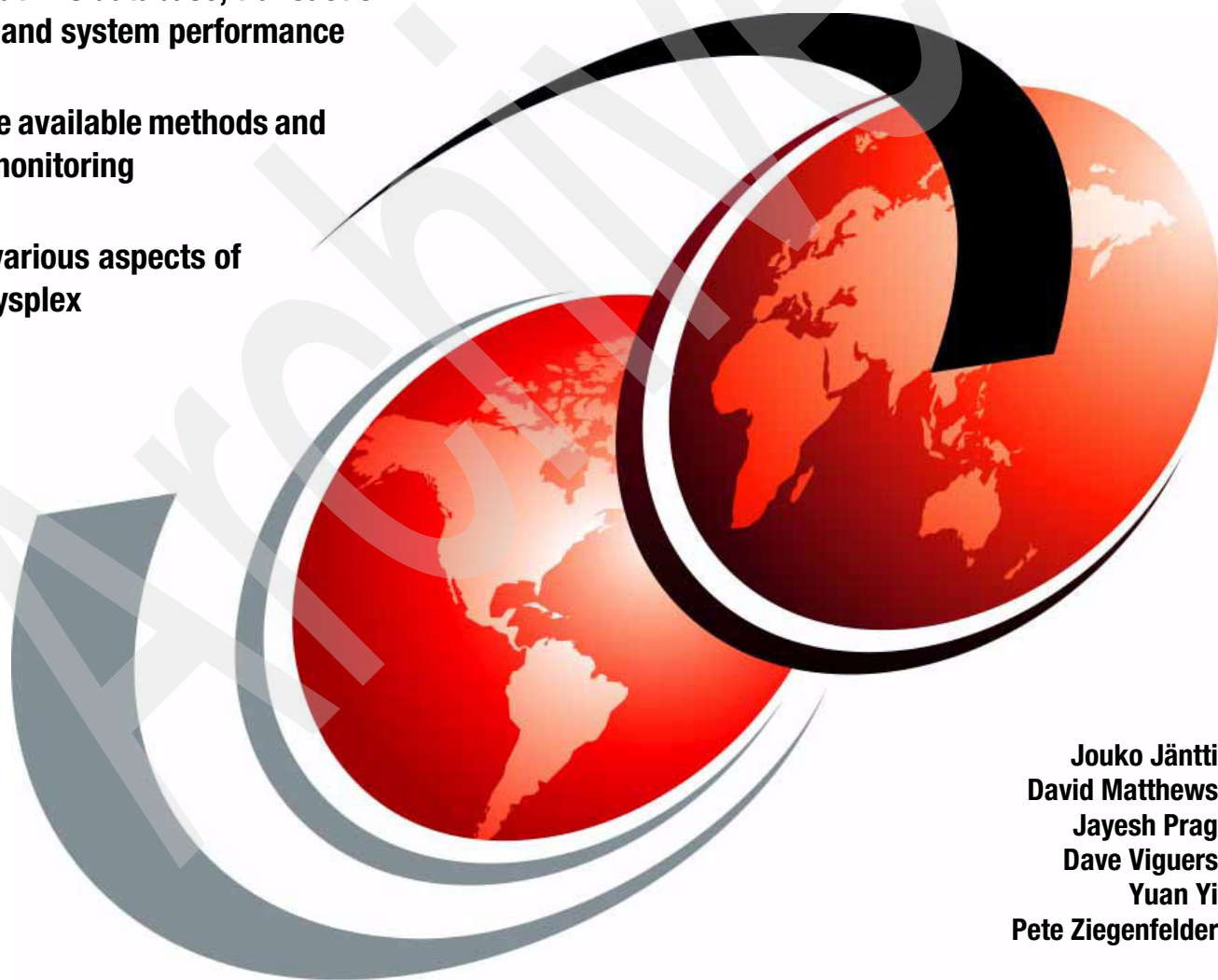


IMS Performance and Tuning Guide

Learn about IMS database, transaction manager, and system performance

Look at the available methods and tools for monitoring

Examine various aspects of Parallel Sysplex



Jouko Jäntti
David Matthews
Jayesh Prag
Dave Viguers
Yuan Yi
Pete Ziegenfelder

Redbooks



International Technical Support Organization

IMS Performance and Tuning Guide

December 2006

Archived

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

First Edition (December 2006)

This edition applies to IMS Version 9 (program number 5655-J38) or later for use with the z/OS Operating System.

© Copyright International Business Machines Corporation 2006. All rights reserved.

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

Contents

Notices	xi
Trademarks	xii
Preface	xiii
The team that wrote this redbook.	xiii
Become a published author	xv
Comments welcome.	xvi
Chapter 1. Defining the performance problem in an IMS environment	1
1.1 Performance overview	2
1.2 Understanding the performance problem	2
1.2.1 Defining a service level agreement.	2
1.2.2 Transaction profiles.	3
1.2.3 Analysis and interpretation	3
1.2.4 Tracking and trending	4
1.3 Events for full function messages	4
1.3.1 Message arrives into IMS	4
1.3.2 Message queuing	5
1.3.3 Message scheduling	5
1.3.4 Scheduling-end to first DL/I call	5
1.3.5 Program elapsed time	5
1.3.6 Sync point processing	6
1.3.7 Message output	6
1.4 Events for Fast Path messages	6
1.4.1 Message arrives into IMS	6
1.4.2 Fast Path EMH queuing	6
1.4.3 Fast Path EMH scheduling	7
1.4.4 Program elapsed time	7
1.4.5 Sync point processing	7
1.5 Events for DBCTL	7
1.5.1 Message arrives into CICS TOR/AOR	8
1.5.2 Application gets control and issues schedule request	8
1.5.3 PSB scheduling	8
1.5.4 Program elapsed time	8
1.5.5 Sync point processing	8
1.5.6 Application issues terminate PSB call.	8
1.6 Common log records produced for transaction flows	8
1.7 Problem identification matrix	9
Chapter 2. Monitoring methodology	11
2.1 Establishing monitoring strategies	12
2.2 Monitoring multiple systems in DB/DC and DCCTL environments	12
2.3 Coordinating performance information in an MSC network.	12
2.4 Monitoring Fast Path systems in DB/DC and DCCTL environments	13
2.5 Transaction flow in DB/DC and DCCTL environments	13
2.6 The IMS Monitor in DB/DC and DCCTL environments	17
2.7 Monitoring procedures in a DBCTL environment	17
Chapter 3. Monitoring tools	19

3.1	IMS monitoring tools	20
3.2	IMS Monitor	21
3.3	IMS Performance Analyzer	26
3.4	File Select and Formatting Print utility (DFSERA10)	27
3.5	Log Transaction Analysis utility (DFSILTA0)	27
3.6	Knowledge-Based Log Analysis (KBLA)	28
3.6.1	MSC Link Performance Analysis	30
3.6.2	Statistic Log Record Analysis	30
3.6.3	DBCTL Transaction Analysis	32
3.6.4	IRLM Lock Trace Analysis	34
3.7	IBM Tivoli OMEGAMON XE for IMS on z/OS	36
3.7.1	OMEGAMON XE for IMS in IMSplex environment	36
3.7.2	Using OMEGAMON XE for IMS for monitoring IMS Connect	40
3.7.3	Additional references	41
3.8	IBM IMS Connect Extensions for z/OS	42
3.9	IBM IMS Buffer Pool Analyzer for z/OS	43
Chapter 4.	IMS and Workload Manager	45
4.1	Workload manager in an IMS world	46
4.1.1	Defining IMS workloads to WLM	46
4.1.2	Rules for ensuring the correct priorities are assigned	47
4.2	CPU management	50
4.3	Memory management	50
4.3.1	Identifying memory-related problems	51
4.4	I/O subsystem	51
4.4.1	Ideas to minimize I/O contention	51
Chapter 5.	Database performance	53
5.1	Access methods	54
5.1.1	Selecting an access method	57
5.2	HISAM as opposed to HD access methods	58
5.2.1	HISAM	58
5.2.2	SHISAM	60
5.2.3	HD access methods	60
5.3	(P)HDAM as opposed to (P)HIDAM	61
5.3.1	Space use	62
5.3.2	Sequential processing	62
5.3.3	I/Os	62
5.3.4	Reorganizations	62
5.3.5	Creeping keys	62
5.3.6	Recommendation summary for (P)HDAM as opposed to (P)HIDAM	63
5.4	HALDB	64
5.4.1	HALDB partition selection	64
5.4.2	Key range partition selection	65
5.4.3	Partition selection exit routine	65
5.4.4	Defining partition selection	65
5.4.5	Recommendation summary for HALDB partition selection	65
5.4.6	HALDB indirect data set lists	66
5.5	Block sizes, CI sizes, and record sizes	66
5.5.1	Index CI sizes and record sizes	66
5.5.2	OSAM block sizes and VSAM ESDS CI sizes	67
5.5.3	FREQ parameter on the SEGM statement	69
5.5.4	Recommendation summary for block sizes, CI sizes, and record sizes	69

5.6	Free space	70
5.6.1	Specifying free space	70
5.6.2	HD space search algorithm	71
5.6.3	Recommendation summary for free space	71
5.7	Randomization parameters	71
5.7.1	Randomizer	72
5.7.2	Number of RAPs	72
5.7.3	Size of root addressable area	73
5.7.4	The BYTES parameter	73
5.7.5	Specifying randomization parameters for PHDAM	73
5.7.6	Recommendation summary for randomization parameters	73
5.7.7	Monitoring HDAM databases	74
5.7.8	Loading or reloading HDAM databases	76
5.8	Fixed length as opposed to variable length segments	76
5.8.1	Variable length segment	77
5.8.2	Fixed length segment	77
5.8.3	Recommendations for fixed as opposed to variable length segments	77
5.9	Pointer options	78
5.9.1	Hierarchic as opposed to child and twin pointers	78
5.9.2	Forward only as opposed to forward and backward pointers	79
5.9.3	HIDAM and PHIDAM root segments	80
5.9.4	Unsequenced dependent segments	80
5.9.5	Defining hierarchical, physical twin, and physical child pointers	81
5.9.6	Recommendation summary for pointer options	81
5.10	SCAN= parameter on the DATASET statement	81
5.11	Multiple data set groups	82
5.12	Compression	84
5.12.1	Key compression as opposed to data compression	85
5.12.2	COMPRTN= parameter	85
5.12.3	Recommendation summary for compression	86
5.13	Encryption	86
5.14	Secondary indexes	87
5.14.1	Secondary index keys	87
5.14.2	Direct as opposed to symbolic pointers	88
5.14.3	Shared secondary indexes	89
5.14.4	Duplicate data	89
5.14.5	User data	89
5.14.6	Sparse indexing	89
5.14.7	Recommendation summary for secondary indexes	90
5.15	Fast Path performance considerations	90
5.15.1	Virtual Storage Option (VSO)	90
5.15.2	Field (FLD) calls support	90
5.15.3	Shared Virtual Storage Option (SVSO)	91
5.15.4	DEDB general performance considerations	91
5.15.5	DASD or channel contention for I/O on DEDB	91
5.15.6	OThread contention	91
5.15.7	Increased I/O or CI contention for independent or dependent overflow	91
5.15.8	Overflow Buffer Allocation (OBA) latch wait	92
5.15.9	DEDB sequential processing	92
5.15.10	I/O error toleration support for DEDB	92
5.15.11	DEDB using the Virtual Storage Option	92
5.15.12	Shared Virtual Storage Option	96
5.15.13	Local buffer pool definitions	97

5.15.14	PRELOAD NOPREL option	99
5.15.15	Block level locking and root-only DEDBs	100
5.15.16	Sequential dependent sharing (shared SDEPs)	100
5.15.17	IMS Fast Path buffers	101
5.15.18	Normal buffer allocation	101
5.15.19	Overflow Buffer Allocation (OBA)	103
5.16	Non-recoverable databases	104
5.17	OSAM as opposed to VSAM	104
5.17.1	Performance results on OSAM and VSAM	105
5.17.2	Recommendation summary for OSAM as opposed to VSAM	105
5.18	Buffer life concept	106
5.19	Overflow sequential access method (OSAM)	106
5.19.1	Tuning OSAM buffers	106
5.19.2	OSAM data set notes	111
5.19.3	OSAM sequential buffering	112
5.20	Virtual storage access method (VSAM)	115
5.20.1	Tuning VSAM buffers	115
5.20.2	VSAM background write	118
5.20.3	VSAM hiperspace buffers	118
5.20.4	VSAM statistics	121
5.20.5	Tuning VSAM data sets	123
5.21	Improve GSAM performance	123
5.22	When to reorganize	124
Chapter 6.	Transaction manager performance	125
6.1	Scheduling to first IMS call	126
6.2	Program load options	127
6.3	Transaction macro parameter options	129
6.4	IMS parameters	130
6.4.1	ARC parameter	130
6.4.2	BSIZ parameter	131
6.4.3	CPLOG parameter	131
6.4.4	DBBF parameter	131
6.4.5	DBFP parameter	131
6.4.6	DBFX parameter	132
6.4.7	Dynamic pools parameters	133
6.4.8	EMHL parameter	135
6.4.9	EXVR parameter	135
6.4.10	Hash tables parameters: LHTS, NHTS, and UHTS	135
6.4.11	Logging parameters	135
6.4.12	LSO parameter	136
6.4.13	Message format buffer pool parameters	136
6.4.14	OTHR parameter	137
6.4.15	Parameters for scheduling pools	137
6.4.16	PI parameters	137
6.4.17	PRLD parameter	137
6.4.18	PST and MAXPST parameters	137
6.4.19	QBUF parameter	138
6.4.20	RECA parameter	138
6.4.21	RES parameter	138
6.4.22	SAV parameter	138
6.4.23	SRCH parameter	138
6.4.24	VAUT parameter	139

6.4.25 VSPEC parameter	139
6.5 Data gathering	139
6.5.1 IMS Monitor	139
6.5.2 Recording the pool transaction	139
6.5.3 Gathering the system monitoring data	140
6.5.4 IMS log data	140
6.6 Page fixing	141
6.7 WLM	142
6.7.1 IRLM address space	142
6.7.2 DBRC address space	142
6.7.3 CQS address space	143
6.7.4 IMS control region	143
6.7.5 IMS DLS address space	143
6.7.6 IMS dependent regions	143
6.8 IMS variable pool considerations	144
6.8.1 Relationship with scheduling	147
Chapter 7. Performance considerations for DBCTL	149
7.1 DBCTL performance considerations	150
7.2 DFSPZPxx	150
7.3 Scheduling	151
7.4 IMS startup parameters for DBCTL	151
7.5 Parallel Sysplex	153
Chapter 8. System considerations	155
8.1 The IMS logger	156
8.1.1 Logging considerations	157
8.2 DBRC	158
8.2.1 DBRC performance considerations	159
8.2.2 Defining the RECON data sets	159
8.2.3 Resolving data set contention problems	161
8.2.4 DBRC RECON maintenance	162
8.3 SMF and RMF considerations	165
8.4 Security considerations	165
8.5 Batch application performance	167
8.5.1 Using DLI or DBB	167
8.6 Utility performance	168
Chapter 9. Application considerations	169
9.1 IMS language interface	170
9.1.1 Structure of a typical program using the language interface	170
9.2 Performance and programming considerations	171
9.2.1 SSA considerations	171
9.2.2 Single as opposed to multiple positioning	173
9.2.3 Variable length segments	174
9.2.4 Secondary indexing	175
9.2.5 Program reusability considerations	175
9.2.6 Processing options and the PROCOPT statement	175
9.2.7 Read only programs	176
9.2.8 PROCOPT=GOT with DBRC SHARECTL	176
9.2.9 Use of checkpointing in batch	177
9.2.10 Multi-streaming your batch processes	178
9.2.11 Why must online programs be serially reusable	178
9.3 Language environment	179

9.3.1 Application and performance considerations in a LE environment	180
Chapter 10. Performance considerations with DB2.	183
10.1 IMS External Subsystem Attach Facility	184
10.1.1 Subsystem member	185
10.2 Tuning the External Subsystem Attach Facility.	187
10.2.1 Thread management.	188
10.2.2 DB2 lock management	189
10.2.3 DB2 free space	191
10.2.4 Static as opposed to dynamic SQL.	191
10.2.5 Security controls	191
10.3 Multi-row FETCH and INSERT	192
10.4 Tools for monitoring	192
10.4.1 IMS Performance Analyzer	193
10.4.2 IMS Monitor.	195
10.4.3 Deadlock report.	198
10.5 When to reorganize your DB2 tablespace or indexspace	199
10.5.1 Tablespace	200
10.5.2 Indexspace	200
10.6 More information	200
Chapter 11. IMS Parallel Sysplex considerations	201
11.1 Hardware and microcode	202
11.1.1 Coupling Facility configuration	202
11.1.2 Coupling Facility microcode	202
11.2 Structure sizing	202
11.3 IRLM considerations	202
11.4 IRLM lock structure	204
11.4.1 Lock structure size	204
11.4.2 False contention	204
11.4.3 Automatic rebuild	204
11.4.4 System-managed duplexing	204
11.5 VSAM cache structure	205
11.6 OSAM cache structure	205
11.7 DEDB considerations	205
11.7.1 Shared VSO	206
11.8 Application considerations	206
11.9 Shared queues	206
11.9.1 IMS parameters	207
11.9.2 Structure size	207
11.9.3 Structure duplexing.	208
11.9.4 Overflow	208
11.9.5 Structure checkpoint.	208
11.9.6 MVS logger	208
11.9.7 FF scheduling differences.	209
11.9.8 FP Parallel Sysplex processing options	209
Chapter 12. IMS On Demand performance	211
12.1 IMS connectivity solutions using IMS Connect	212
12.1.1 Socket types	213
12.1.2 Asynchronous output processing	213
12.1.3 Performance statistics on IMS Connect	215
12.2 IMS SOAP Gateway	215
12.2.1 Performance considerations using the IMS SOAP gateway.	216

12.3 IMS Java environment	217
12.3.1 IMS Java application performance considerations	218
12.3.2 COBOL to Java translations	219
12.3.3 Performance statistics comparing COBOL to Java	219
Appendix A. Guidelines and recommendations.	223
A.1 First step	224
A.2 Choosing an IMS access method	224
A.3 HISAM	224
A.3.1 HISAM performance general guidelines	225
A.4 HDAM	225
A.4.1 HDAM performance general guidelines	226
A.5 HIDAM	227
A.5.1 HIDAM performance general guidelines	227
A.6 OSAM	228
A.6.1 OSAM tuning general guidelines	228
A.7 VSAM tuning general rules	229
A.7.1 VSAM data set tuning general guidelines	230
A.7.2 ESDS performance guidelines	230
A.7.3 KSDS performance guidelines	231
A.8 Secondary index performance guidelines	231
A.9 Block or CI size performance guidelines	232
A.10 FREESPACE performance guidelines	232
A.11 Segment edit/compression performance considerations	232
A.12 Programming performance considerations	232
A.13 Logging performance considerations	233
A.14 Use HDAM physical sequence sort and reload	233
A.15 I/O error processing	233
A.16 What is a Kilobyte	234
Appendix B. Coding examples.	237
B.1 Sparse index examples	238
B.1.1 Source segment	238
B.1.2 Index segment	241
Abbreviations and acronyms	245
Related publications	249
IBM Redbooks	249
Other publications	249
Online resources	251
How to get IBM Redbooks	251
Help from IBM	251
Index	253

Archived

Notices

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

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

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

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

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

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

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

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

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


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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which 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

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

Redbooks (logo) ™

eServer™

ibm.com®

z/Architecture™

z/OS®

zSeries®

Candle®

CICS®

DB2 Universal Database™

DB2®

FICON®

Hiperspace™

IBM®

IMS™

IMS/ESA®

Language Environment®

MQSeries®

MVS™

OMEGAMON®

OS/390®

Parallel Sysplex®

Redbooks™

RAA®

RACF®

RMF™

Tivoli Enterprise™

Tivoli®

VTAM®

WebSphere®

The following terms are trademarks of other companies:

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

Java, JDBC, JDK, JRE, JSP, JVM, J2EE, RSM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. 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® Redbook provides IMS™ performance monitoring and tuning information. This book differs from previous IMS performance and tuning redbooks in that there is less emphasis on the internal workings of IMS and more information about why and how certain options might affect the performance of IMS.

Most of the information in the previous IBM Redbook *IMS Version 7 Performance Monitoring and Tuning Update*, SG24-6404, is still valid, and in most cases, continues to be valid in any future versions of IMS. This book is not an update or rewrite but instead attempts to be more of a guide than a reference. As such, the team gathered experiences and data from actual production environments as well as from IBM benchmarks and solicited input from experts in as many areas as possible.

