in DB2 for processing SQL statements, stored procedures, user-defined functions, or triggers. It only includes the time for work specifically requested by the transaction. For example, if DB2 has to move pages from a local buffer pool to the global buffer pool because a threshold has been met, the time to do that work is charged to one of the DB2 address spaces and does not show up in the Class 2 CPU time.

Table 6-173 Average DB2 CPU time (in milliseconds) for TPNO transaction

DB2 CPU Time	0 km	20 km	40 km	60 km	80 km
DB2 CPU Time - #@$2	4.6	4.8	4.9	5.0	5.0
DB2 CPU Time - #@$3	4.5	4.6	4.9	5.1	5.3

We were unable to find any key metric that would allow us to predict how the CPU time would change with distance. Although the CPU time used on #@$3 was slightly higher than on #@$2 at 80 km (but only by 6%), the maximum increase from 0 km to 80 km was only .8 ms. For that reason we felt that it was not worth spending significant time to investigate this small part of the overall response time.

Non-DB2 CPU time

Non-DB2 CPU time is time spent in the application program, but outside DB2. Because our application performs most of its work in DB2, and specifically does not perform any I/Os outside DB2, there was very little change in non-DB2 CPU time as the distance changed. The non-DB2 CPU time on both systems was about 1.1 ms at 0 km. At 80 km, the non-DB2 CPU time on both systems had increased to 1.2 ms.

Summary

The top six contributors to the #@$3 TPNO response time at 80 km were:

Lock/Latch Accounted for 71.6% of response time

Global contention Accounted for 20% of response time

Update Commit Accounted for 2.8% of response time

Asynch CF Requests Accounted for 2.1% of response time

Not Account Accounted for 0.8% of response time

Synchronous I/Os Accounted for 0.6% of response time

Of these, the two largest contributors consist mainly of contention time, and we showed that it is not possible to predict how this will change over time. However, we were able to predict one constituent of the global contention time: the case where DB2 lock requests are converted to asynchronous CF requests.

We are also able to predict the next two contributors: Update Commit and Asynch CF Requests. We do not have sufficient detail to be able to predict how *each* one will change. However, when combined, the projections come very close to the actual.

Based on our experiences, although we do not have visibility to what is making up the Not Account time, it does appear that Not Account stays as a consistent percentage of the overall transaction response time.

Finally, the last component, Synchronous I/Os, grew in line with the projection, based on modelling changes in response times as the distance changes.

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 207. Note that some of the documents referenced here may be available in softcopy only.

- *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322
- *Locking in DB2 for MVS/ESA Environment*, SG24-4725
- *How does the MIDAW facility improve the performance of FICON channels using DB2 and other workloads,* REDP-4201

Other publications

These publications are also relevant as further information sources:

- *CICS DB2 Guide,* SC34-6457
- *DB2 UDB for z/OS V8 Data Sharing Planning and Administration*, SA18-7417
- *DB2 for z/OS Performance Monitoring and Tuning Guide*, SC18-9851
- *z/OS RMF User’s Guide*, SC33-7990

How to get IBM Redbooks

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

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

X

Z

Considerations for Multisite Sysplex Data Sharing

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

Considerations for Multisite Sysplex Data Sharing

Understand how distance impacts data sharing

Sample measurements at distances up to 80 km

Measurements for CICS/DB2

This IBM Redbooks publication provides a broad understanding of the considerations to keep in mind when you implement IBM z/OS sysplex data sharing across geographically distributed sites.

This is a relatively new technology with many complexities, so it can be challenging to understand and accurately predict how its implementation might affect the performance and capacity of your environment. To promote that understanding, we measured and documented a number of workloads over various distances.

In this book, we discuss in detail how the performance changed in each case, and why we experienced the results that we did. While it is not possible to accurately model the impact of distance on data sharing, by providing a discussion of our results you will be better prepared to undertake a similar project yourself.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7263-00 ISBN 0738431265