You should be able to find some valuable new information and perhaps validate some things you might have questioned. Hardware and software characteristics are constantly changing but hopefully the information you find here provides a basis to help you react to that change and keep your IMS running efficiently.

In this IBM Redbook, we introduce methods and tools for monitoring and tuning IMS systems and in addition to IMS TM and DB system-wide performance considerations, we dedicate separate chapters for application considerations, IMS and DB2® interoperability, the Parallel Sysplex® environment, and On Demand considerations.

The team that wrote this redbook

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

Jouko Jäntti is a Senior IT Specialist at IBM Global Services in Finland and also works for the Silicon Valley Laboratory as a member of IMS Worldwide Advocate Team. From 2001 to 2003, he was a Project Leader specializing in IMS with the IBM International Technical Support Organization, San Jose Center. He is the lead author of the IMS-related IBM Redbooks™ listed in “IBM Redbooks” on page 249.

David Matthews is a Senior IMS Systems Programmer for IBM Global Technology Services in Australia. He has over 23 years of IMS experience, including application programming, database administration, and systems programming. He provides technical support and advice for several major clients in the telecommunications, financial, and airline industries. His areas of expertise and responsibilities include IMS installation and maintenance, IMS system management, IMS problem determination, IMS Tools, and IMS-related products.

Jayesh Prag is a Senior Systems Architect for FirstRand Bank Limited in South Africa. He has 23 years of experience in the financial industry. His areas of expertise include programming in assembler and Cobol, specializing in IMS TM/DM, and providing guidance about performance tuning and benchmarking of all mainframe banking applications. He gives direction and guidance within FirstRand Bank to ensure that banking solutions are cost-effective and superior with regard to continuous availability and real-time banking requirements. He has taught many IMS classes about IMS fundamentals and IMS application design.

Dave Viguers is a member of the IBM IMS Development Team at Silicon Valley Lab working with performance, test, development, and client support. Dave also develops and delivers education on IMS Parallel Sysplex. He has spent over 30 years working with IMS in various functions and organizations.

Yuan Yi is an Advisory IT Specialist at IBM Global Technology Services in China. She has supported the IMS system for one of the biggest banks in China for about five years. She was in charge of IMS/DBCTL system management for all client projects, which includes installation, maintenance, IMSplex setup, migration, and performance monitoring. She also helped the client set up the backup and recovery procedure and was involved in disaster recovery planning.

Pete Ziegenfelder is a Senior Database Administrator for EDS in the United States specializing in DB2 and IMS TM/DM. He has 28 years of experience in the automotive industry. He holds a Bachelor of Science degree in Computer Information Systems and a Associate of Science degree in Data Processing from Lawrence Technological University. His areas of expertise include DB2 and IMS TM/DM performance and tuning, database recovery, high volume transactions, and resolving problems with very large DB2 tables and IMS databases. He gives direction and guidance to DBA teams around the world and has authored the IMS programmer and IMS DBA reference material used by EDS organizations worldwide. He has taught classes in a variety of DBMS topics to both application programmers and database administrators.



Figure 1 The team: Yuan Yi, JJ, Dave, Ziggie, Matt, and Jay

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

Paolo Bruni
Emma Jacobs
Leslie Parham

Deanna Polm
Sangam Racherla
International Technical Support Organization, San Jose Center

Rich Conway
Bob Haimowitz
International Technical Support Organization, Poughkeepsie Center

Rose Levin
John Butterweck
Jenny Choi
Jeff Fontaine
Mike Gonzales
John Harper
Jeff Maddix
Ernie Marek
Hiram Neal
Bovorit Pibulsonggram
Peggy Rader
Frank Ricchio
Pat Schroeck
Alan Smith
Greg Vance
Vern Watts
Gary Wicks
IBM Silicon Valley Laboratory, San Jose, USA

Rich Lewis
IBM zSeries® Software - IMS, Advanced Technical Support, Americas

John Maher
IBM Software Group, Tivoli®, OMEGAMON® XE for IMS, IBM USA

Mary Innes
IBM Australia

Special thanks to Pete Sadler, an Independent Consultant in UK, for providing a quick but still an extensive review despite such short notice.

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an e-mail to:

redbooks@us.ibm.com

- Mail your comments to:

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

Defining the performance problem in an IMS environment

Performance problems are erratic in nature and generally occur when you least expect them. This applies to any kind of a system, but in this book, we discuss the performance of IMS systems. In an environment where we have a combination of both internal and external variables, what must we do to apply good practices in ensuring that we become proactive? A number of techniques are available, but the most important is the ability to monitor, profile, track, and trend transactions.

This chapter discusses defining the problem and looks at the life cycle of an IMS transaction. We provide a detailed view of a message from its inception at a terminal through to its processing as an application and finally its reply back to the terminal. The purpose of this chapter is to highlight areas where performance problems can occur and give you a reference point in identifying and resolving performance problems.

In this chapter, we provide a brief overview of performance and then discuss full function, Fast Path, and DBCTL message flows. The message flow depicts the life of a transaction through IMS and highlights areas where performance problems can occur.

This chapter contains the following:

- ▶ Performance overview
- ▶ Understanding and defining the problem
- ▶ Events for full function messages
- ▶ Events for Fast Path messages
- ▶ Events for DBCTL
- ▶ Common log records produced with transaction flows
- ▶ Problem identification checklist

This chapter does not cover any Parallel Sysplex data sharing constraints or shared message queue performance issues. They are covered in Chapter 11, “IMS Parallel Sysplex considerations” on page 201.

1.1 Performance overview

The performance of an IMS system is directly related to a number of internal variables. These variables can be found in the z/OS® operating system, in IMS/TM, in IMS/DM, in the application, or in the hardware. External variables include the network and the physical infrastructure of your private network. These external variables are mostly out of our control, although they are integral in ensuring respectable response times. An understanding of your network architecture is paramount in diagnosing network-related response time issues.

IMS is an event driven system and events are represented internally by event control blocks (ECB). Each event or a multitude of events could result in performance bottlenecks. We attempt to provide a generic manner by which to monitor and view the IMS system in order to identify problems as events occur.

1.2 Understanding the performance problem

The z/OS operating system components together with IMS lead you into an information highway where you become totally enveloped in all its behaviors. This results in a degree of frustration if you are unable to determine what is causing your performance bottlenecks. Let us begin by defining the problem and then looking at some of the principles that need to be in place.

Definition: A performance problem is generally noted as bad or erratic response times or an unacceptable amount of resource usage.

Events that could trigger a performance problem are generally indicated by:

- ▶ Service level objectives not being met
- ▶ Users complaining about slow response
- ▶ Unexpected changes in response times or resource utilizations
- ▶ z/OS operating system showing signs of stress
- ▶ The throughput on the system is erratic
- ▶ Changes in workload which were not anticipated
- ▶ Changes in the profile of transactions

The problem we face is how do we differentiate among the various problems and what needs to be in place to statistically compare workloads in order to isolate the performance bottleneck.

1.2.1 Defining a service level agreement

Performance objectives must be defined as part of an service level agreement (SLA) with the relevant business unit. The SLA must define the following:

- ▶ Acceptable response times to the business
- ▶ Expected current volumes of transactions
- ▶ Growth strategy and anticipated future volumes
- ▶ Details of transactions and their usage

It must not come as a surprise that we need the ability to track and trend transactions across all business units throughout the organization. This tracking and trending requires the use of sophisticated tools and a repository to manage each transaction with its profile. Depending on what we have agreed to in our SLA, we might need to collect, summarize, and store statistics

and profiles on all transactions 24 hours a day, 365 days a year in a repository. There are many ways to create a repository and many products that provide you with a view to access this repository. These tools must have the capability to perform statistical analysis of the data stored in the repository.

1.2.2 Transaction profiles

The unit of work by which we measure IMS workload is a *transaction*. An IMS transaction is a message from either a terminal or an application program that causes application program logic to be executed.

A transaction profile typically covers the following:

- ▶ Host response times:
 - Input queue time measurement
 - Total elapsed time measurement
- ▶ The CPU time required to process the transaction
- ▶ The number of database (DL/I and SQL) calls performed by the transaction
- ▶ The type of database calls performed:
 - By database or table listing each database or table and the type of call
- ▶ Number of I/Os required to perform this transaction

A typical transaction generally performs many different types of functions. A transaction designated to process internet banking can perform transfers and payments as well as balance inquiries. The profile by transaction is different for each of these functions. Analysis must be done to quantify these profiles to percentiles of the various functions of a transaction in order to make meaningful decisions.

This implies that making changes to a very low percentile-used function would not be beneficial, when compared to the total amount of CPU consumed and DL/I calls made. Cost as opposed to effort becomes a factor in most decisions.

When you have a transaction profiling repository with daily feeds into the repository from the IMS log records, other performance log records generated by vendor products, and SMF records, analysis and interpretation of the data become crucial issues.

1.2.3 Analysis and interpretation

As we introduce new functionality to applications, the transaction profile varies over time and the day of the week. Certain functions might only be used heavily during certain times of the month. When analyzing data from the repository, you need to consider the following:

- ▶ Am I comparing profiles of similar days?
- ▶ Has there been a sudden shift in the profile as a result of introducing new functionality?
- ▶ Has my I/O profile changed?
- ▶ Have the volumes increased?

This and many more questions need to be answered before analyzing data and comparing data to similar historical periods.

This leads us into the next section, which requires that we track and trend our profiles to ensure that we remain proactive in identifying and resolving problems before they occur.

1.2.4 Tracking and trending

As time progresses, so does technology. We need to ensure that we manage expectations and contain cost per transaction within reasonable limits. This containment of costs can only be achieved if we track and trend our workload to understand future capacity requirements of all workloads. Our recommendation is that full time tracking and trending become the responsibility of a performance expert with the ability to influence the manner in which business areas operate.

Running out of capacity as a result of being unable to project increased workloads can prove to be disastrous to any organization. There are many tools available to aid in capacity planning. What needs to be achieved is the ability to forecast future workload increases on your current capacity. This forecast is based on the ability to track and trend workloads as well as future business requirements.

1.3 Events for full function messages

This section provide us with a means to identify possible performance problems. We explore both full function and Fast Path transaction flows with a view to identifying performance problems as we flow through the various events. Table 1-1 shows the list of events for full function messages.

Table 1-1 Event list for full function messages

Event	Activity	Problem identification
Message arrives into IMS	Message format services (MFS) routines, basic edit, or intersystem communication (ISC) edit	RECA, High I/O Pool (HIOP), CIOP, and MFBP pool shortages evident
Message queuing	Message queued to scheduler message block (SMB)	QBUF shortage evident
Message scheduling	Schedule message in dependent region	Transaction queuing or pool intent failures
Scheduling-end to first DL/I call	Load programs, subroutines, and initialize working storage	Shows up as high CPU and elapsed time
Program elapsed time	Application program invoked and DL/I calls performed	High elapsed times as a result of application, database, WLM, I/O subsystem, or system issues
Sync point	Phase one and two commit	High elapsed times as a result of WADS, OLDS, or I/O subsystem delays
Message output	Send output to destination	HIOP pool shortage, or network delays before message is dequeued

1.3.1 Message arrives into IMS

The message arrival event begins when the message is placed by either VTAM® in an IMS receive-any (RECANY) buffer or transported through IMS Connect to IMS Open Transaction Manager Access (OTMA). Either way, the message is moved to a buffer acquired in the High

I/O Pool (HIOP) where it is edited by message format services (MFS) routines, IMS basic edit, or intersystem communication (ISC) edit.

1.3.2 Message queuing

Message queuing events require the message in the standard IMS format *llzz|trancode|data*. It is allocated a position on one of the message queue data sets. The message is moved to the message queue pool and enqueued on the scheduler message block (SMB).

1.3.3 Message scheduling

Message scheduling events are dependent on a number of variables. The transaction definition is the starting point for any scheduling issues. Variables that affect transaction scheduling are:

- ▶ SCHDTYP on APPLCTN macro
- ▶ Serial or parallel
- ▶ PROCLIM
- ▶ PARLIM
- ▶ PRIORITY
- ▶ MAXRGN
- ▶ CLASS of transaction
- ▶ SCHED

The next step is to identify any pool intent failures. IMS requires the program specification block (PSB) directory (PDIR) and data management block (DMB) directory (DDIR) for the scheduled PSB to be available. The PSB and DMB are loaded when the transaction first executes. If not resident, then ACBLIB I/Os are required to load the PSBs and DMBs. The PSB, PSB work pool (PSBW), and DMB pools must be large enough to accommodate all PSBs and DMBs, if possible, and must be monitored regularly.

1.3.4 Scheduling-end to first DL/I call

Scheduling-end to first DL/I events include program or subroutine load and any program and working storage initialization being performed by the application. You need to review long time periods when evident in this event from a program load or application design perspective.

1.3.5 Program elapsed time

Program elapsed time events are composed of many variables that can influence the elapsed time of a transaction. *Elapsed time* is measured from the time that the application is scheduled into the IMS regions until synchronization point (sync point) processing. Variables that can affect elapsed time are:

- ▶ Program execution time
- ▶ Time required to complete the DL/I calls and this is made up of two components:
 - IWAIT time, which is the time DL/I has to acquire the database record
 - NOT-IWAIT time, which is the time DL/I spends in code execution
- ▶ I/O subsystem delays as a result of DASD response times
- ▶ System waits, over which IMS has no influence

1.3.6 Sync point processing

Sync point processing events for full function (FF) require that all I/Os actually take place. IMS follows the standard two-phase commit process. In an IMS only workload, this two-phase commit is actually a call to two separate modules. Generally, a two-phase commit is between IMS and DB2 and follows the standard two-phase commit process. In IMS terms, a get unique (GU) call to the IOPCB to retrieve the next message implies sync point.

1.3.7 Message output

Message output event implies termination of the program so that output messages are ready to be sent to their final destination. The HIOP pool usage is important in identifying possible performance-related issues in this area.

1.4 Events for Fast Path messages

Table 1-2 is specific to messages being processed through Fast Path expedited message handling (EMH). Certain overlaps exist between Fast Path and full function and are mentioned below.

Table 1-2 Event list for Fast Path messages

Event	Activity	Problem identification
Message arrives into IMS	MFS, basic edit or ISC edit	RECA, MFBBP, HIOP, or CIOP pool shortages evident
Fast Path expedited message handling (EMH) queuing	Determine if Fast Path (FP) potential or exclusive	EMHB pool shortages
Fast Path EMH scheduling	First-in-first-out scheduling by balancing group (BALG)	Queuing on BALG
Program elapsed time	Application program invoked and DL/I calls performed	High elapsed times as a result of application, database, WLM, I/O subsystem, or system issues
Sync point	Phase one and two commit	Output thread shortage (OTHR)
Message output	Send output to destination	EMHB pool shortage, or network delays before message is dequeued

1.4.1 Message arrives into IMS

Message arrival event for Fast Path is essentially the same as for full function.

1.4.2 Fast Path EMH queuing

Fast Path EMH event requires IMS to make a decision on whether the message is Fast Path potential (FPP) or Fast Path exclusive (FPE). If FPP, then DBFHAGU0 is called to decide whether the message needs to be processed by full function IMS or whether a routing code needs to be assigned to make it Fast Path. If FPE, then IMS queues the message off the BALG. The BALG becomes the queue anchor point.

1.4.3 Fast Path EMH scheduling

Fast Path EMH scheduling event has no complex priority scheduling schema when compared to FF. Messages are processed using a BALG on a first-in-first-out (FIFO) basis and scheduled into an available IMS Fast Path (IFP) region. All IFP regions are always in wait-for-input (WFI) mode so the overhead of program load is avoided.

1.4.4 Program elapsed time

Program elapsed time event involves the same principle as mentioned for FF, with the exception that any Fast Path DEDB or MSDB reads are performed under the control of the control region instead of the DL/I separate address space.

1.4.5 Sync point processing

Fast Path sync point processing is significantly different from full function sync point processing. A key element of Fast Path sync point processing is that all Fast Path writes happen asynchronously through output threads (OTHR). Two-phase commit process is still used by Fast Path, but it operates differently. IMS Fast Path does not have to wait for the I/O to be physically written. If phase one processing is unsuccessful, the buffers are thrown away, and the transaction is either rescheduled or thrown away in its entirety, depending on the nature of the problem.

1.5 Events for DBCTL

Table 1-3 is specific to messages being processed via DBCTL.

Table 1-3 Event list for DBCTL

Event	Activity	Problem identification
Message arrives into CICS® TOR/AOR.	Format message and call/link related application programs.	
Application gets control and issues schedule request.	EXEC DL/I command.	
PSB scheduling.	DBT thread TCB is given control. Recovery token establishes.	Insufficient PSB, DMB, DB work, PSB work, or EPCB pool space can cause scheduling delays or failures
Program elapsed time.	Application program invoked and DL/I calls performed.	High elapsed times as a result of application, database, WLM, I/O subsystem, or system issues.
Sync point.	IMS hardens the log data and writes updated full function database blocks. Fast Path database updates are asynchronous.	WADS and OLDS activity. Database DASD I/O.
Application issues terminate PSB call.	IMS frees scheduled resources.	

1.5.1 Message arrives into CICS TOR/AOR

Message arrives at CICS TOR first and unless also acting as an AOR, the message can be routed to a CICS AOR.

1.5.2 Application gets control and issues schedule request

Application issues DL/I SCHED call to reserve IMS resources.

1.5.3 PSB scheduling

In most cases, one PSB is scheduled for one CICS transaction. IMS checks if the PSB and related databases are available. It then allocates space for the necessary control blocks in various scheduling pools and loads the required blocks.

1.5.4 Program elapsed time

Program elapsed time event is composed of many variables that can influence the elapsed time of a transaction. Elapsed time is measured from the time the PSB is scheduled until the PSB is terminated by the application. Variables that can affect elapsed time are:

- ▶ Program execution time
- ▶ Time required to complete the DL/I calls and is made up of two components:
 - IWAIT time, which is the time DL/I has to acquire the database record
 - NOT-IWAIT time, which is the time DL/I spends in code execution
- ▶ I/O subsystem delays as a result of DASD response times
- ▶ System waits, which IMS has no influence over

1.5.5 Sync point processing

IMS hardens the log data and writes updated full function database blocks. Fast Path writes are performed asynchronously to sync point processing.

1.5.6 Application issues terminate PSB call

PSB terminates and scheduling resources are freed.

1.6 Common log records produced for transaction flows

Table 1-4 on page 9 provides a list of the most commonly used log records produced during the life cycle of an IMS transaction. Identifying these log records provides the ability to diagnose performance-related problems.

Table 1-4 Common log records produced by both full function and Fast Path

Record	Description
X'01'	Message received from a terminal.
X'03'	Message received from DL/I.
X'07'	An application program was terminated.
X'08'	An application program was scheduled.
X'31'	Message queue GU.
X'32'	Message queue reject.
X'33'	Message queue free.
X'34'	Message cancel.
X'35'	Message queue enqueue.
X'36'	Message queue dequeue.
X'37'	Sync point record.
X'38'	Message after abend.
X'50'	Database undo/redo record.
X'5901'	Fast Path input.
X'5903'	Fast Path output.
X'5936'	Fast Path dequeue.
X'5937'	Fast Path sync point.
X'5938'	Fast Path abend.
X'5950'	Fast Path phase1 sync record.
X'5953'	Fast Path sequential dependent (SDEP) write.

1.7 Problem identification matrix

Table 1-5 on page 10 provides a view through the life cycle of an IMS transaction and is useful for isolating specific events where performance problems occur.

Table 1-5 Starting point in looking for performance bottlenecks

Problem	What might be going on	Chapter
Erratic transaction arrival rates evident with bad response times	Possible RECA, HIOP buffer pool shortages, or IMS has gone into selective dispatching.	See Chapter 6, "Transaction manager performance" on page 125.
Transactions queuing	Possible QBUF shortage, message queue I/O bottlenecks, scheduling problems, or DMB/PSB pool failures.	See Chapter 6, "Transaction manager performance" on page 125.
Schedule to first DL/I call time is high	Possible program load, program initialization problem.	See Chapter 6, "Transaction manager performance" on page 125 and Chapter 4, "IMS and Workload Manager" on page 45.
High elapsed times	Program execution time. DL/I IWAIT or NOT-IWAIT time. System waits.	See Chapter 4, "IMS and Workload Manager" on page 45, Figure 5 on page 53, Chapter 6, "Transaction manager performance" on page 125, and Chapter 9, "Application considerations" on page 169.
Long waits for sync point	I/O subsystem delays.	See Chapter 4, "IMS and Workload Manager" on page 45.
Output message delays	Possible HIOP or network delays.	See Chapter 6, "Transaction manager performance" on page 125.

Monitoring methodology

This chapter discusses monitoring methodology. *Monitoring* is the collection and interpretation of IMS data. Monitoring should be an ongoing task because:

- ▶ Monitoring helps you establish base profiles, workload statistics, and data for capacity planning and prediction.
- ▶ Monitoring gives early warning and comparative data to help you prevent performance problems.
- ▶ Monitoring validates tuning you have done in response to a performance problem and ascertains the effectiveness of that tuning.

An historical base and conclusions from continuous monitoring provide a good start to answering user complaints and an initial direction for tuning projects.

In this chapter, we describe the following topics:

- ▶ Establishing monitoring strategies
- ▶ Monitoring multiple systems in DB/DC and DCCTL environments
- ▶ Coordinating performance information in an MSC Network
- ▶ Monitoring Fast Path systems in DB/DC and DCCTL environments
- ▶ Transaction flow in DB/DC and DCCTL environments
- ▶ The IMS Monitor in DB/DC and DCCTL environments
- ▶ Monitoring procedures in a DBCTL environment

2.1 Establishing monitoring strategies

Several types of monitoring strategies are available. You can:

- ▶ Summarize actual workload for the entire online execution. This can include both continuous and periodic tracking. You can track total workload or selected representative transactions.
- ▶ Take sample snapshots at peak loads and under normal conditions. It is always useful to monitor the peak periods for two reasons:
 - Bottlenecks and response time problems are more pronounced at peak volumes.
 - The current peak load is a good indicator of what the future average will be like.
- ▶ Monitor critical transactions or programs that have documented performance criteria.
- ▶ Use the z/OS Workload Manager to help manage workload distribution, balance workloads, and distribute resources.

Plan your monitoring procedures in advance. A strategy should explain the tools to be used, the analysis techniques to be used, the operational extent of those activities, and how often they are to be performed.

Regardless of which strategy you use, you need to:

- ▶ Develop performance criteria
- ▶ Develop a master plan for monitoring, data gathering, and analysis

2.2 Monitoring multiple systems in DB/DC and DCCTL environments

You should plan to obtain both statistical and performance data for IMS online systems that are part of a multi-system network. You can use the same monitoring tools that are used for generating performance data for single IMS systems.

The IMS Monitor can be executed concurrently in several systems. You obtain IMS Monitor reports for each IMS system and coordinate your processing analysis:

- ▶ The IMS Statistical Analysis utility produces summaries of transaction traffic for each system. Again, you combine the statistics for a composite picture.
- ▶ The IMS Transaction Analysis utility enables you to trace transactions across multiple systems and examine the traffic using various active physical links.

2.3 Coordinating performance information in an MSC network

The IMS system log for each system participating in Multiple Systems Coupling (MSC) contains only the record of events that take place in that system. The logging of traffic received on links is included. You can augment the system log documentation that records the checkpoint intervals with the system identifications of all coupled systems. This helps you interpret reports, because you know of transactions that might be present in message queues but are not processed, and you can expect additional transaction loads from remote sources. In your analysis procedures, include ways of isolating the processing triggered by transactions originating from another system.

To satisfy the need for monitoring with typical activity that includes cross-system processing, coordinate your scheduling of the IMS Monitor and other traces between master terminal operators. The span of the monitoring does not have to be exactly the same, but if it is widely different, the averaging of report summaries can make it harder to interpret the effect of the processing triggered by cross-system messages.

Related Reading: For more information about interpreting MSC reports, see *IMS Version 9: Utilities Reference: Database and Transaction Manager*, SC18-7833.

2.4 Monitoring Fast Path systems in DB/DC and DCCTL environments

The major emphasis for monitoring IMS online systems that include message-driven Fast Path application programs is the balance between rapid response and high transaction rates. With Fast Path, performance data is made part of the system log information. Also, the IMS Monitor can be used to monitor Fast Path activities. You can use the IMS Fast Path Log Analysis utility to generate statistical reports from the system log records. This utility can provide summaries of the Fast Path transaction loads, reports that highlight exceptional response time, and a breakdown of the elapsed time between key events during the time in the system.

The system administration tasks of setting up a monitoring strategy, performance profiles, and analysis procedures should be carried into the Fast Path environment.

Related Reading: For more information about using either the IMS Monitor or the IMS Fast Path Log Analysis utility, see *IMS Version 9: Utilities Reference: System*, SC18-7834.

2.5 Transaction flow in DB/DC and DCCTL environments

A distinct sequence of events occurs during the processing of a transaction. Message-related processing is asynchronous within IMS, that is, not associated with a dependent region's processing. Examples of this kind of processing are message traffic, editing, formatting, and recovery-related message enqueueing, any of which can be done concurrently with application program processing for other transactions. Events from application program scheduling to termination are associated with a PST and can be regarded as synchronous.

Figure 2-1 on page 14 shows you a sequence of events to give you an overall picture of the activity that takes place when an online IMS system is processing a mix of transactions concurrently. Each event is explained in the notes that follow.

The unit of work by which most IMS systems are measured is the transaction (or a single conversation iteration, from entering the input message to receipt of one or more output messages in response). One way of representing the flow of units of work is to compare it to three funnels through which all transactions must pass, as illustrated in Figure 2-1 on page 14. The events that account for the principal contributions to transaction response time are numbered in the center. The items entered on the left of the diagram are message-related, and those on the right are related to the application program. The arrows trace the flow for an individual transaction. (The diagram does not show the paging element or system checkpoint processing that is distributed through the elapsed times.)

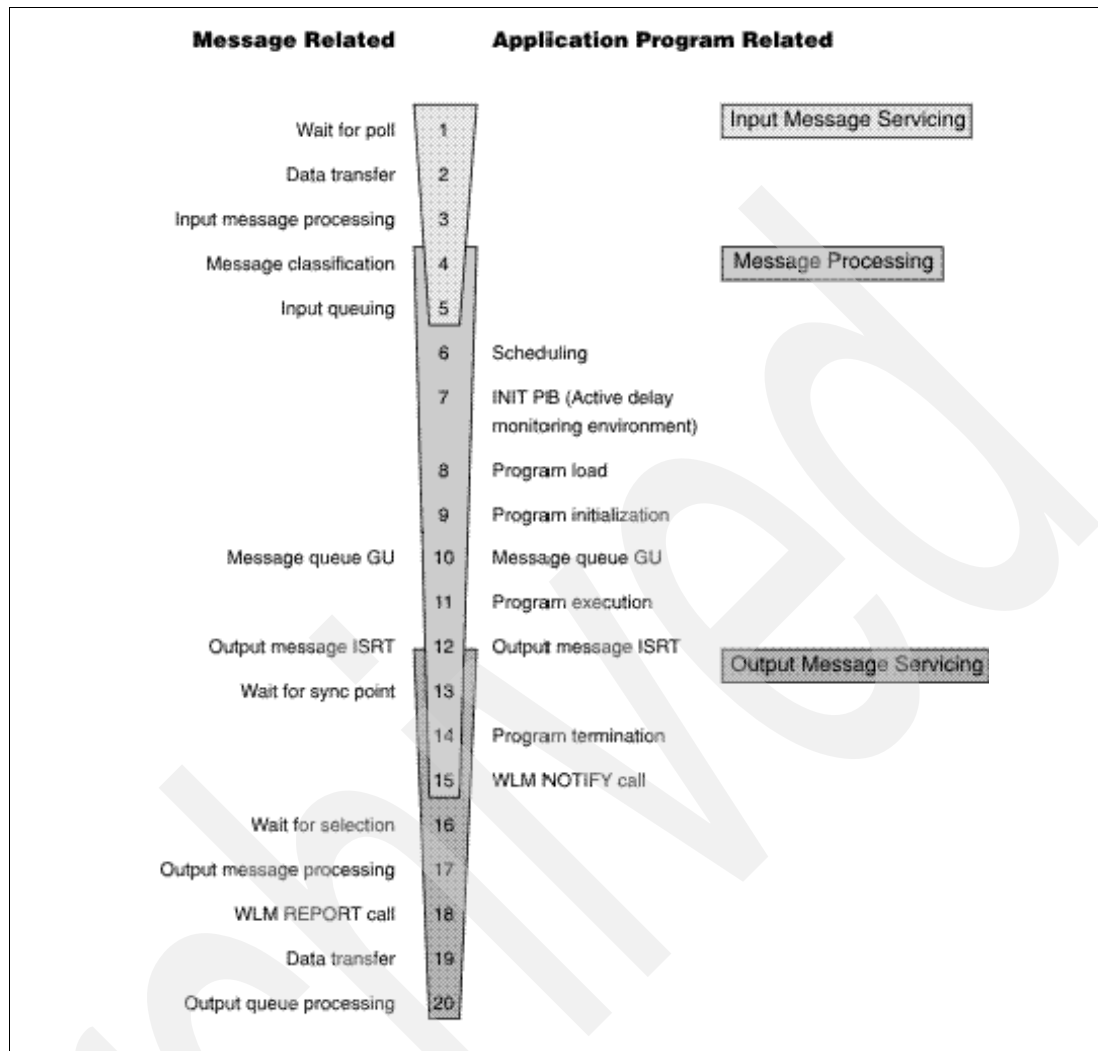


Figure 2-1 Processing events during transaction flow through IMS

Notes to Figure 2-1:

1. Wait for poll

This used to be the time between pressing the Enter key and receiving a poll that results in the data being read by the channel program when using BTAM. Now this time does not belong to IMS.

2. Data transfer

This time includes propagation delay and modem turnarounds for multi-block input messages. You can estimate the data transfer times if the volume of data transmitted is known.

3. Input message processing

The IMS control of the transaction begins when the input message is available in the HIOP. The time that the message spends in this pool, in MFS processing, and in being moved to the message queue buffers affects response time. Individual transaction I/O to the Format library affects the message queue. A major factor in determining response time is whether the respective pools are large enough for the current volume of transactions flowing into input queuing. In particular, if the message queue pool is too small, overflow to the message queue data sets occurs.

4. Message classification

This is the call to the z/OS WLM to obtain a WLM service classification for the incoming message.

5. Input queuing

This is the time spent on the input queue or in the message queue buffers waiting for a message region to become available. In a busy system, this time can become a major portion of the response time. The pattern of programs scheduled into available regions and the region occupancy percentage are important and should be closely monitored.

6. Scheduling

Because of class scheduling, regions can be idle while transactions are still on the queue. The effects of scheduling parameters can be:

- Termination of scheduling as a result of PSB conflict or message class priorities
- Termination of scheduling as a result of intent conflict
- Extension of scheduling by I/Os to IMS.ACBLIB for intent lists, PSBs, or DMBs
- Pool space failures in either the PSB or DMB pools

7. Init PB call (activate delay monitoring environment)

Activate the WLM delay monitoring environment for the message when it is placed into the dependent region. The WLM PB is initialized with the Service Classification and transaction name, message arrival time, program execution start time (current time), user ID, and so forth.

8. Program load

See event 9.

9. Program initialization

After scheduling, several kinds of processing events occur before the application program can start:

- Contents supervision for the dependent region
- Location of program libraries and directories to them
- Program fetch from the program library
- Program initialization up to the time of the first DL/I call to the message queue

For monitoring, you can obtain the overall time for the above activities. The number of I/Os should be checked periodically.

10. Message queue GU

This is the GU call to the message queue. It is chosen as a measuring point because the event is recorded on the system log and is used as a starting point for iterations of processing when more than one message is serviced at a single scheduling of the program.

11. Program execution

The time for program execution, from first message call to the output message insert, is a basic statistic for each transaction. It is important to account for that time in terms of the work performed:

- The number of transactions processed per schedule
- The number and type of DL/I calls per transaction
- The number of I/Os per transaction

A useful breakdown of elapsed time into processor time and I/O helps determine which transactions use significant resource.

12. Output message insert

This time begins the asynchronous processing for the output response. The output message requests flow into the funnel to be serviced while the application program is either beginning to process another input message or is performing closeout processing and program termination.

13. Wait for sync point

When an output message is issued by a program, it is enqueued on a temporary destination until the program reaches a synchronization point. For programs specified as MODE=MULT, a long delay in output transmission can occur when the program goes on to process several transactions at one scheduling. None of the previous output messages can be released for transmission until the program ends. If the program fails, all current transactions are backed out. Actually, when the messages are dequeued, LIFO sequence is used, from temporary to permanent destination. With MODE=SNGL, the wait for sync point (at the next GU to the message queue) is normally negligible.

14. Program termination

In the case of MODE=MULT, discussed in event 13, the synchronization point occurs at program termination. Any database updates are purged from the database buffer pools, and the waiting output messages are released.

In the MODE=SNGL case, the synchronization point occurs at the previous message queue GU call (usually a GU with a QC status code), and no database commit processing occurs at termination, unless the application program has updated a database after the last message queue GU.

15. WLM notify call

This tells WLM that the application program has ended execution. The PB and current time are passed to WLM.

16. Wait for selection

This time is similar to Wait for Poll on input, with one difference, which is an output message might not have to wait for the completion of a polling cycle if no polling is in progress on the line at the time the output message is enqueued. However, there might be a wait for the duration of data transmission to other terminals on the line. In a busy system, this could account for the majority of time spent on the output queue.

17. Output message processing

This activity is similar to event 3.

18. WLM report call

This tells WLM the response is being sent. IMS passes the input message arrival time, the Service Classification, and the current time (output message send time).

19. Data transfer

This activity is similar to event 2.

20. Output queue processing

Output messages that have been sent are dequeued after their receipt has been acknowledged by the terminal. In the case of paged output, the acknowledgment is a consequence of another input or a PA2 entry from the terminal.

2.6 The IMS Monitor in DB/DC and DCCTL environments

The principal monitoring tool provided by IMS is the IMS Monitor. It is an integral part of the control program in the DB/DC environment. The counterpart of the IMS Monitor in the batch environment is the Database Batch Monitor.

The IMS Monitor collects data while the DB/DC environment is operating. Information in the form of monitor records is gathered for all dispatch events and placed in a sequential data set. The IMS Monitor data set is specified on the IMSMON DD statement in the control region JCL; data is added to the data set when the **/TRACE** command activates the monitor. The MTO can start and stop the monitor, guided by awareness of the system's status, to obtain several snapshots.

Related Reading: For more information about interpreting IMS Monitor reports, see *IMS Version 9: Utilities Reference: System*, SC18-7834.

2.7 Monitoring procedures in a DBCTL environment

This topic explains how to establish monitoring procedures for your DBCTL environment. First, consider that monitoring in a DB/DC environment generally refers to the monitoring of transactions. The transaction is entered by an user on a terminal, is processed by the DB/DC environment, and returns a result to the user. Transaction characteristics that are measured include total response time and the number and duration of resource contentions.

A DBCTL environment has no transactions and no terminal users. It does, however, do work on behalf of CCTL transactions that are entered by CCTL terminal users. DBCTL monitoring provides data about the DBCTL processing that occurs when a CCTL transaction accesses databases in a DBCTL environment. This access is provided by the CCTL making the DRA request.

The most typical sequence of DRA requests that represents a CCTL transaction would be:

- ▶ A SCHED request to schedule a PSB in the DBCTL environment
- ▶ A DL/I request to make database calls
- ▶ A sync point request, COMMTERM, to commit the database updates and release the PSB

The DBCTL process that encompasses this request is called a unit of recovery (UOR).

DBCTL provides UOR monitoring data, such as:

- ▶ Total time the UOR exists
- ▶ Wait time to schedule a PSB
- ▶ I/O activity during database calls

This information is similar to, and is often the same as, DB/DC monitoring data. However, in a DBCTL environment, the UOR data represents only a part of the total processing of a CCTL transaction. You must also include CCTL monitoring data to get an overall view of the CCTL transaction performance.

In this topic, the term *transaction* refers to a CCTL transaction. When it applies, UOR is specifically named.

The CCTL administrator must decide what strategy to use to monitor transaction performance. Several types of monitoring strategies are available.

You can:

- ▶ Summarize actual workload for the entire online execution. This can be continuous or at an agreed-to frequency. Total workload or selected representative transactions can be tracked.
- ▶ Take sample snapshots at peak loads and under normal conditions. It is always useful to monitor the peak periods for two reasons:
 - Bottlenecks and response time problems are more pronounced at peak volumes.
 - The current peak load is a good indicator of what the future average will be like.
- ▶ Monitor critical transactions or programs that have documented performance criteria.

Monitoring tools

This chapter describes some of the most commonly used IMS monitoring tools that give a detailed picture of how IMS uses system resources, how to identify bottlenecks within IMS, and how z/OS performance problems affect overall IMS performance.

In this chapter, the following monitoring tools are described:

- ▶ IMS Monitor
- ▶ IMS Performance Analyzer
- ▶ File Select and Formatting Print utility
- ▶ Log Transaction Analysis utility
- ▶ Knowledge-Based Log Analysis
- ▶ IBM Tivoli OMEGAMON XE for IMS on z/OS
- ▶ IBM IMS Connect Extensions for z/OS
- ▶ IBM IMS Buffer Pool Analyzer for z/OS

3.1 IMS monitoring tools

The effective use of system resources, CPU cycles, real storage, and I/O devices is the goal of any IMS installation. Measurement tools that determine how the resources are used are necessary for any performance evaluation. Although important measurement tools are integrated into the z/OS system, your installation needs additional tools to collect all of the data required for IMS performance evaluation.

The program offerings most commonly used in IMS performance evaluation are the only ones discussed. Many more program offerings are available, and your installation can evaluate their usefulness. Table 3-1 lists the IMS tools that are required to perform an IMS tuning exercise. Measurement tools for an IMS system consist of the IMS Monitor, IMS Performance Analyzer (IMS PA), IMS Version 9 Knowledge-Based Log Analysis (KBLA), OMEGAMON, and IMS utilities that process the IMS log. Some of the tools are integrated into the IMS product, and others are program offerings. These tools give you a detailed picture of how IMS uses system resources. They can identify bottlenecks within IMS, and, when used in conjunction with the z/OS data, they show how z/OS performance problems affect overall IMS performance.

The measurement tools discussed below do not provide all the data necessary for a complete evaluation of current system performance. They do not provide information about how and under what conditions each resource is used, nor do they provide information about the existing system configuration while the data is being collected. This data, used in conjunction with the data produced by the measurement tools, provides the basic information that you must have to conduct a system performance evaluation. It is therefore extremely important to use as many techniques as possible to get information about the system.

Table 3-1 IMS monitoring tools

Tool	Type	Description	Documentation
IMS Monitor	IMS component	IMS Monitor traces IMS activities at the ITASK level of detail. An offline program generates reports describing IMS online characteristics. It is the primary tool for IMS tuning.	<i>IMS Version 9 Utilities Reference: Database Manager and Transaction Manager</i> , SC18-7833
File Select and Formatting Print utility (DFSERA10)	IMS component	File Select and Formatting Print utility formats and prints selected records from the IMS log.	<i>IMS Version 9 Utilities Reference: System</i> , SC18-7834
IMS Version 9 Knowledge-Based Log Analysis (KBLA)	IMS component	KBLA has an ISPF panel driven interface, which you can use to invoke existing log and trace utilities as well as several new programs, which provide more interpretative and user friendly information. KBLA can significantly reduce the need to reference the utilities manuals, and it can help you avoid JCL or control statement errors. It should also reduce or eliminate the need to cross reference with control block DSECTS.	<i>IMS Version 9 Utilities Reference: System</i> , SC18-7834, and <i>IMS Version 9 Implementation Guide - A Technical Overview</i> , SG24-6398

Tool	Type	Description	Documentation
IMS Performance Analyzer (IMS PA)	Program product - 5655-E15	IMS Performance Analyzer processes IMS log and monitor data to provide enhanced summary and detailed reports on system and program performance.	<i>IBM IMS Performance Analyzer for z/OS User's Guide</i> , SC18-9778
IBM Tivoli OMEGAMON XE for IMS on z/OS	Program product - 5698-A39	IBM Tivoli OMEGAMON XE for IMS on z/OS is a remote monitoring agent that resides on z/OS managed systems. It assists you in anticipating performance problems and warns you when critical events take place on your systems. With Tivoli OMEGAMON XE for IMS on z/OS, you can set threshold levels and flags as desired to alert you when the systems reach critical points.	<i>Using IBM Tivoli OMEGAMON XE for IMS on z/OS</i> , GC32-9351
IBM IMS Connect Extensions for z/OS	Program product - 5655-K48	IMS Connect Extensions provides a new level of control, enabling users to easily monitor, manage, tune, and secure IMS Connect for z/OS as it opens and closes TCP/IP sockets and routes transactions.	<i>IMS Connect Extensions for z/OS User's Guide</i> , SC18-7255
IBM IMS Buffer Pool Analyzer	Program product - 5697-H77	IMS Buffer Pool Analyzer provides statistical analysis that helps you determine the impact of changes to IMS online and batch job buffer pools.	<i>IMS Buffer Pool Analyzer for z/OS User's Guide</i> , SC18-7068

Your installation can evaluate the use of the other available tools, such as PI trace, PSB trace, and Fast Path trace and the many other available offerings in the IBM DB2 and IMS Tools portfolio.

3.2 IMS Monitor

The IMS Monitor, which is integrated into the IMS product, collects information about most dispatchable events in the IMS system, including Fast Path events starting with IMS Version 7. It is the primary tool for tuning the IMS system, analyzing application programs, validating database design, and monitoring system performance. It is also a means of inferring the effect on IMS of the interaction of the hardware and software components of the total system. The IMS Monitor collects its data while the online IMS system is in operation. The monitor can be started and stopped by means of IMS commands from the IMS master terminal. More details on the /TRACE command can be found in the *IMS Version 9 Operations Guide*, SC18-7830.

Table 3-2 on page 22 lists most of the report titles and the most important fields (by no means, all inclusive) from the DFSUTR20 reports.

Table 3-2 IMS Monitor reports

	Key report fields	Comments ^a
Buffer pool statistics		
Message queue pools report	Number of immediate fetch (I/F) I/Os Number of directory I/O operations Number of times blocks washed for FRE	TM
Database buffer pools report	Number of read-I/O requests Number of blocks written by purge Number of locate calls waited due to busy ID Number of locate calls waited due to buffer busy wrt Number of locate calls waited due to buffer busy read Total number of I/O errors for this subpool	DB and DBCTL
VSAM buffer pool report	Number of Times Ctrl Interval Requested Already in Pool Number of Ctrl Intervals Read from External Storage Number of VSAM Writes Initiated by IMS/ESA® Number of VSAM Writes to Make Space in the Pool	DB and DBCTL
Message format buffer pool report	Number of I/F I/Os Number of times blocks washed for FRE Quotient	TM
General reports		
Latch conflict statistics report	LOGL Contentions	TM, DB, and DBCTL
General IWAIT time events report	Mean IWAIT Time	TM, DB, and DBCTL
Region and jobname report	None	TM, DB, and DBCTL
Region summary		
Scheduling and termination section	Elapsed Time Mean Not IWAIT Time(Elapsed - IWAIT) Mean	TM, DB, and DBCTL
Schedule to first call section	Elapsed Time Mean	TM and DBCTL
Elapsed execution section	Elapsed Time Mean	TM, DB, and DBCTL
DL/I calls section	Elapsed Time Mean Not IWAIT Time(Elapsed - IWAIT) Mean IWT/Call	TM, DB, and DBCTL
Idle for intent section	Elapsed Time Mean	TM, DB, and DBCTL
Checkpoint section	Elapsed Time Mean Not IWAIT Time(Elapsed - IWAIT) Mean	TM, DB, and DBCTL

Region occupancy section	Percentage	TM, DB, and DBCTL
Region IWAIT		
		Report is one per region.
Scheduling and termination section	IWAIT Time Mean Function	TM, DB, and DBCTL
DL/I calls section	IWAIT Time Mean Function Module	TM, DB, and DBCTL
Checkpoint	IWAIT Time Mean Function	TM, DB, and DBCTL
Programs by region		
		Report is one per region.
	Elapsed Execution Time Mean	TM, DB, and DBCTL
	Scheduling End to First Call Time Mean	TM, DB, and DBCTL
Program summary		
	Calls/Tran I/O IWAITS / Call CPU Time / Sched Elapsed Time / Sched Sched to 1st Call / Sched Elapsed Time / Trans	TM, DB, and DBCTL
Program I/O		
		Report is one per PSB.
	IWAITS IWAIT Time Mean DDN/Func	TM, DB, and DBCTL
Communication summary		
	Elapsed Time Mean Not IWAIT Time Mean	TM
Communication IWAIT		
	IWAIT Time Total IWAIT Time Mean Blksize	TM
Line functions		

	Mean Receive Blksize Max Receive Blksize Mean Trans Blksize Max Trans Blksize Mean Interval Max Interval	TM
Transaction queuing		
	Number Dequeued On Queue When Scheduled Mean Dequeued Mean	TM
Reports		
	Intent Failure Summary Pool Space Failure Summary Deadlock Event Summary Monitor Overhead Data	TM, DB, and DBCTL
Run profile		
	Number of Transactions Per Second Number of DI/I Calls Per Transaction Number of Message Queue Pool I/Os Number of Format Buffer Pool I/Os Ratio of Program Elapsed to DL/I Elapsed Ratio of DL/I Elapsed to DL/I IWAIT	TM, DB, and DBCTL
Call summary		
		Report is one per PSB.
	IWAITs/Call Elapsed Time Mean Not IWAIT Time Mean	TM, DB, and DBCTL

- a. Reports marked TM apply to IMS Monitor data.
 Reports marked DB apply to IMS DB Monitor data.
 Reports marked DBCTL apply to DBCTL users.

Example 3-1 shows one VSAM buffer pool statistics. Each VSAM buffer pool defined in DFSVSMxx member has its report section. The most important indicator for buffer pool performance is “VSAM Wrts to make space.” It should always be 0. “Buffer Pool Read I/O Rate” is divided by transaction rate, which helps you to understand which pool is most affected or needs adjusting. “Hit Ratio” is not always an accurate indicator for random access. For additional reference, see *IMS Version 9 Utilities Reference: Database Manager and Transaction Manager*, SC18-7833. The IBM IMS and DB2 Tools portfolio includes a product, IMS Buffer Pool Analyzer (program number 5697-H77), that might be helpful in this analysis.

Example 3-1 IMS Monitor VSAM Buffer Pool report

IMS MONITOR	*** BUFFER POOL STATISTICS ***	TRACE START 2006 250, 18:39:19	TRACE STOP 2006 250, 18:49:12	PAGE 0013
VSAM BUFFER POOL				
FIX INDEX/BLOCK/DATA				N/N/N
SHARED RESOURCE POOL ID				XXXX
SHARED RESOURCE POOL TYPE				D
SUBPOOL ID				5
SUBPOOL BUFFER SIZE				8192
NUMBER HIPERSPACE BUFFERS				0
TOTAL BUFFERS IN SUBPOOL				200
		18:39:19	18:49:12	

	START TRACE	END TRACE	DIFFERENCE
NUMBER OF RETRIEVE BY RBA CALLS RECEIVED BY BUF HNDLR	3	19	16
NUMBER OF RETRIEVE BY KEY CALLS	21321	24316	2995
NUMBER OF LOGICAL RECORDS INSERTED INTO ESDS	0	0	0
NUMBER OF LOGICAL RECORDS INSERTED INTO KSDS	1434	11818	10384
NUMBER OF LOGICAL RECORDS ALTERED IN THIS SUBPOOL	0	0	0
NUMBER OF TIMES BACKGROUND WRITE FUNCTION INVOKED	0	0	0
NUMBER OF SYNCHRONIZATION CALLS RECEIVED	1434	11818	10384
NUMBER OF WRITE ERROR BUFFERS CURRENTLY IN THE SUBPOOL	0	0	0
LARGEST NUMBER OF WRITE ERRORS IN THE SUBPOOL	0	0	0
NUMBER OF VSAM GET CALLS ISSUED	22775	36278	13503
NUMBER OF VSAM SCHBFR CALLS ISSUED	0	0	0
NUMBER OF TIMES CTRL INTERVAL REQUESTED ALREADY IN POOL	14016	16559	2543
NUMBER OF CTRL INTERVALS READ FROM EXTERNAL STORAGE	8878	19848	10970
NUMBER OF VSAM WRITES INITIATED BY IMS/ESA	1458	12097	10639
NUMBER OF VSAM WRITES TO MAKE SPACE IN THE POOL	0	0	0
NUMBER OF VSAM READS FROM HIPERSPACE BUFFERS	0	0	0
NUMBER OF VSAM WRITES TO HIPERSPACE BUFFERS	0	0	0
NUMBER OF FAILED VSAM READS FROM HIPERSPACE BUFFERS	0	0	0
NUMBER OF FAILED VSAM WRITES TO HIPERSPACE BUFFERS	0	0	0

QUOTIENT : TOTAL NUMBER OF VSAM READS + VSAM WRITES = 7.55

TOTAL NUMBER OF TRANSACTIONS

Example 3-2 shows important summary information for each dependent region that was active during the time period that the monitor trace was on. The “Elapsed Time Mean” on each report section is one of the important indicators of system performance. Of course, the percentage of the “Region Occupancy” is another key indicator.

Example 3-2 Region Summary report

IMS MONITOR *** REGION SUMMARY *** TRACE START 2006 250, 18:39:19 TRACE STOP 2006 250, 18:49:12 PAGE 0019

	ELAPSED TIME.....			NOT IWAIT TIME(ELAPSED-IWAIT)			
OCCURRENCES		TOTAL	MEAN	MAXIMUM	TOTAL	MEAN	MAXIMUM	
SCHEDULING AND TERMINATION								
SCHEDULE TO FIRST CALL								
**REGION	1	1	48314669	48314669	48314669			
**REGION	2	1	42696868	42696868	42696868			
**REGION	3	1	46601395	46601395	46601395			
**REGION	4	1	44136700	44136700	44136700			
**REGION	5	1	45194561	45194561	45194561			
**TOTALS		5	226944193	45388838				
ELAPSED EXECUTION								
**REGION	1	1	270851515	270851515	270851515			
**REGION	2	1	275004759	275004759	275004759			
**REGION	3	1	478134880	478134880	478134880			
**REGION	4	1	548731226	548731226	548731226			
**REGION	5	1	547673367	547673367	547673367			
**TOTALS		5	2120395747	424079149				
DL/I CALLS								
**REGION	1	2762	19399174	7023	372895	19399174	7023	372895
**REGION	2	14848	22430506	1510	278980	17016470	1146	278980
**REGION	3	1849	12210030	6603	357590	12210030	6603	357590
**REGION	4	31152	82452077	2646	212657	12980967	416	63772
**REGION	5	98838	40940175	414	42517	11069873	112	37571
**TOTALS		149449	177431962	1187		72676514	486	
IWT/CALL								
IDLE FOR INTENT								
NONE								
CHECKPOINT								
	1	19670	19670	19670	19670	19670	19670	
REGION OCCUPANCY								
**REGION	1	53.8%						
**REGION	2	53.5%						
**REGION	3	88.5%						
**REGION	4	99.9%						
**REGION	5	99.9%						

3.3 IMS Performance Analyzer

The IMS Performance Analyzer (IMS PA) is a performance and analysis tool that processes data from IMS system logs, IMS Monitor data sets, and IMS Connect Extensions Event Collector journal files.

Here are a few of the highlights of IBM IMS Performance Analyzer for z/OS, Version 4.1:

- Provides comprehensive performance analysis and tuning assistance for IMS
- Delivers end-to-end transit analysis for all types of transaction workloads
- Provides a complete picture of the transaction life cycle through IMS Connect and IMS logs
- Manages reporting requirements across the IMS enterprise through ISPF dialog or batch
- Offers DBRC Log selection for quick and easy log report requests
- Provides comprehensive IMS Monitor reporting, including Fast Path, which is not reported with DFSUTR20

IMS Performance Analyzer gives you an opportunity to select or merge IMS log data sets you need, so it is easy to generate any report in any given time period. You can customize how detailed your reports are, look for IMS performance trends, and try to foresee or avoid IMS performance problems based on your performance reports.

For some performance study cases, you might need to analyze the Monitor trace to get a better understanding of the whole transaction flow. The Program Trace report is a detailed trace of the events associated with a program schedule. There is a detailed line of information for each call and, optionally, each IWAIT occurring during the program schedule. There is also a summary of schedule activity.

To obtain the Program Trace report, select Program Trace in the Monitor Report Set and specify the following options for each trace required:

- Date and time range of the trace
- At least one of the following:
 - Program (PSB) name to be traced
 - Transaction code to be traced
 - Region to be traced
- Type of trace: either Short, Long, or Summary
- Schedule limit: the maximum number of schedules to be traced

Example 3-3 shows the output of the PAMON PGMTRACE option.

Example 3-3 PAMON PGMTRACE option

Relative Time	Pgm Time	Call	ST PCB Feedbk				Mod IWT	DDname	IWT Elapsed	Breakdown of Call Time					Call Elap	
Secs.Mil.Mic	Sc.Mil.Mic	No.	PCBname	Func	Cd	Segname	Lvl	ULE (# IWAITs)	Sc.Mil.Mic	Pct	CPY	Pct	DLA	Pct	IWT	Sc.Mil.Mic
3.238	0.037	10	HISTDB	GN	GE	MYROOT	01				.00%	100.00%		.00%		0.007
4.470	1.225	11	HISTDB	ISRT		HIST	02	1	1.112	.00%	100.00%	86.34%				1.288
4.603								FPD DEDB HISTDD	1.112							
24.009.143	24.003.385	12	HISTDB	GU		DEPO2	02	1	0.958	.00%	100.00%	81.67%				1.173
24.009.313								FPD DEDB HISTDD	0.958							
24.010.332	0.016	13	HISTDB	GN	GE	MYROOT	01			.00%	100.00%	.00%		.00%		0.009
24.010.374	0.033	14	HISTDB	GHU		DEPO2	02			.00%	100.00%	.00%		.00%		0.005

3.4 File Select and Formatting Print utility (DFSERA10)

The File Select and Formatting Print utility can be used with IMS and its related databases. Its primary function is to assist in the examination and display of data from the IMS log data set. The utility can:

- ▶ Print or copy an entire log data set.
- ▶ Print or copy from multiple log data sets based on control statement input.
- ▶ Select and print log records on the basis of sequential position in the data set.
- ▶ Select and print external trace data sets.
- ▶ Select and print log records based on data contained within the record itself, such as the contents of a time, date, or identification field.
- ▶ Allow exit routines to perform special processing of selected log records. For example, DFSERA30 is an exit that formats deadlock trace records.

The File Select and Formatting Print utility uses a series of control statements to define the input and output options, selection ranges, and various field and record selection criteria.

You can use Knowledge-Based Log Analysis (KBLA) functions for most analysis that DFSERA10 provides. KBLA was introduced with IMS Version 9.

3.5 Log Transaction Analysis utility (DFSILTA0)

The IMS Log Transaction Analysis utility collects information about individual transactions from the IMS system log. It generates a report that identifies each transaction, the class and priority of the transaction, and the length of time the transaction is in the system. It then calculates the total transaction response time. The definition of response time in this report is the sum of the time on the input queue, the processing time, and the time on the output queue. This information enables an installation analyst to isolate problem areas in the IMS system. It can be used to quickly identify problem transactions and to evaluate or aid in setting up class-scheduling algorithms.

Execution-time parameters include the specification of a start time and the subsequent number of checkpoints to be included from the log. To correspond with the IMS Monitor data, the report can be run to process only the portion of the IMS log that was produced at the same time. The program can also produce another IMS log that includes only a subset of the input log records, which might expedite some subsequent analysis.

When DFSILTA0 is run as a batch job, execution time and CPU utilization are high, and the report can be long. Therefore, subsetting should be done to limit the use of system resources. If wait-for-input (WFI) transactions have been processed with a high processing-limit count (greater than 1 000), DFSILTA0 can execute for a very long time and occupy a large amount of virtual and real storage as it builds in-storage tables for all transactions that are processed at a single scheduling.

DFSILTA0 and the Statistical Analysis utility (DFSISTS0) have been rewritten for IMS Version 10. This was done to improve the performance, simplify the execution, and lift several restrictions. DFSILTA0 and DFSISTS0 no longer require the Log Merge utility to be run in order to merge the input from multiple IMS systems. Both of these utilities also have added support for shared queues and non-recoverable transactions. Also, the execution of the DFSISTS0 utility has been simplified by replacing a previously six step job with a single step job.

3.6 Knowledge-Based Log Analysis (KBLA)

The Knowledge-Based Log Analysis (KBLA), which was introduced in IMS V9, is a set of tools and utilities to assist with finding and interpreting information about the IMS log or IMS trace data sets. The ISPF panel driven user interface is designed to simplify JCL job creation and to prevent JCL errors. All the necessary control statements are created in a syntactically correct form. KBLA reduces the need to reference the manuals and eliminates the work involved in creating jobs for extracting log records. The utility also includes a new set of log formatting routines that allows the user to extract an interpreted version of the IMS log records, producing easy to read output. Key data for each log record is interpreted in plain English, which relieves you from having to research the meaning of each flag or field.

Components of KBLA:

- ▶ A few new log analysis utilities documented in *IMS Version 9 Utilities Reference: System*, SC18-7834
- ▶ ISPF interface to create JCL and control statements for those utilities and for preexisting log analysis utilities
- ▶ ISPF interface to select, group, and sort log data sets
- ▶ ISPF interface to perform environment-related tasks

KBLA processes any and all log records. Example 3-3 only lists the log records which can be formatted by the “K” formatting option under KBLA Log Record Formatting.

Table 3-3 Log records analyzed

Record	Description	Record	Description
X'01'	Message queued from a CNT	X'33'	A record was released by the queue manager
X'02'	A condensed command record	X'34'	A message was cancelled
X'03'	Message queued by a DL/I call	X'35'	A message was enqueued or re-enqueued
X'06'	IMS was started or stopped, or FEOV was issued	X'36'	A message was dequeued, saved, or deleted
X'07'	Program termination	X'37'	Sync point processing
X'08'	Program scheduled	X'38'	A message was put back on the queue after application abend
X'10'	A security violation occurred	X'39'	The output queue was freed during cleanup processing of a RELEASE call
X'11'	A conversational program started	X'40'	Checkpoint records
X'12'	A conversational program terminated	X'42'	OLDS switch or a checkpoint was taken
X'16'	A /SIGN command successfully completed	X'48'	Padding records
X'20'	Database data set open	X'50'	Database update
X'21'	Database data set close	X'55'	External subsystem information

Record	Description	Record	Description
X'24'	The buffer handler detected an I/O error	X'56'	IMS external subsystem log records
X'29'	HALDB online reorganization	X'59'	Fast Path Records
X'31'	A GU was issued for a message	X'63'	Log session initiation and termination
X'32'	A message was rejected	X'72'	Dynamic terminals sign on or sign off

Figure 3-1 shows the overview of KBLA ISPF panel structure.

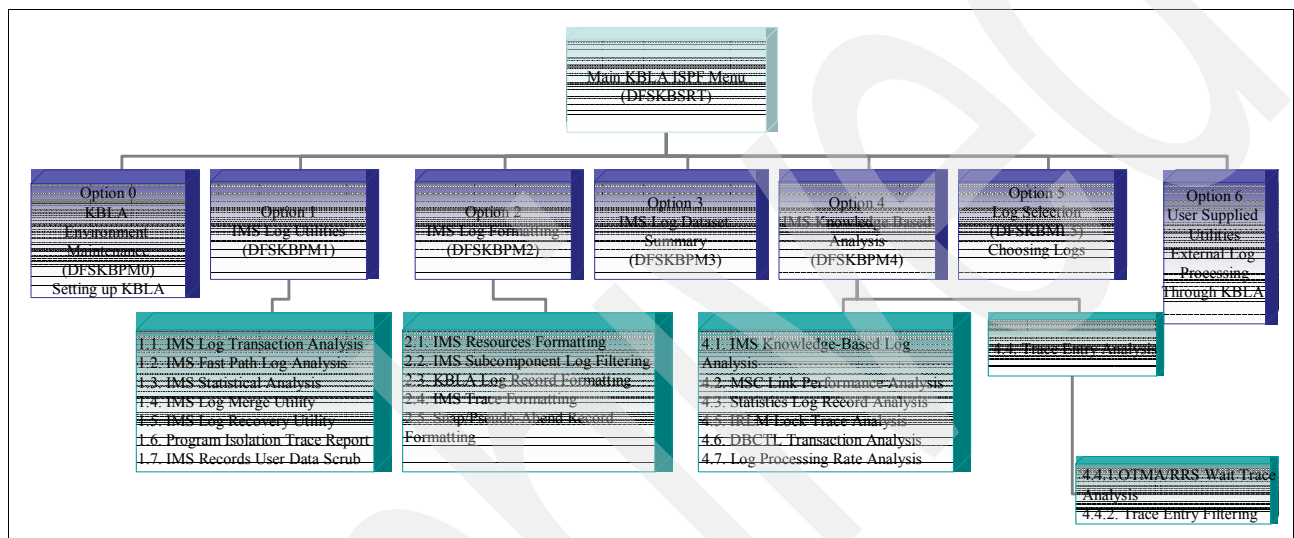


Figure 3-1 KBLA panel structure

Beyond log searching and formatting, there are also utilities for helping to find and diagnose performance-related issues. These utilities include IMS Knowledge-Based Log Analysis, MSC Link Performance Analysis, Statistics Record Analysis, Trace Entry Analysis, IRLM Lock Trace Analysis, DBCTL Transaction Analysis, and Log Processing Rate Analysis. In this chapter, we focus on MSC Link Performance Analysis, Statistics Record Analysis, IRLM Lock Trace Analysis and DBCTL Transaction Analysis, and the use of the utilities.

IMS Knowledge-Based Log Analysis gives you summary statistical information from the log data sets or based on specified search criteria.

The summary function includes:

- ▶ First and last LSN in the log
- ▶ Timestamp (UTC) of the first and last log record
- ▶ Total number of log records in the log data set
- ▶ Presence of internal traces record, system restarts, dump log record, and system checkpoint.
- ▶ Number of log records present for each record ID
- ▶ System configuration
- ▶ Transaction, program, and database record instances

The Log Processing Rate Analysis calculates the rate at which log records are generated. The average length of each type of log record is calculated. The log record generation rates are expressed in records per second and bytes per second for individual log record types, and for log record subtypes if such granularity is requested.

The log rate analysis can include the following:

- ▶ Statistics for record types
- ▶ Statistics for each record subtype within type

On an IMS subsystem, the log record generation rate can vary depending on system activity. An option is available to sample the system logging rate at specified time intervals and to track differences in system activity.

3.6.1 MSC Link Performance Analysis

The MSC Link Performance Analysis utility (DFSKMSC0) uses the MSC link trace written to the log by the `/TRA SET ON LINK n` command. Multiple links can be traced and used as input to the program. The IMS MSC link trace log records provide information about link response times, which can be used to help isolate performance problems with MSC links.

Input for DFSKMSC0

The input is:

- ▶ IMS log:
 - OLDS or SLDS
 - Link trace records (x'6701') captured by `/TRA SET ON LINK n`

You can trace multiple links, but use this capability carefully, because the amount of data can be hazardous to your logs. Currently, only logs from a single system should be used in each execution.

Output for DFSKMSC0

The output of the formatting program shows a line for each link send or receive containing three different delta times. The output also includes the actual time that the operation completed so that send or receive can be matched against the full trace data for further analysis if necessary.

3.6.2 Statistic Log Record Analysis

The Statistic Log Record Analysis utility (DFSKDVS0) processes the X'45' log records generated at each IMS simple or statistics checkpoint and formats a report showing the detailed statistical information between each pair of checkpoints. This information can be used to look for bottlenecks within the IMS system or to detect trends in internal resource usage that can help you to determine if tuning is necessary. There are no IRLM statistics for the statistics checkpoint (`/CHE STATISTICS` command). DFSKDVS0 can be used in DB/TM, DBCTL, or DCCTL environments.

Input for DFSKDVS0

The input is:

- ▶ An IMS log:
 - OLDS or SLDS
 - At least two sets of checkpoint records (x'45') on the logs

Output for DFSKDVS0

The output is:

- ▶ A report for each checkpoint interval

Invoking statistics formatting

Figure 3-2 is an example of the Statistic Log Record Analysis panel in the KBLA panel-driven interface.

== K.B.L.A. Statistic Log Record Analysis ==	
COMMAND ==> _____	
Input IMS Log DSN <u>IMSPSA.SLDSP.IM1B.D06250.T1837390.V2E</u>	Cataloged?
IMS Log Version <u>9</u>	
Output DSN Keyword <u>STAT</u>	The Output DSN will be:
Log DSNs were extracted from RECON. <u> </u>	JOUK03.keyword.KBLA
PDS member containing logs. _____	

Figure 3-2 DFSKDVS0 invoking panel

Statistics record output and interpretation

Each x'45' statistics record has a report as listed below. You should focus on the important values you choose for your environment and look for trends and anomalies over extended periods:

- ▶ Statistics information (header page)
- ▶ Qpool statistics
- ▶ Format pool statistics
- ▶ OSAM subpool starts
- ▶ VSAM subpool starts
- ▶ Variable pools
- ▶ Scheduling statistics
- ▶ Logger statistics
- ▶ PI statistics
- ▶ Latch statistics
- ▶ CBT pools
- ▶ Receive any statistics
- ▶ Fix storage pool
- ▶ Dispatcher statistics
- ▶ Dynamic SAP® statistics
- ▶ RACF® signon statistics
- ▶ IRLM subsystem statistics
- ▶ IRLM system statistics

Example 3-4 shows the AOIP fix storage pool. You might need to specify an additional size to avoid GETMAIN and FREEMAIN requests when values under OVERSIZE keep increasing over time.

Example 3-4 Fix Storage Pool

```
FIXED STORAGE POOL: AOIP
CURRENT POOL SIZE :    22912
MAX POOL SIZE    :    40120
# BYTE IN OVERSIZE:      0
```

OVERALL POOL SIZE :	48752								
SUBPOOL NUMBER :	0								
ABOVE/BELOW 16M :	ABOVE								
BUFFERSET	01	02	03	04	05	06	07	08	OVERSIZE
BUFFER SIZE (BYTES)	56	144	264	584	1056	2104	4200	32776	
PRIMARY BLK BUFFERS	50	500	100	32	16	8	4	4	
INITIAL (Y/N)	Y	N	N	N	N	N	N	N	
SECONDARY BLK BUFFS	50	1500	100	32	8	4	2	2	
MAX BLK SINCE INIT	1	0	0	1	2	0	0	0	
MAX BLK SINCE CHKPT	1	0	0	1	1	0	0	0	0
MIN BLK SINCE CHKPT	1	0	0	1	0	0	0	0	0
AVERAGE BLKS ALLOC	0	0	0	0	1	0	0	0	
MAX BUF SINCE INIT	0	0	0	2	19	0	0	0	0
MAX BUF SINCE CHKPT	0	0	0	2	3	0	0	0	
MIN BUF SINCE CHKPT	0	0	0	2	0	0	0	0	
AVERAGE BUFFS ALLOC	0	0	0	0	1	0	0	0	
TOTAL GET REQUESTS	0	0	0	0	207	0	0	0	
GET REQ PER SECOND	.00	.00	.00	.00	.58	.00	.00	.00	.00
PGLOAD REQUIRED	0	0	0	0	0	0	0	0	0
PGLOAD-IWAIT REQD	0	0	0	0	0	0	0	0	0
BLK ALLOCATE REQD	0	0	0	0	1	0	0	0	0
BLKS RELEASED	0	0	0	0	2	0	0	0	0
CURRENT BLOCK COUNT	0	0	0	0	207	0	0	0	0
CURRENT BUFFER COUNT	0	0	0	0	239	0	0	0	
WASTED STORAGE	0	0	0	0	7452	0	0	0	0
AVERAGE REQ SIZE	0	0	0	0	1020	0	0	0	
UPPER LIMIT REACHED	0	0	0	0	0	0	0	0	0
LARGER BUFF SIZE USED	0	0	0	0	0	0	0	0	0

3.6.3 DBCTL Transaction Analysis

The DBCTL Transaction Analysis utility (DFSKDBC0) combines some of the information found in the DBFULTA0 and DFSILTA0 utilities plus additional DBCTL specific information. By default, the output is presented in termination (sync point) time order, but it can be sorted by specific key fields to help identify potential performance problems.

Input for DFSKDBC0

The input is:

- ▶ An IMS log:
 - OLDS or SLDS
 - x'07', x'08', x'5937', x'5938' records
 - Additional schedule and execution information specifically written for DBCTL

Output for DFSKDBC0

The output is:

- ▶ A report for each transaction or BMP termination

Invoking DBCTL report

Figure 3-3 on page 33 is an example of the DBCTL Transaction Analysis panel in the KBLA panel-driven interface.

```

== K.B.L.A.  DBCTL Transaction Analysis ==
COMMAND ===> _____

Input IMS Log DSN IMSPSA.SLDSP.IM1B.D06250.T1837390.V2E      Cataloged?
IMS Log Version. . . . . 9

Transaction Summary Report Sorted by:
DLI I/O Time . . . . . N (A/D/N)
NBA Buffers Used . . . . . N (A/D/N)
PSBNAME . . . . . N (A/D/N)
Scheduling Elapsed Time. . . . . N (A/D/N)
SYNC Failure . . . . . N (A/D/N)
Time Waiting for DEDB BUFFER . N (A/D/N)
Time Waiting for INTENT. . . . . N (A/D/N)
Time Waiting for POOL SPACE. . N (A/D/N)
Time Waiting for LOCKS . . . . . N (A/D/N)
Time Waiting for CI LOCK . . . . N (A/D/N)
Time Waiting for UOW LOCK. . . . N (A/D/N)

Output DSN Keyword. . . . . DBC      The Output DSN will be:
                                           JOUK03.keyword.KBLA.*

Log DSNs were extracted from RECON . . . . . -
PDS member containing logs . . . . . _____

```

Figure 3-3 DFSKDBC0 invoking panel

DBCTL transaction output and interpretation

Example 3-5 on page 34 shows the output of a DBCTL transaction. Each line shows one PSB from schedule to termination. The information related to that transaction is:

- ▶ PSB name
- ▶ Region type
- ▶ Total elapsed time
- ▶ Transaction schedule time (Termination time = schedule time + total elapsed time)
- ▶ Elapsed time in scheduling
- ▶ Time waiting for intent
- ▶ Time waiting for pool space
- ▶ Total full function calls
- ▶ Total DL/I and I/O count
- ▶ DL/I I/O time
- ▶ Time waiting for locks (including both full function and Fast Path databases)
- ▶ Total DEDB calls
- ▶ DEDB get and put calls
- ▶ Overflow buffer used
- ▶ DEDB buffer waits
- ▶ NBA buffers used
- ▶ Buffers sent to OTHREAD
- ▶ Buffers used for SDEP
- ▶ CI lock waits
- ▶ UOW lock waits
- ▶ VSO reads from data space
- ▶ VSO reads from DASD
- ▶ Updates to VSO data space
- ▶ Sync failure codes

So you can request the sort sequence by the elapsed time (ELAP) to look for long running transactions or sort on lock wait time (LWT) for locking problems.

Example 3-5 DBCTL transaction analysis report

COLUMN HEADING EXPLANATIONS:																											
SCHEL - ELAPSED TIME IN SCHEDULING					INT - TIME WAITING FOR INTENT					PWT - TIME WAITING FOR POOL SPACE													LWT - TIME WAITING FOR LOCK				
DLI - TOTAL FULL FUNCTION CALLS					IOC - DLI I/O COUNT					IOT - DLI I/O TIME																	
DEC - TOTAL DEDB CALLS					DEG - DEDB GET CALLS					DEP - DEDB PUT CALLS																	
OVF - OVERFLOW BUFFERS USED					BWT - DEDB BUFFER WAITS					NBA - NBA BUFFERS USED																	
UPD - BUFFERS SENT TO OTHREAD					SDP - BUFFERS USED FOR SDEP					CLK - CI LOCK WAITS																	
ULK - UOW LOCK WAITS					VRD - VSO READS FROM DATA SPACE					VDR - VSO READS FROM DASD																	
VWR - UPDATES TO VSO DATA SPACE					S/F - SYNC FAILURE CODE - SEE UTILITIES REFERENCE:SYSTEM DBFULTAO																						
1PSBNAME	SUBSYS	R	RGN	ELAP	SCHED_TIME	SCHEL	INT	PWT	DLI	IOC	IOT	LWT	DEC	DEG	DEP	OVF	BWT	NBA	UPD	SDP	CLK	ULK	VRD	VDR	VWR	S	
	ID	T	NBR	SS.T	HH:MM:SS.T	MS	MS	MS	#	#	MS	MS	#	#	#	#	#	#	#	#	#	#	#	#	#	F	
PCROFO	CI1CSAC3	D	59	.0	08:00:41.4	25	0	0	0	0	0	0	23	4	0	0	0	10	1	0	0	0	6	0	0		

3.6.4 IRLM Lock Trace Analysis

The IRLM Lock Trace Analysis consists of three programs (DFSKLTA0, DFSKLTB0, and DFSK LTC0) that run serially to perform IRLM Lock Trace Analysis. DFSKLT A0 is run first to create the control file of global data management block (DMB) numbers and their respective database names. DFSKLT B0 is then used to create the lock wait detail and summary records. DFSKLT C0 formats and prints the information and creates the optional output data set.

Input for DFSKLTx0

The input is:

- ▶ IMS lock trace data
 - x'67FA' log records are needed either from the external trace data set DFSTRAXx, OLDS, or SLDS
- ▶ RECON

Output for DFSKLTx0

The output is:

- ▶ IRLM lock request report on DMB name order
- ▶ IRLM lock request report on wait time order
- ▶ IRLM lock request report on request completion order

Invoking statistics formatting

Figure 3-4 on page 35 is an example of the IRLM Lock Trace Analysis panel in the KBLA panel-driven interface.

```

== K.B.L.A.  IRLM Lock Trace Analysis ==
COMMAND ===> _____

Input Trace DSN.  IMSPSA.IM1B.DFSTRA01          Cataloged?  }
IMS Log Version.  . . . . . 9
COPY1 DSN. . . . IMSPSA.IM0B.RECON1

Output DSN Keyword. . . . . LOCK                The Output DSNs will be:
                                                    JOUK03.keyword.S.KBLA.*
                                                    JOUK03.keyword.R.KBLA.*
                                                    JOUK03.keyword.D.KBLA.*
Create data set with raw output . . _ (Y/N)
Raw output sort selection:
  Sorted by Database Name. . . . . N (A/D/N)
  Sorted by RBA . . . . . N (A/D/N)
  Sorted by Database Name & RBA. . N (A/D/N)
  Sorted by PST Number . . . . . N (A/D/N)
  Sorted by Elapsed Wait Time. . . N (A/D/N)

Log DSNs were extracted from RECON . _
PDS member containing logs . . . . . _____

```

Figure 3-4 DFSKLTx0 invoking panel

Statistics record output and interpretation

Example 3-6 shows the lock request report by wait time order. Normally, you can start from here (unless you are looking for a specific database) and quickly see how much time was spent waiting for locks and which databases are involved. The example shows 8.247 seconds were spent on the database ITEMDB during a 30 second trace elapsed time.

Example 3-6 Lock trace analysis output - Wait time order

Suspended IRLM Lock Requests Summary Report - Wait Time Order Page 001
Trace Date = 09/12/2006 Trace Start Time = 13:07:34 Trace End Time = 13:08:04
Trace Elapsed Time (secs) = 30
Trace Input DSN = IMSPSA.IM1B.DFSTRA02

Database Name	DS Id	Lock Req Count	Wait Count	Not Int Count	Total Time	Average Time	Maximum Time
ITEMDB	01	2163	1905	1905	8.247	0.004	0.04
CUSTDB	01	951	662	630	2.712	0.004	0.02
NORDDB	01	1117	959	959	1.960	0.002	0.01
ORDRDB	01	491	407	407	1.491	0.003	0.07
ORDRSI	01	1013	756	756	1.448	0.001	0.02

Example 3-7 on page 36 shows the detailed report by the request completion order. Based on the DB name you knew from the previous report, you might find a control record or root segment RBA causing the contention in this report.

Example 3-7 Lock trace analysis output - Request completion order

Suspended IRLM Lock Requests Report - Req Comp Order										Page 0001							
Trace Date = 09/12/2006 DSN = IMSPSA.IM1B.DFSTRA02																	
Lock Request	Lock Request	----	Wait-----	PST	--Lock--	-----	Resource-----	Flag	--IRLM---	-----	Call-----	Trace					
Start Time	End Time	Elapsed	Type	Num	Type	Lvl	DB	DS	RBA/HASH	S	RCFB	TRAC	Type	Num	Time	Seq#	
13:07:34.334	13:07:34.334	0.000	F	002	BIDP	4	CUSTDB	01	149B8000	P	CPKF	0000	08C0	REPL	191	13:07:34.334	042A
13:07:34.334	13:07:34.334	0.000	F	002	FPCI	8	HISTDB	01	00011FC0	F	K	0440	08C0				0442
13:07:34.334	13:07:34.334	0.000	F	001	FPCI	8	ORDLDB	01	000FC660	F	K	0440	08C0				0463
13:07:34.334	13:07:34.334	0.000	F	001	FPCI	2	ITEMDB	01	00004790	F		0000	08C0				0471
13:07:34.334	13:07:34.334	0.000	F	001	FPCI	8	STCKDB	03	000100C0	F	K	0440	08C0				0488

3.7 IBM Tivoli OMEGAMON XE for IMS on z/OS

IBM Tivoli OMEGAMON XE for IMS on z/OS, product number 5698-A39, is a monitoring tool under the IBM Tivoli Monitoring Services product portfolio.

IBM Tivoli OMEGAMON XE for IMS on z/OS helps you optimize the performance and availability of your IMS systems. From a single point of control, you can view comprehensive information and analysis across multiple IMS subsystems, or across your IMSplex environment. With Tivoli OMEGAMON XE for IMS on z/OS, you can collect and summarize information about key resources, such as enqueue, I/O, CPU, paging rates, pool storage, and buffer pool metrics. The product provides both granular and system-wide views of your IMS operations, giving you comprehensive information and analysis across multiple IMS subsystems or across your IMSplex environment.

Tivoli OMEGAMON XE for IMS on z/OS has the following features:

- ▶ Views IMS pools showing utilization, pool storage sizes, and amount of free blocks
- ▶ Auto discovers IMS and Internal Resource Lock Manager (IRLM) subsystems
- ▶ Lets you view Coupling Facility statistics to identify factors affecting the performance of IBM Parallel Sysplex environments
- ▶ Monitors workload balancing using shared queues support and data sharing to minimize the impact of locks on shared databases
- ▶ Tracks and optimizes both resource usage and transaction processing
- ▶ Monitors resource usage for IMS regions with detailed metrics, such as CPU usage, I/O activity, storage, paging, and EXecute Channel Program (EXCP)

OMEGAMON XE for IMS divides monitoring among IMS, IMS Connect monitoring, and IMSplex monitoring. In an IMSplex environment, OMEGAMON XE for IMS can be used to monitor the IMSplex Coupling Facility structure sizes, OSAM/VSAM cache, and IRLM lock structures.

3.7.1 OMEGAMON XE for IMS in IMSplex environment

All IMSplex information can be viewed under the IMSplex node within the Tivoli Enterprise™ Portal (TEP). The TEP navigation view displays each IMS Data Sharing and Shared Queues group active from all of the monitored LPARs. OMEGAMON XE for IMS automatically discovers all IMSplex information and displays this group information within the TEP under the IMSplex node without any additional user configuration.

Figure 3-5 on page 37 shows a TEP screen, where there are two data sharing groups active: IMS9 and IRLM91. Coupling Facility statistics are displayed for the IRLM91 data sharing group.

Along with connection information (connection status, connected jobname, and Coupling Facility name), the cache structure statistics are displayed. The Directory Entries (allocated, in use, and percentage of use) are displayed as are the Data Entries (elements) (allocated, in use, and percentage of use).

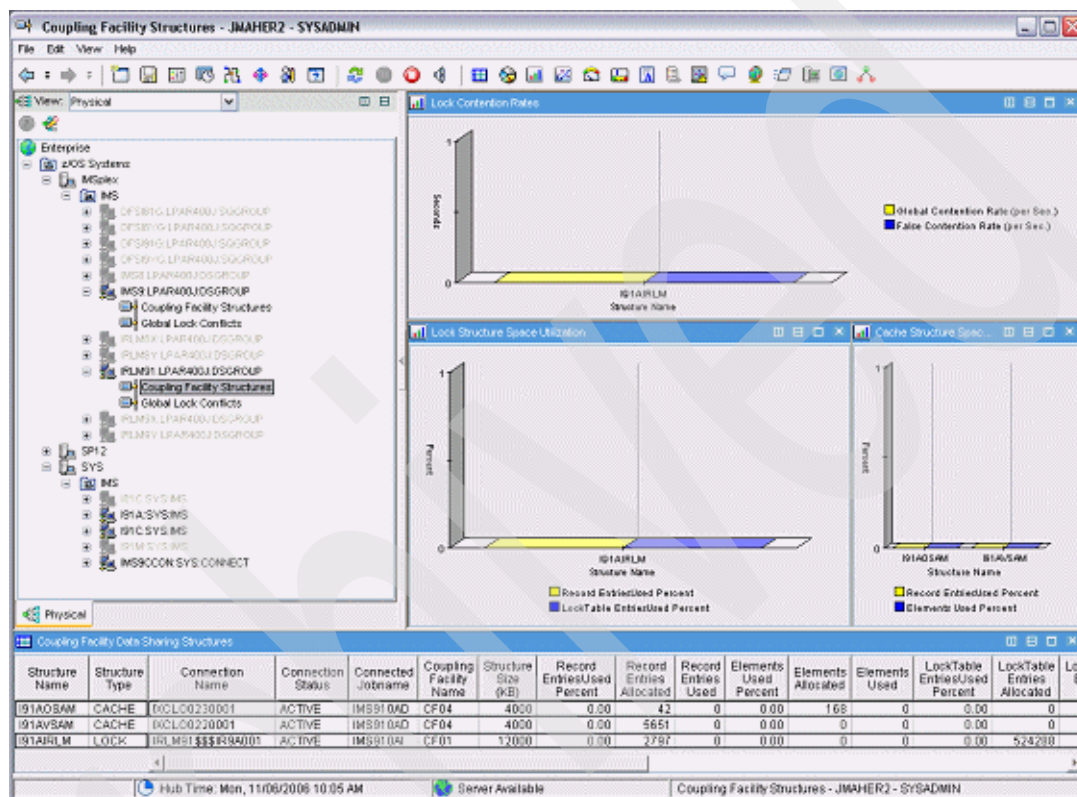


Figure 3-5 TEP screen showing two data sharing groups: IMS9 and IRLM91

Statistics to help determine if the lock structure has adequate space are also displayed. Record List Entries allocated, in use, and Percentage of use are shown. Some of these values require scrolling to the right (Figure 3-6 on page 38).

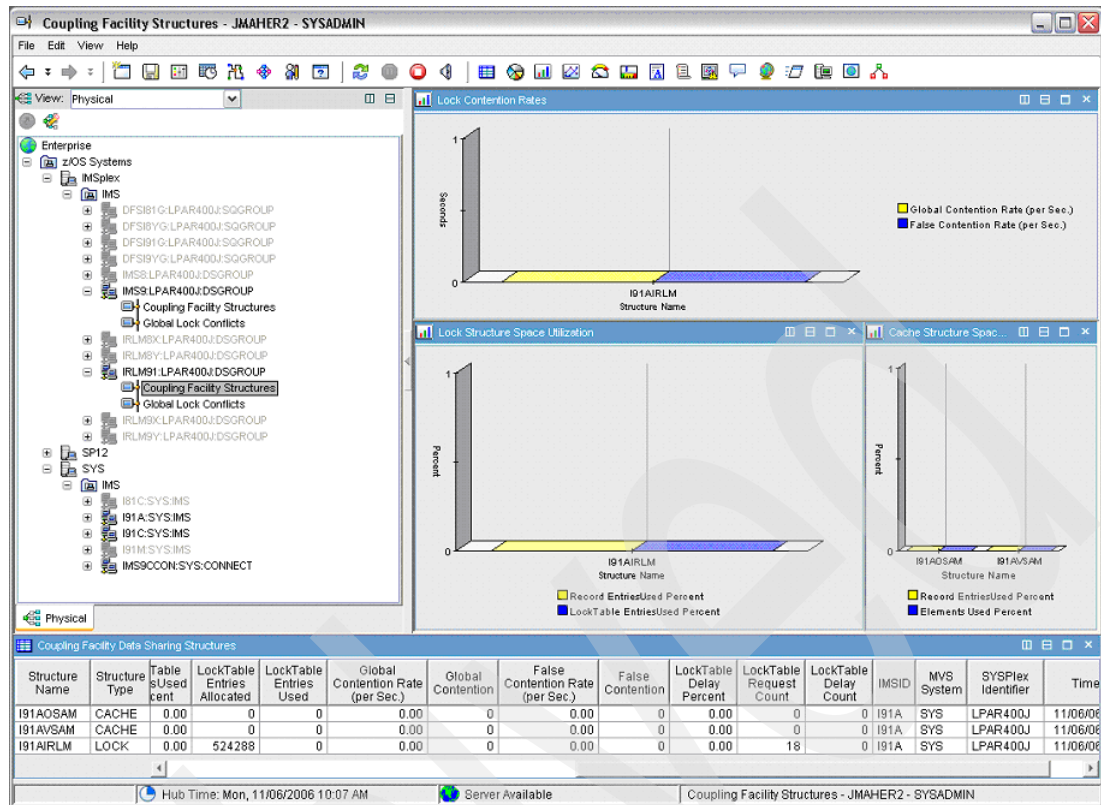


Figure 3-6 TEP screen showing lock structure statistics

A particularly powerful feature of OMEAGAMON XE is the Data Warehousing feature. Using this feature, data can be collected at specified intervals and warehoused for a user-specified amount of time. Further data can be summarized at hourly, daily, weekly, monthly, quarterly, and yearly intervals. Applying this feature to Coupling Facility structure statistics, you can examine structure trends, for example, over the past 24 hours or the past week between the hours of 6 am and 5 pm.

Figure 3-7 on page 39 displays the Data Warehousing selection criteria. This request displays the last 24 hours worth of data.

Select the Time Span

☐ Real time

☒ Last

Last parameters

☒ Use detailed data
Time Column

☐ Use summarized data
Shift
Days

☐ Custom

Custom parameters

☐ Use detailed data
Time Column

☒ Use summarized data
Interval
Shift
Days
Start Time End Time

☐ Apply to all views associated with this view's query

OK Cancel Help

Figure 3-7 Data Warehousing selection criteria specification window

A row is displayed for each structure for each collection interval, as shown in Figure 3-8 on page 40. The collection interval is established during Data Warehouse setup. A unique interval can be specified for each work space.

The screenshot shows a window titled "Coupling Facility Structures - JMAHER2 - SYSADMIN". The window contains a table with the following columns: Recording Time, Structure Name, Structure Type, Connection Name, Connection Status, Connected Jobname, Coupling Facility Name, Structure Size (K-B), Record Entries Used Percent, Record Entries Allocated, Record Entries Used, Elements Used Percent, Elements Allocated, Elements Used, and Lock Entries Per Second. The table displays data for various structures (I91AOSAM, I91AIVSAM, I91AIRLM) and connections (XCLO0220001, IRLM91\$\$\$IR9A001) at different recording times. The status of all connections is "ACTIVE". The table is paginated, showing "Page: 1 of 3".

Recording Time	Structure Name	Structure Type	Connection Name	Connection Status	Connected Jobname	Coupling Facility Name	Structure Size (K-B)	Record Entries Used Percent	Record Entries Allocated	Record Entries Used	Elements Used Percent	Elements Allocated	Elements Used	Lock Entries Per Second
11/08/06 10:45:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 10:45:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 10:45:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 11:00:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 11:00:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 11:00:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 11:15:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 11:15:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 11:15:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 11:30:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 11:30:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 11:30:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 11:45:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 11:45:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 11:45:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 12:00:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 12:00:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 12:00:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 12:15:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 12:15:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 12:15:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 12:30:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 12:30:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 12:30:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 12:46:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 12:46:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 12:46:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 13:01:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 13:01:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 13:01:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 13:16:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 13:16:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 13:16:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 13:31:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 13:31:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	
11/08/06 13:31:00	I91AIRLM	LOCK	IRLM91\$\$\$IR9A001	ACTIVE	IMS910AI	CF01	12000	0.00	2797	0	0.00	0	0	
11/08/06 13:46:00	I91AOSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	42	0	0.00	168	0	
11/08/06 13:46:00	I91AIVSAM	CACHE	XCLO0220001	ACTIVE	IMS910AD	CF04	4000	0.00	5651	0	0.00	0	0	

Figure 3-8 Data Warehouse feature showing structure statistics at collection intervals

3.7.2 Using OMEGAMON XE for IMS for monitoring IMS Connect

IMS Connect support within OMEGAMON XE for IMS is displayed by IMS Connect address space under the LPAR on which it is executing within the Tivoli Enterprise Portal (TEP). Figure 3-9 on page 41 shows several monitored IMS regions (I81C, I91A, and I91C) as well as one monitored IMS Connect region (IMS9CCON).

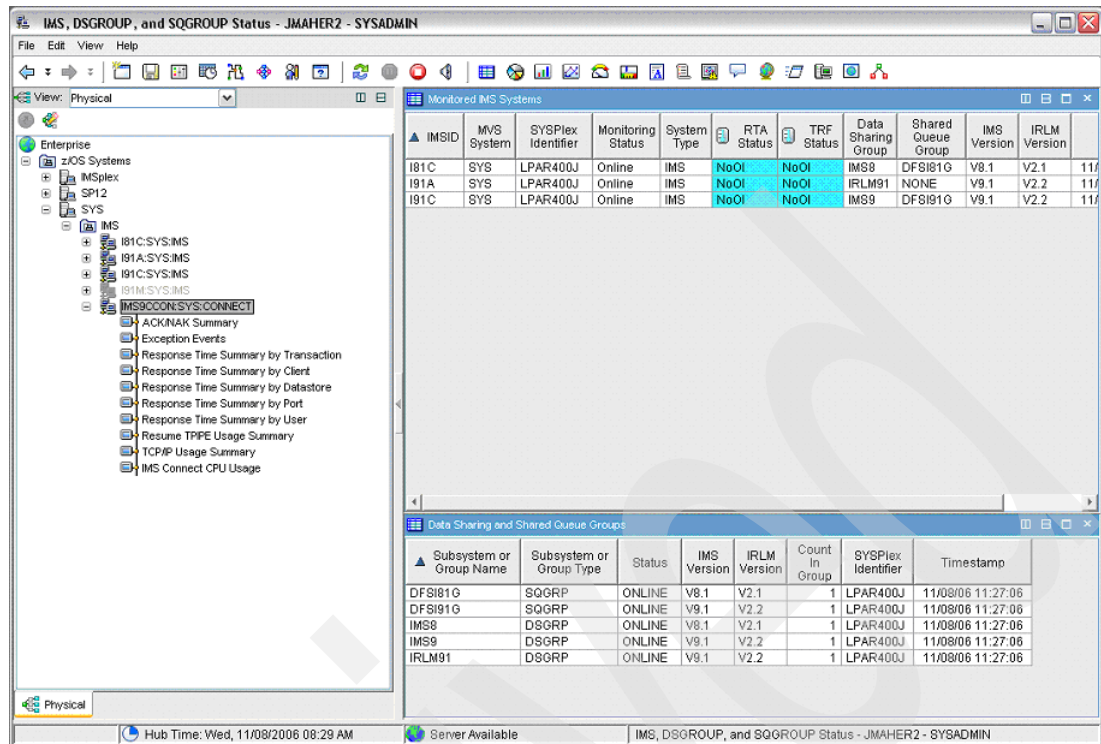


Figure 3-9 TEP screen for IMS Connect

OMEGAMON XE for IMS breaks the IMS Connect processing time into discreet response time components. This response time breakdown allows the IMS Connect user to determine where the response time bottlenecks exist. The IMS Connect processing time is broken down into the following categories:

- ▶ Input Pre-OTMA
- ▶ Input Read Exit
- ▶ Input SAF
- ▶ Process OTMA
- ▶ Output Confirm
- ▶ Output Post-OTMA
- ▶ Output XMIT Exit

Further OMEGAMON use provides response time summaries by the following groupings:

- ▶ Transaction
- ▶ Client ID
- ▶ Datastore
- ▶ User ID
- ▶ IMS Connect Port

3.7.3 Additional references

IBM Tivoli OMEGAMON XE for IMS on z/OS was originally developed by Candle® Corporation and became an IBM product when IBM acquired Candle. Candle products have new IBM names, and during the transition period, some publications use the old names and others use the new names. For example, some publications use the term CandleNet Portal, and others use the new name Tivoli Enterprise Portal.

For a mapping of previous Candle names and new IBM names, refer to:

<http://www.ibm.com/software/tivoli/products/product-matrix.html>

For more information about IBM Tivoli OMEGAMON XE for IMS on z/OS and other members of the Tivoli Monitoring Services family, refer to:

<http://www.ibm.com/software/tivoli/sw-atoz/index0.html>

3.8 IBM IMS Connect Extensions for z/OS

IMS Connect Extensions is an IBM DB2 and IMS Tools program product (program number 5655-K48) that extends and enhances the services of IMS Connect. The tool helps you tune IMS and IMS Connect performance, perform problem determination, and better manage workloads, user exits, and security.

For monitoring and tuning, IMS Connect Extensions provides the following features:

- ▶ Event recording and reporting

IMS Connect Extensions records details about IMS Connect internal events in journal data sets. They give information about these main points:

- Performance and response time for IMS, IMS Connect, and user message exits
- Availability for datastore and ports
- Throughput information for different transaction types, for example, conversational, non-conversational, and send only
- Resource availability

IMS Connect Extensions provides batch utilities to format and print the recorded events. It also provides interfaces to the IMS Performance Analyzer and IMS Problem Investigator products.

- ▶ Status Monitor

The IMS Connect Extensions Status Monitor displays the current activity status information for an active IMS Connect system.

The Status Monitor presents information for the IMS Connect system, datastores, and user message exits for various intervals over the previous hour. It also displays statistical information across the active ports. It enables you to monitor and display IMS Connect activity and utilization in real time.

If the Event Collection feature is active, IMS Connect Extensions continuously collects events as incoming message requests are processed. The number and type of event records collected varies depending on the collection level specified for the IMS Connect system as follows:

- ▶ Level 0

Minimum level. Collects startup, shutdown, and some error events.

- ▶ Level 1

Accounting level. Collects Return from Exit events, OTMA timeout, and session error events. This level provides accounting information in terms of the number of messages by Transaction, User Exit, and so on.

- ▶ Level 2

Transit time reporting. Collects the minimum number of records to run simple transit time reports.

- ▶ Level 3

Comprehensive performance analysis. Collects all TCP/IP read and write events, which provides for analysis of TCP/IP activity.

- ▶ Level 4

Maximum level. Collects all event records.

Event records are first written to an Active Journal data set on DASD, and subsequently archived to the Archive Journal data set on tape or DASD. After collection, the IMS Connect Extensions Print utility provides event record report output. Detailed analysis and reporting is provided by the IBM IMS Performance Analyzer.

For additional information about IMS Connect Extensions, refer to Chapter 11, “IMS Connect Extensions”, in the IBM Redbook *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794, and to the product manuals.

3.9 IBM IMS Buffer Pool Analyzer for z/OS

IBM IMS Buffer Pool Analyzer for z/OS is a program product (program number 5697-H77) in IBM DB2 and IMS Tools portfolio. IMS Buffer Pool Analyzer provides more information than just IMS database buffer pool hit ratios and I/O rates. It provides a way to determine the impact of buffer pool changes before they are made. IMS Buffer Pool Analyzer provides modeling facilities to help you make informed decisions about the addition of buffers to an existing pool or sizing requirements for a new buffer pool. IMS Buffer Pool Analyzer can analyze IMS OSAM and VSAM database buffer pools to provide statistical analysis about the impact of changes, such as:

- ▶ Determining if additional buffers improve the performance of a given buffer pool
- ▶ Determining if reducing buffers impact the performance of a given buffer pool, allowing better allocation of real storage resources
- ▶ Modeling buffer pool usage to estimate I/O rates for different numbers of buffers in each pool
- ▶ Identifying which databases use each database subpool most heavily
- ▶ Providing I/O rates and buffering requirements for a specific database to facilitate buffer pool changes required for changes to database structure (buffer size, access method, or creation of a separate buffer pool for a given database)
- ▶ Performing “what if” scenario analysis, such as identifying the impact of splitting a specific database into a new buffer pool

For additional information about IMS Buffer Pool Analyzer, refer to the product manual *IBM IMS Buffer Pool Analyzer for z/OS User's Guide*, SC18-7068.

Archived

IMS and Workload Manager

This chapter provides a detailed discussion about setting up, identifying, and resolving performance problems related to Workload Manager. It follows from the discussion in Chapter 1, “Defining the performance problem in an IMS environment” on page 1 and discusses z/OS considerations for IMS.

This chapter describes external components outside IMS that can impact the performance of IMS. Variables and events internal to the IMS database manager (DM) are covered in Figure 5 on page 53 and those internal to the IMS transaction manager (TM) are covered in Chapter 6, “Transaction manager performance” on page 125. Application performance considerations are covered in Chapter 9, “Application considerations” on page 169.

This chapter discusses the following topics:

- ▶ Workload manager (WLM) from an IMS perspective
- ▶ CPU management
- ▶ Memory-related issues
- ▶ I/O subsystem issues

4.1 Workload manager in an IMS world

With the introduction of the zSeries and the z/OS operating system, dynamic workload management was introduced. This is implemented by a component known as Workload Manager (WLM). WLM is responsible for ensuring that the multi-workload architecture evident in a z/OS system uses resources based on goals and the importance of each workload. Each workload is also given a service class to distinguish it from other workloads.

WLM is a total replacement of the old IPS and ICS settings and, therefore, cannot coexist. What is required is a business view of all your workloads in order to define the importance of the workload. Let us define some of the rules that are required in order to ensure optimum usage and functionality of WLM:

- ▶ In order to define the correct business policy for WLM, we need to implement a concept of a service level agreement (SLA). The SLA is essentially a contract between the business and the Information Technology (IT) departments to provide a service based on end-to-end response times, availability, and throughput.
- ▶ The workloads need to be defined and grouped into service classes.
- ▶ Each service class or collection of service classes needs to be defined with the SLA agreed to importance and performance goals.
- ▶ The importance of the workload varies from importance 1 through to 5, with 5 the least important.
- ▶ The performance goal is either based on response time, velocity, or discretionary:
 - Response time requires that you specify either your average host response times or percentage of work that needs to complete in a certain response time.
 - Velocity is the measurement of how fast you want work to run when it is ready.
 - A discretionary goal is only executed when system resources are available for this type of workload.

4.1.1 Defining IMS workloads to WLM

The recommended approach is to define all IMS workloads to WLM using response time percentile goals. This is to ensure that WLM does not keep adjusting workloads in order to meet response time measurements, which could cause other workloads to suffer. Workloads defined with predefined goals are called *Goal Mode*, and this is the only supported way to specify z/OS performance objectives.

In order to define your workloads, you need to profile all IMS workloads at peak times in order to define the following:

- ▶ Separate long running bad transactions from transactions that run quickly.
- ▶ Separate transactions that run quickly into high, medium, and low volume transactions.
- ▶ Define service classes for all region types based on transaction class rather than transaction codes. Too many service classes can produce unpredictable results, so your workload needs to be analyzed to determine optimum settings.
- ▶ Obtain average host response times for all workloads.
- ▶ Define what percentage of workloads will meet the average response times. This is normally obtained from your SLA.
- ▶ Set up the workload using response time percentiles.
- ▶ Monitor and change if necessary.

For detailed information about WLM, refer to the IBM Redbook, *System Programmers Guide to: Workload Manager*, SG24-6472.

4.1.2 Rules for ensuring the correct priorities are assigned

Depending on your installation, you might need to adapt the generalized rules that follow to your specific requirements. Table 4-1 provides a list of sample transactions with response and CPU time analyses of all workloads. What follows is the analysis of workload that can be adopted in any organization.

Profile your workload

The workload identified shows a total of 14 transactions with a profile of each transaction during peak hours. This profile must be compared to a historical repository of your workload to ensure response and CPU times are similar.

Table 4-1 Analysis of IMS workload during peak hours

Transaction name and class	Transaction volume	Average host response time in seconds	Average CPU time in milliseconds	Average DB calls per transaction
HG0TRN01 class 1	1 238 421	0.450 s	0.0315 ms	45
HG0TRN02 class 2	1 421 358	0.380 s	0.0311 ms	30
HG0TRN03 class 3	1 212 854	0.410 s	0.0320 ms	34
HG0TRN04 class 4	1 892 251	0.480 s	0.0330 ms	40
HG0IFP01 no class as IFP	12 238 722	0.220 s	0.0225 ms	25
HG0IFP02 no class as IFP	34 228 212	0.240 s	0.0235 ms	28
HG0WFI01 class 7	18 228 354	0.180 s	0.0204 ms	18
HG0WFI02 class 9	22 851 004	0.205 s	0.0215 ms	22
HG0WFI03 class 11	19 251 644	0.150 s	0.0195 ms	18
HG0PFI01 class 15	4 327 890	0.320 s	0.0290 ms	32
HG0PFI02 class 15	5 327 085	0.310 s	0.0280 ms	36
HG0LNG01 class 21	854 356	2 s	9.5800 ms	856
HG0LNG02 class 25	768 252	3 s	9.2588 ms	945
HG0LNG03 class 27	824 931	2 s	9.9824 ms	825

Group your workload into service classes

Group workloads from your transaction profiles and determine workloads that can be set up using the same service class. Table 4-2 describes the type of information that is required to set up the service class definitions in WLM:

- ▶ Transactions HG0TRN01, HG0TRN02, HG0TRN03, and HG0TRN04 can be combined to run in a single class with multiple copies of the dependent region and specifying the MAXRGN parameter to ensure no domination by any specific transaction.
- ▶ HG0IFP01 and HG0IFP02 have separate service classes, because they are high volume transactions. The transaction name needs to be specified, because they are Fast Path transactions.
- ▶ HG0WFI01, HG0WFI02, and HG0WFI03 have individual service classes.
- ▶ HG0PFI01 and HG0PFI02 have a single service class.
- ▶ HG0LNG01, HG0LNG02, and HG0LNG03 have a single service class, because their CPU usage profiles are similar.
- ▶ The percentile provides WLM with a percentage of work in a period that should complete within the response time provided.

Table 4-2 Defining elements of your policy

Service class definition in WLM	IMS transaction class definition in IMS	Response time goal to be achieved	Percentile to achieve goal
IMSTRN01	Class 1	0.430 s	85%
IMSIFP01	Specify transaction	0.220 s	95%
IMSIFP02	Specify transaction	0.240 s	95%
IMSWFI01	Class 7	0.180 s	95%
IMSWFI02	Class 9	0.205 s	95%
IMSWFI03	Class 11	0.150 s	95%
IMSPFI01	Class 15	0.315 s	90%
IMSLNG01	Class 21	2.5 s	80%

When you are confident that your workload is specified correctly, you can proceed to define the WLM policy. Speak to your WLM performance specialist to set up the service class definitions.

Setting up workload in the correct priority

Once the workloads are defined, we need to set up all the IMS address spaces and their relevant importance. With WLM, there are only five importance levels, with 1 the most important and 5 the least important. Table 4-3 on page 49 describes the various IMS address spaces, their importance, and velocity. The WLM specialist in your organization must be provided with information in order to be able to define the WLM policy correctly. He requires input from all areas to ensure business rules are applied throughout the WLM policy, because your system might have a combination of IMS, CICS, and DB2 workloads. You might also decide to introduce a policy for online workloads favoring online transactions and another policy for batch workload at night favoring batch workload. Decisions such as these have to be made in conjunction with your WLM specialist.

Some IMS workloads are put into special WLM service classes called SYSSTC to guarantee service. All other workloads are given importance and execution velocity. The control regions (IMSCCTL) and DLISAS (IMSDLI) could be put in SYSSTC if the need arises. Monitoring the workload reveals the need to change the importance of address spaces and workloads. A special hot batch workload is defined to run special batch workload during your critical online periods. It must be clearly understood that this service class is only meant for workloads that have prior approval before running.

Table 4-3 Recommended settings for IMS address spaces

Address space name	Importance	Velocity
GRS, WLM, and MASTER	SYSTEM	
DBRC, IRLM, CQS, VTAM, TCP/IP, RMF™, and IMS CONNECT	SYSSTC	
IMSCCTL and MONITORS	1	95
IMSDLI	1	75
Message regions high	1	50
Message regions medium	2	85
Message region low	2	40
BMP batch hot	1	20
BMP batch high	2	80
BMP batch medium	2	40
BMP batch low	2	20

When you have defined your address spaces, dependent regions, and BMP workload with the correct importance and velocity, the policy can be implemented by your WLM specialist.

If you are using XRF, RSR, or FDBR, the partners of active IMS systems (FDBR, RSR, or XRF), they need to be set at the same WLM priority as the active systems they are shadowing to ensure that they do not fall behind and thereby lose their purpose of enhancing data and service availability.

Monitoring your workloads

The implementation of the policy requires monitoring all aspects of your system to ensure response times are not degraded. You will probably end up changing the policy many times, before it becomes an ideal fit for your organization. Table 4-3 assumes that IMS online workload is extremely important to your organization and has been structured as such.

WLM by itself has no way of knowing IMS response times. When the control region starts up, it registers itself to WLM and informs WLM of the identity of all other address spaces, such as DLISAS, DBRC, and IRLM. During dependent region startup, a special control block called the *performance block (PB)* is built. The PB is used to report the status of the work executing in the dependent region, allowing WLM to meet its goals. One of the components of the PB is the response time measurement updated by IMS and used by WLM to manage its goals.

Monitoring must include the following:

- Reviewing transaction profiles to ensure response times are not compromised and CPU usage is comparable to what you have historically.

- ▶ Reviewing the WLM reports available through the Resource Management Facility (RMF) to ensure your performance index is close to 1 for all address spaces and response time goal measurements on transactions. Change the importance or the percentiles for response time goals or shuffle address spaces if required:
 - A performance index below 1 means that your goals are being exceeded and your workload could possibly be a CPU donor if WLM requires CPU.
 - A performance index of 1 means your goals are being met.
 - A performance index above 1 means your goals are not being met.
- ▶ Review the usage of memory by various tasks to ensure no paging.
- ▶ Review the I/O subsystem to ensure no I/O delays evident.

4.2 CPU management

CPU usage is controlled through dispatching priorities. The dispatcher has a queue of ready work, sorted in priority order. When running WLM in goal mode, dispatching priorities are provided by WLM.

A dispatching priority is a number from 0 to 255 associated with a dispatchable piece of work in the form of a task control block (TCB) or service request block (SRB). Table 4-4 provides dispatching priorities as related to managing workloads.

Table 4-4 Dispatching priorities as related to CPU management

Dispatching priority	Usage
255	Reserved for SYSTEM tasks only
254	Reserved for SYSSTC tasks only
253	Reserved for SYSSTC tasks only
208 to 252	Importance 1 through to 5
202 to 207	Not used
192 to 201	DISCRETIONARY or SYSOTHER
191	Quiesce work, only use CPU if available

The IMS address spaces, IMS regions, and BMP batch workload are dispatched according to the importance and velocity. The dispatching priority of the address spaces also changes depending on whether the goals are being achieved.

The obvious issue is that we cannot define all workloads as high priority. Constant monitoring is required to ensure goals are being met. In a busy system, WLM finds a donor from which to steal CPU. We need to make sure the donor is not a critical workload. Review your performance indexes regularly to ensure they are as close to 1 as possible.

4.3 Memory management

With z/Architecture™ and z/OS V1R6, expanded storage no longer exists. All page frames are managed in central storage. Paging still exists between central and auxiliary storage. With the 64-bit z/OS architecture, memory management has been improved significantly. IMS

provides various ways to load most of its structures into memory. Chapter 6, “Transaction manager performance” on page 125 provides more details about IMS structures that can be loaded into memory.

Memory is a relatively cheap resource and you should consider upgrading memory on your processors on any paging-related problems.

4.3.1 Identifying memory-related problems

With WLM, memory usage is determined by the importance of workload and the achievement of goals. Paging and page stealing would only be noticed with small amounts of memory defined to the z/OS operating system. If your architecture has memory constraints, then it would be feasible to ensure that you do not page fix any large IMS control blocks.

The best performance for an IMS system is achieved when all IMS structures exist as pages in central storage. In a busy IMS system, this is almost always the case.

The only recommendation we have about memory is that you must ensure zero or as close to zero paging as possible. This can only be achieved if your workload is structured correctly in WLM, and memory is not a constraint in your architecture.

4.4 I/O subsystem

The I/O response time is the most significant delay that an IMS transaction experiences in its life cycle. As CPU speeds increase, the I/O response time becomes more important. The I/O response time is essentially made up of four components:

- ▶ The IOSQ time is the time spent in the z/OS operating system waiting for the device.
- ▶ The PEND time is the time spent from the start of the SSCH to when dialog begins between the channel and the controller. With the introduction of FICON®, this time has almost disappeared and should not appear as a problem. Most of this time has moved into disconnect time.
- ▶ Disconnect time is the time that the channel and controller are not in dialog. With modern controllers, this time is mostly related to synchronous remote copy (PPRC). Other causes could include high sequential writes, control units busy, or the block is not in the cache resulting in a cache miss.
- ▶ Connect time is the time spent doing the actual I/O. This is the transfer of data to and from the controller cache.

4.4.1 Ideas to minimize I/O contention

IMS provides many ways in which to improve I/O response time or eliminate I/O totally. Databases in IMS can either use the virtual storage access method (VSAM) or the overflow sequential access method (OSAM):

- ▶ All heavily used databases together with their index and data components can be loaded in either VSAM or OSAM buffer pools. You need to analyze all of your database profiles in order to decide whether to dedicate buffer pools to databases.
- ▶ Enabling sequential buffering for OSAM performs read ahead processing, reducing application run times significantly.

More information about rules for allocating buffer pools appears in Figure 5 on page 53.

WLM also provides ways to assist you with I/O contention and priority:

- ▶ If I/O priority management is activated, then WLM calculates the I/O priority based on demand and goal achievement for each service class. In this case, the I/O priority is independent from the CPU dispatching priority.
- ▶ If I/O priority is not used, WLM uses the CPU dispatching priority to perform the I/O.
- ▶ Enabling parallel access volumes (PAV) allows z/OS to perform multiple I/O operations against the same physical volume.

Database performance

The primary goal of improving how a database performs is to reduce the number of I/Os that an application program must make. Reduce the number of I/Os a program makes, the faster the application runs. In this chapter, we discuss the following topics:

- ▶ Selecting an access method
- ▶ HISAM as opposed to HD access methods
- ▶ (P)HDAM as opposed to (P)HIDAM
- ▶ HALDB
- ▶ OSAM as opposed to VSAM
- ▶ HALDB partition selection
- ▶ Block sizes, CI sizes, and record sizes
- ▶ Free space
- ▶ Randomization parameters
- ▶ Fixed length as opposed to variable length segments
- ▶ Pointer options
- ▶ SCAN= parameter on the DATASET statement
- ▶ Multiple data set groups
- ▶ Compression
- ▶ Encryption
- ▶ Secondary indexes
- ▶ Fast Path performance considerations
- ▶ Non-recoverable databases
- ▶ Overflow sequential access method (OSAM)
- ▶ Virtual storage access method (VSAM)
- ▶ Improve GSAM performance
- ▶ When to reorganize

In this chapter, we do not provide advice about the logical structure of a database, such as the hierarchical structure, the placement of fields within segments, or the keys that you should use. These items are determined by the application data and the requirements of the application design. Logical databases and logical relationships are not covered in this chapter. Hierarchical sequential access method (HSAM) and simple hierarchical sequential access method (SHSAM) are not covered. These are special purpose databases, which you cannot update. They can only be loaded and read.

5.1 Access methods

IMS has several access methods for storing data. Each has its own attributes and uses. Some methods perform better if the application is sequentially reading the data; others perform better if the application randomly accesses the data. Choosing the right access method is important to the overall performance of the database.

When creating the IMS database descriptor (DBD), you have a few options to choose from to select your database access method. The database access method defines to IMS how the program processes the segments. In general, you base the access method chosen on the application requirements. Below are the types of IMS databases.

Hierarchical sequential access method

Hierarchical sequential access method (HSAM), which can use either BSAM or QSAM, has the following attributes:

- ▶ Like a flat or sequential file
- ▶ Must process in sequential, top to bottom, left to right, and front to back sequence
- ▶ No direct access to root segments
- ▶ Can have many levels of segments
- ▶ Can perform GU, GN, and GNP calls
- ▶ ISRT call allowed only when database is loaded
- ▶ Database update done by merging databases and writing new database
- ▶ Not used in IMS online environment

Simple hierarchical sequential access method

Simple hierarchical sequential access method (SHSAM), which can use either BSAM or QSAM, has the following attributes:

- ▶ This database structure must be a root only database.
- ▶ Same as HSAM.
- ▶ Not used in IMS online environment.

Hierarchical indexed sequential access method

Hierarchical indexed sequential access method (HISAM), which uses VSAM, has the following attributes:

- ▶ Direct access to root segments but access to dependent segments is sequential through displacements.
- ▶ Simple database structures: a root and under six children.
- ▶ Used when deletion of segments is minimal.
- ▶ Used when there are lots of roots and very few children.
- ▶ Good access method for tables when data remains the same, or when very few updates are done.
- ▶ Used when inserts are few.
- ▶ Used when database records, (a root and all its children), are the same length.
- ▶ All DL/I calls allowed.
- ▶ Can be used in IMS online environment.
- ▶ There are two bytes of segment prefixes: the segment code and the delete byte.
- ▶ The DBD is not used to define the CI size or free space; the VSAM delete define statements take care of that function.

- ▶ VSAM size limit = 4 294 967 296 bytes (4 gigabyte limit).

Simple hierarchical indexed sequential access method

Simple hierarchical indexed sequential access method (SHISAM), which uses VSAM, has the following attributes:

- ▶ This database structure must be a root only database.
- ▶ No segment prefix.
- ▶ Same as HISAM.
- ▶ VSAM size limit = 4 294 967 296 bytes (4 gigabyte).

Generalized sequential access method

Generalized sequential access method (GSAM), which can use QSAM/BSAM or VSAM, has the following attributes:

- ▶ Like a flat or sequential file
- ▶ Has no hierarchy, database records, segments, or keys
- ▶ Used only for batch or BMP checkpoint/restart processing
- ▶ Not used in IMS online environment
- ▶ Can reposition to a specific block/record through restart
- ▶ Can perform GU, GN, and ISRT calls

Hierarchical direct access method

Hierarchical direct access method (HDAM), which can use OSAM or VSAM, has the following attributes:

- ▶ Direct access to root segments but access to dependent segments can also be direct through the use of pointers.
- ▶ Root segments are not physically stored in root ascending key sequence but are stored in a random order, that is, if you perform an unqualified GN on the root segment of the HDAM database, the sequence of the roots is 123, 001, 077, 078, 415, 002, and so on.
- ▶ Can be used in IMS online environment.
- ▶ Usually, it requires one I/O to get to the root segment.
- ▶ Complex database structures.
- ▶ Segment prefix: segment code, delete byte, and pointers.
- ▶ Default pointers are physical twin forward (PTF) and physical child first (PCF).
- ▶ CI size is determined in the define cluster, not in the DBD if VSAM.
- ▶ FREESPACE is determined in the primary DBD, not the define cluster.
- ▶ FREESPACE is determined in the define cluster, not in the DBD for secondary indexes along with CI size.
- ▶ OSAM size limit = 85 899 345 920 bytes (80 gigabytes [8 GB limit per data set, 10 data sets max]).
- ▶ VSAM size limit = 42 949 672 960 bytes (40 gigabytes [4 GB limit per data set, 10 data sets max]).

Partitioned hierarchical direct access method

Partitioned hierarchical direct access method (PHDAM), which can use OSAM or VSAM, has the following attributes:

- ▶ Same as HDAM.
- ▶ Partition on a key (by high key or a substring of the root key).

- ▶ Partitions of a HALDB can be operated in parallel and independently of each other, so one is offline and others can be online.
- ▶ Partitions have the ability to be reorganized online with complete availability with HALDB integrated online reorganization (OLR) in IMS Version 9.
- ▶ HALDBs do not require utilities to correct pointers in logical relationships. The pointers are automatically corrected as needed using the indirect list data set (ILDS).
- ▶ Must be registered with DBRC.
- ▶ Size limit = 43 980 465 111 040 bytes (40 terabytes [1 001 partitions, 10 data sets per partition maximum, 4 gigabyte limit per data set]).

Hierarchical indexed direct access method

Hierarchical indexed direct access method (HIDAM), which can use OSAM or VSAM, has the following attributes:

- ▶ Direct access to root segments but access to dependent segments can also be direct through use of pointers.
- ▶ Root segments are physically stored in root ascending key sequence, that is, if you perform an unqualified “GN” on the root segment of the HIDAM database, the sequence of the roots is 001, 002, 077, 078, 123, 415 and so on.
- ▶ Can be used in IMS online environment.
- ▶ In general, it requires three I/Os to get to the root segment, two to read the index and one more to read the data.
- ▶ Complex database structures.
- ▶ OSAM size limit = 85 899 345 920 bytes (80 gigabytes).
- ▶ VSAM size limit = 42 949 672 960 bytes (40 gigabytes).

Partitioned hierarchical indexed direct access method

Partitioned hierarchical indexed direct access method (PHIDAM), which can use OSAM or VSAM, has the following attributes:

- ▶ Same as HIDAM.
- ▶ Partition on a key (by high key or a substring of the root key).
- ▶ Partitions of a HALDB can be operated in parallel and independently of each other, so one is offline and others can be online.
- ▶ Partitions have the ability to be reorganized online with complete availability with HALDB integrated online reorganization (OLR) in IMS Version 9.
- ▶ HALDBs do not require utilities to correct pointers in logical relationships. The pointers are automatically corrected as needed using the indirect list data set (ILDS).
- ▶ Must be registered with DBRC.
- ▶ Size limit = 43 980 465 111 040 bytes (40 terabytes)

Data Entry Database

Data Entry Database (DEDB), which uses Media Manager, has the following attributes:

- ▶ Virtual storage
- ▶ VSAM Entry-sequenced data set (ESDS)
 - Requires Media Manager, ICF, and DFSMS
 - 127 segment types, including SDEP
- ▶ The field (FLD) call

- ▶ Full DBRC support
- ▶ HSSP support
- ▶ DEDB utilities
- ▶ A full hierarchical model, including support of insert and delete calls
- ▶ Partitioned into areas (from 1 to 2 048)
- ▶ Allows for up to seven multiple area data sets (MADS)
- ▶ VSO DEDB
 - Load structures in data space to prevent I/O
- ▶ VSAM size limit = 8 796 093 022 208 bytes (8 terabytes [4 GB limit per area (data set), [2 048 areas maximum]])
- ▶ For DBCTL, only available to BMPs

Main Storage Database

Main Storage Database (MSDB), which uses memory, and has the following attributes:

- ▶ Virtual storage
- ▶ The field (FLD) call
- ▶ Fixed length segments
- ▶ Not offered in DBCTL
- ▶ MSDB not recommended any more, use VSO DEDB instead
- ▶ Not shareable in a data sharing environment

INDEX

This method has the following attributes:

- ▶ Primary index to occurrences of the root segment type in a HIDAM database, or a secondary index to a segment type in a HISAM, HDAM, or HIDAM database.
- ▶ For the primary or secondary index to a HIDAM database, VSAM must be specified as the operating system access method.
- ▶ There is no separate index database defined for the primary index of a PHIDAM database.

PSINDEX

This method has the following attributes:

- ▶ Partitioned secondary index to a segment type in PHDAM and PHIDAM databases.
- ▶ VSAM must be specified as the operating system access method.

5.1.1 Selecting an access method

The overriding choice for new databases is HALDB, PHDAM, or PHIDAM, because that is where new development will occur. Otherwise, the first question you need to ask yourself is what type of processing will be done against my database? Will it be direct or sequential? If the processing will be direct, then HISAM, SHISAM, (P)HIDAM, or (P)HDAM are good choices. If the access is going to be sequential, then HISAM, SHISAM, or (P)HIDAM are appropriate. If the applications that run against this database are doing both sequential and direct processing, then (P)HIDAM will perform well.

If there are a large number of updates performed; or the database records vary in length; or you need to take advantage of logical relationships; or there is a good chance that a secondary index is necessary; then we recommend using (P)HIDAM or (P)HDAM. For existing HIDAM and HDAM databases, our recommendation is to convert to PHIDAM or PHDAM, especially when you require greater capacity.

5.2 HISAM as opposed to HD access methods

IMS full function databases can use either the hierarchical indexed sequential access method (HISAM) or one of the hierarchical direct (HD) access methods.

The HISAM database types are:

- ▶ Hierarchical indexed sequential access method (HISAM)
- ▶ Simple hierarchical indexed sequential access method (SHISAM)

The HD access method database types are:

- ▶ Partitioned hierarchical direct access method (PHDAM)
- ▶ Partitioned hierarchical indexed direct access method (PHIDAM)
- ▶ Hierarchical direct access method (HDAM)
- ▶ Hierarchical indexed direct access method (HIDAM)

HISAM and the HD access methods differ primarily in the way that they store segments and use space in their data sets.

5.2.1 HISAM

HISAM stores segments in two data sets. These are the primary data set and the overflow data set. The primary data set is a key-sequenced data set (KSDS). The overflow is an ESDS. For each database record, HISAM places the root and some dependents in a logical record in the primary data set. The number of dependents that HISAM stores in the primary data set depends on the size of the logical record that you define. HISAM places dependent segments, which do not fit in the index, in the overflow data set. The dependents might require multiple logical records in overflow.

Your choices for the logical record sizes of the primary and overflow data sets can have a significant effect on database performance. Ideally, you want IMS to store a database record only in the primary data set. This eliminates I/Os to the overflow data set. If the database records are uniform in size and not too large, your choice is simplified. You can make the logical record size of the primary data set large enough to hold a database record without wasting a lot of space. If database record sizes vary greatly, a primary record size large enough to hold the larger database records would waste space for others.

Each logical record in the overflow data set contains segments from only one database record. Any free space in the logical record can only be used for inserts into the same database record. This can make sizing logical records difficult when database records vary in size. Large logical record sizes tend to waste space. Small sizes tend to spread database records over more logical records. This requires IMS to perform more I/Os to process the database.

HISAM does not use pointers to navigate between segments in a database record. It always stores segments in hierarchical sequence. IMS accesses a segment by reading all of the segments from the root to the required segment. This can be disadvantageous with large database records. Large records might require IMS to read multiple logical records in the overflow data set. There are two bytes of segment prefixes: the segment code and the delete byte. The DBD is not used to define the CI size or free space, the VSAM delete define statements take care of that function.

HISAM does not reuse space that is occupied by deleted dependent segments. These segments remain in the data sets when they are deleted. IMS sets a bit in their prefix to indicate that they have been deleted. This has two effects. First, IMS might have to read through these segments to reach other segments. Second, later inserts cannot use space

created by these deletions. Instead, they have to expand the database. You must reorganize the database to recover the space.

Inserts of roots into a HISAM database require a new logical record in the primary data set. If a logical record is not available in the CI, a CI split is required.

Inserts of dependent segments might require updates to multiple CIs. Because the segments are maintained in hierarchical sequence, the insert of a segment in the middle of a database record requires the movement of the following segments. If there is not room at the end of the last logical record used by the database record, new dependents require a new logical record. This logical record is at the end of the overflow data set. If there are high volumes of inserts, there tends to be a high volume of activity at the end of the data set. IMS must extend the data set frequently.

Replacing variable length segments often requires the movement of other segments. If the size of the variable length segment changes, the segments, which follow it in the database record, must be moved. Similar considerations apply to fixed length segments when they are compressed. Compressed fixed length segments are actually variable length when they are stored. An explanation of compression appears in 5.12, "Compression" on page 84.

HISAM does not support multiple data set groups. You can define only two data sets, the primary (KSDS) and the overflow (ESDS), for a HISAM database. See 5.11, "Multiple data set groups" on page 82 for an explanation of multiple data set groups.

To insert dependent segments under the root, define the VSAM record size to be larger than the average database record length at initial load time. This leaves unused space in the VSAM record, which can then be used at update time to insert dependent segments for a given root.

To insert new roots into the database, there needs to be free space in the KSDS after the initial load. This is accomplished by coding the free space percentage parameter. This percentage needs to be a multiple of the amount of space it takes to store one record using the average database record length. For example, if it takes 3% of the CI to store one record, then free space should always be expressed in 3% increments. Such as 3, 6, 9, 12, and so on. If you code anything other than zero for the CA free space, VSAM leaves at least one CI for each CA empty during a load. If inserts to the database are evenly distributed throughout, then the CA free space should be the same as the percentage of new records. If you are always inserting roots whose keys are higher than the highest key in the data set, do not leave any free space in the CI or CA. For more detailed information about free space, see 5.6, "Free space" on page 70.

Recommendations for using HISAM

HISAM is best suited for databases with the following characteristics:

- ▶ Database records are relatively small. They typically require only one logical record in the primary data set (KSDS). Occasionally, they might require one logical record in the overflow data set (ESDS).
- ▶ Database records are relatively uniform in size. There is not a large range in size for most database records.
- ▶ The database is stable. There are not high volumes of deletes and inserts in the database.
- ▶ The database will not grow past the data set size limitation of 4 gigabytes per data set.

5.2.2 SHISAM

Simple HISAM (SHISAM) is a simplified version of HISAM. SHISAM databases have the following restrictions:

- ▶ They have no dependent segments. Only root segments are allowed.
- ▶ They cannot have secondary indexes.
- ▶ They cannot have logical relationships.
- ▶ All segments are fixed length. Variable length segments are not supported.
- ▶ Compression or encryption is not allowed.

A SHISAM database is stored in a KSDS. The segments do not have an IMS prefix. Since they do not have an IMS prefix, you can process a SHISAM database as a KSDS without using DL/I calls. Conversely, you can process a KSDS with fixed length records as a SHISAM database using DL/I calls.

Since SHISAM segments have no prefix and SHISAM requires only one data set, SHISAM makes efficient use of space. It requires less space than a root-only HISAM, HIDAM, or PHIDAM database.

Recommendations for using SHISAM

If you have a database that meets the SHISAM restrictions, you should consider using SHISAM for it. Additionally, if you need to process a KSDS with fixed length records as an IMS database, you can define it as SHISAM.

5.2.3 HD access methods

HD access methods store segments in one or more data sets. They can use either OSAM or VSAM ESDS for these data sets. HIDAM and PHIDAM databases also include indexes. The indexes point to the root segments, which are stored in the ESDS or OSAM data sets.

HD access methods and HISAM have different techniques for navigating between segments. HD uses pointers, which are stored in the prefix of segments. A prefix can contain separate pointers to different segment types. This allows IMS to more directly navigate to dependent segments. Figure 5-1 illustrates this. Segment A has pointers to the first B segment, the first C segment, and the first D segment. IMS does not have to access any of the B or C segments to reach the D segments.

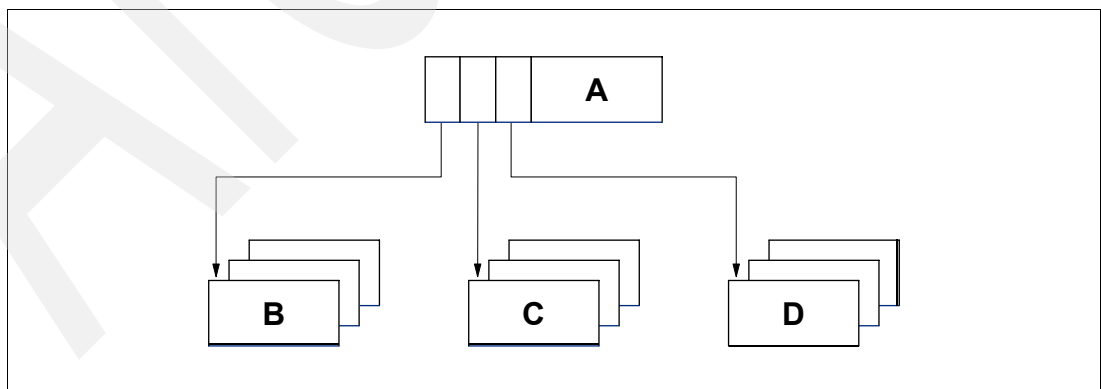


Figure 5-1 HD pointers

The use of direct pointers usually makes HD the preferable access method for databases with large database records. It is easier for IMS to navigate to individual segments within the database.

HD is the preferable access method for highly volatile databases with many deletes and inserts. HD reuses the space occupied by deleted segments. When you delete segments, their space is freed. Inserts can place segments in this free space.

CI splits can occur when you insert roots into HISAM, HIDAM, and PHIDAM databases. But there is a significant difference in the likelihood of splits with the different database types. The logical records for HISAM KSDSs are much larger than those for HIDAM and PHIDAM. The logical records for HISAM contain the root segment and, typically, several dependents. HIDAM and PHIDAM indexes contain only the key, a delete byte, and a four-byte pointer to the root segment. Because HISAM logical records are much larger, fewer will fit in a CI. This affects the probability of CI splits. The following example illustrates this point. Consider a HISAM database whose primary data set has 1K logical records and an 8K CI size. Twenty-five percent free space would provide room for only two inserts before a split would be required. Compare that with a HIDAM or PHIDAM database. If the key were 10 bytes, each logical record would be only 15 bytes. Twenty-five percent free space in an 8K CI would provide room for 136 inserts before a split would be required. HIDAM and PHIDAM are typically better choices for databases with many inserts of root segments.

Replacing variable length segments does not require the movement of other segments in HD databases. If the segment grows so that it does not fit in its previous location, IMS stores the data portion of the segment elsewhere. The segment's prefix does not move. Pointers to this segment from other segments are not changed. A pointer to the data portion is added to the prefix. This technique minimizes the effect on the rest of the database for these types of replacements.

HD supports multiple data set groups. They give you greater flexibility in handling varying space requirements and access patterns for different segment types. See 5.11, "Multiple data set groups" on page 82 for an explanation of multiple data set groups.

HD includes HALDB, PHDAM, and PHIDAM databases. These databases can contain up to 1 001 partitions and hold up to 40 terabytes of data. Additional HALDB specific information and recommendations can be found in 5.4, "HALDB" on page 64.

Recommendations for using HD access methods

HD is well suited for databases with the following characteristics:

- ▶ Database records have great variations in their sizes.
- ▶ Database records are large and contain many types of segments.
- ▶ There are high volumes of deletes and inserts in the database.
- ▶ The database might have significant growth.

5.3 (P)HDAM as opposed to (P)HIDAM

PHDAM, HDAM, PHIDAM, and HIDAM databases share some of the same characteristics. They use direct pointers to navigate between segments in a database record. When segments are deleted, the space they occupied is available for the insertion of other segments. The primary difference between the types is that PHIDAM and HIDAM have indexes. This allows you to easily process the databases in root key sequence. PHDAM and HDAM use hashing, or if you prefer, randomizing modules to determine where root segments are stored. IMS does not store or retrieve PHDAM or HDAM roots in key sequence.

HDAM or PHDAM are good for key read access to the root and segments under that root. A good general rule when choosing between (P)HIDAM and (P)HDAM is that if the application (all of the programs that access this database) is doing about five to 10 times as many GUs as GNs on the root segment, then it is a candidate for (P)HDAM.

5.3.1 Space use

There are some important distinctions in how space is used by PHDAM and HDAM as opposed to PHIDAM and HIDAM. PHIDAM and HIDAM tend to use less space. When they are reorganized, the database records are written in sequence leaving only the specified free space. When PHDAM and HDAM are reorganized, the database records are chained from their root anchor points (RAPs). These are spread across the root addressable area (RAA®). If the database records vary in size, it might be difficult to create free space that is evenly spread across the RAA. More information and advice about free space appears in 5.6.1, “Specifying free space” on page 70. Because PHIDAM and HIDAM tend to use less space, batch jobs, which process the entire database, typically perform better with these database types.

5.3.2 Sequential processing

PHIDAM and HIDAM databases allow you to access database records in key sequence. If you require key-sequential access for a PHDAM or HDAM database, you can use a secondary index where the secondary index key is the root key. Applications, which require sequential processing, can use the secondary index when accessing the database. This allows IMS to use space according to the rules of PHDAM and HDAM databases while allowing applications to access the databases in key sequence. It is also possible to create a *sequential randomizer*, which places roots in the PHDAM or HDAM database in root key sequence. You can use the IBM product, *IMS Sequential Randomizer Generator*, to create sequential randomizers.

5.3.3 I/Os

PHIDAM and HIDAM databases have indexes. When you request a root segment by its key, IMS reads the index to find the location of the key. This processing is not required with PHDAM and HDAM databases. For them, IMS invokes the randomizer to find the root anchor point (RAP) from which the root is chained. Typically, IMS does more processing to retrieve or insert PHIDAM and HIDAM roots. This tends to give PHDAM and HDAM a performance advantage. Of course, this assumes that the PHDAM partition or HDAM database is reasonably well organized.

5.3.4 Reorganizations

Most databases have to be reorganized occasionally. The time between reorganizations usually depends on the type of insert and delete activities, the distribution of free space, and the database type. PHIDAM and HIDAM databases differ from PHDAM and HDAM databases in their reorganization characteristics. If a PHIDAM or HIDAM database has large amounts of inserts in a range of root keys without deletes in the same range, it is likely to need reorganization. This is not necessarily true with PHDAM and HDAM. Because randomizers spread the roots across the partitions or databases, activity for a range of keys tends to be spread across the entire PHDAM partition or HDAM database. PHDAM and HDAM tend to be better choices for these types of databases.

5.3.5 Creeping keys

The root keys in some applications increase in value over time. An example is a key based on the time of the insertion of the root. Such keys are called *creeping keys*.

For HDAM databases, this does not have special significance. HDAM spreads the roots randomly across the root addressable area. Deleted segments typically make room for new segments. Space use tends to be randomly distributed across the root addressable area.

For PHDAM databases using key range partitioning, space use tends to be randomly distributed across the root addressable areas of the partitions. New roots with creeping keys are placed in the last partition. You might have to create new partitions for new time periods.

You might want to avoid the requirement to add new partitions with PHDAM. If the total amount of data in the database does not increase, but the key values increase, you might want to use a partition selection exit routine instead of using key range partitioning. The exit routine can assign keys to partitions by using only a low order subset of the key. For example, if the key were eight numeric digits long and you wanted 10 partitions, you could use the low order digit to assign the key to a partition. Key zzzzzzz1 could be assigned to the first partition; key zzzzzzz2 could be assigned to the second partition; and so on with key zzzzzzz0 assigned to partition 10. Assuming that the values of the low order digit were evenly distributed from 0 to 9, the keys would be evenly distributed across the partitions. Since the database is PHDAM, deletes would create usable space for future inserts.

For HIDAM databases, IMS places new roots with creeping keys near the last ones inserted. This is at the end of the used space in the data set. When you delete the oldest database records, you make space near the beginning of the data set. IMS typically cannot use this space for new insertions. In this sense, the space used in the data set tends to creep from the front to the end of the data set. The database likely requires reorganizations. HIDAM with creeping keys has a second disadvantage. The inserts of new keys are into the last CI in the primary index. This causes CI/CA splits and disorganization of the index. This is a second reason why these databases likely require frequent reorganizations.

PHIDAM is similar to HIDAM. With key range partitioning, all inserts for creeping keys are made into the last partition. Space used in the partition data set tends to creep from the front to the end. You might need to add new partitions occasionally. This starts the process again for the new partition. With a partition selection exit routine, new roots might be placed in any of the partitions. This does not eliminate the problem of the creeping use of the data sets. New roots in a partition always go into the end of the partition. The creeping occurs in parallel across all the partitions.

PHDAM or HDAM is often a better choice than PHIDAM or HIDAM for databases whose root key values increase over time. They eliminate the problem of creeping space use. They can substantially reduce the frequency with which you must reorganize. Of course, PHDAM or HDAM do not maintain roots in key sequence. If you must process a database in key sequence, you might be able to use PHDAM or HDAM with a secondary index. The secondary index can provide the sequencing you need. The secondary index might suffer the effects of creeping keys, but the space use in the database would not creep.

5.3.6 Recommendation summary for (P)HDAM as opposed to (P)HIDAM

PHIDAM and HIDAM have the following advantages over PHDAM and HDAM:

- ▶ They tend to use less space. This provides a performance advantage for batch jobs, which sequentially process the entire database, especially if you enable OSAM sequential buffering.
- ▶ They allow you easily to process a database in root key sequence.
- ▶ To a certain extent, creeping keys can be resolved by using a Partition Selection Exit and using specific bytes of the key to allow for equal distribution across all partitions.

PHDAM and HDAM have the following advantages over PHIDAM and HIDAM:

- ▶ They tend to require fewer I/Os to retrieve and insert root segments.
- ▶ They tend to require fewer reorganizations when update activity is concentrated in a range of keys.
- ▶ They tend to handle space management with creeping root keys better.

5.4 HALDB

High Availability Large Database (HALDB) has three additional types of full function databases: PHDAM, PHIDAM, and PSINDEX. Application programs that use non-HALDB databases work without change when the databases are migrated to HALDB. HALDB provides these major advantages over non-HALDB full function databases:

- ▶ **Larger database capacity**
HALDB databases can be spread across 1 to 1 001 partitions. Each partition can have one to 10 data sets. Each data set can be up to 4 gigabytes. This means that HALDB databases have a maximum capacity of 40 terabytes.
- ▶ **Shorter offline reorganization times**
HALDB partitions can be reorganized independently and in parallel. When you reorganize a partition or database, you do not have to rebuild or update the secondary indexes that point to it. Similarly, you do not have to update any logically related databases. IMS automatically updates the secondary index pointers and logical relationship pointers when they are first used following the reorganization. These updates use the HALDB “self-healing pointer” process. The combination of reorganizing partitions in parallel and the self-healing pointer process can greatly reduce the elapsed time required for reorganizing HALDB databases with offline processes.
- ▶ **Online reorganization**
HALDB online reorganization allows an IMS online system to reorganize HALDB partitions while applications read and update the partitions. The applications can run in the same IMS online system or in data sharing subsystems.

The redbook, *The Complete IMS HALDB Guide, All You Need to Know to Manage HALDBs*, SG24-6945, provides information about migrating databases to HALDB, defining their partitions, and maintaining them.

Recommendation: Installations, which have database requirements answered by HALDB, should convert those databases to HALDB. You do not need to convert existing databases, which do not require greater capacities and do not need shorter reorganization outages. IMS intends to support non-HALDB full function database types for the foreseeable future. On the other hand, enhancements for full function databases will be concentrated in HALDB. For this reason, you should create new IMS full function databases as HALDB types.

5.4.1 HALDB partition selection

HALDB databases have one to 1 001 partitions. IMS assigns database records to partitions based on the key of the root segment. This assignment is done as part of partition selection. You specify how partition selection is done. You tell IMS to use either key ranges or a partition selection exit routine. Partition selection also determines the order in which partitions are processed by sequential processing.

5.4.2 Key range partition selection

Most HALDB databases use key range partition selection. In this method, you assign a high key to each partition. IMS assigns roots within a key range to each partition. The partitions are ordered in the order of the high keys. With PSINDEX and PHIDAM databases, sequential processing retrieves all the roots in key sequence. With PHDAM, you assign a range of keys to a partition, but the randomizer determines the order of the roots within a partition.

5.4.3 Partition selection exit routine

You can write a partition selection exit routine for any type of HALDB database. Your exit routine assigns root keys to partitions. It also determines the order of the partitions. IMS passes the key of the root segment and information about all of the partitions to the exit routine. The routine uses the key to assign the root to a partition.

These exit routines are typically used to assign database records to partitions based on a subset of the key. For example, part of the key might be a country code, department code, or similar bit of information. You could use this to group data from each country or department into its own partition. Application processing requirements usually determine if you should use this type of partitioning. For example, you might want to process only the data for a particular country or department with certain jobs. It would be more efficient if only the database records for that country or department resided in a partition. Then, your job would need to process only one of the partitions in the database.

Another use of these exit routines is to spread activity across multiple partitions. A typical example is a database where each new root has a higher key than the existing roots in the database. Key range partitioning would place these new roots in the last partition. You could use a partition selection exit routine to spread the insert activity across all of the partitions. This is further explained under 5.3.5, “Creeping keys” on page 62.

You should be careful about choosing a partition selection exit routine for PSINDEX and PHIDAM databases. The exit routine typically does not assign roots to partitions in key order. If you have chosen PHIDAM so that you can process the database in key sequence order, you cannot use an exit routine that assigns keys by other than key range. Similarly, if you have created the secondary index for secondary index key sequence processing, you cannot use an exit routine that assigns secondary index keys by other than key range.

5.4.4 Defining partition selection

A partition selection exit routine is defined with the PSNAME parameter on the DBD statement. The parameter specifies the name of the exit routine. If the PSNAME parameter is not included, key range partitioning is implied. The specification on the DBD can be changed during partition definition or with a DBRC command. The HALDB Partition Definition utility includes a “Part Selection Routine” field in the “Master Database values” section. If you specify a name, it is used as the partition selection exit routine name. If you make the field blanks or nulls, key range partitioning is used. The INIT.DB and CHANGE.DB DBRC commands include PARTSEL and HIKEY parameters. PARTSEL specifies the name of a partition selection exit routine. HIKEY specifies that key range partitioning is to be used. These parameters override the specification on the DBD.

5.4.5 Recommendation summary for HALDB partition selection

The recommendations for partition selection are:

- Key range partitioning is appropriate for most databases.

- ▶ You can use a partition selection exit routine to group data into partitions by a subset of the root key.
- ▶ You can use a partition selection exit routine to spread activity across all of the partitions in the database.
- ▶ Do not use a partition selection exit routine for a PHIDAM database that you must process in key sequence order.
- ▶ Do not use a partition selection exit routine for a secondary index that you will use for processing a database in secondary key order.

5.4.6 HALDB indirect data set lists

HALDB uses indirect data set lists (ILDS) for the self-healing pointer process. This process eliminates the need to rebuild secondary indexes or resolve logical relationships when databases are reorganized. ILDS is a VSAM KSDS data set. If you reorganize a HALDB frequently, over time, the ILDS can become poorly organized. The APAR PQ88848 in IMS Version 9 adds support for HD reload for the ILE insert to handle this case. It would save the ILE updates in a data space, then sort them, and insert them in order. This change allows reload to honor the free space specified when the ILDS KSDS was defined and allows the ILDS to have space for later updates. In IMS Version 10, there is an enhancement to DFSPREC0 utility to make a similar change.

5.5 Block sizes, CI sizes, and record sizes

You specify the block sizes of OSAM data sets, CI sizes of VSAM data sets, and record sizes of KSDSs when you create databases. This section explains the requirements and recommendations for these sizes.

5.5.1 Index CI sizes and record sizes

PHIDAM primary indexes, HIDAM primary indexes, secondary indexes, and HISAM databases use VSAM KSDSs. This section discusses record sizes and CI sizes for these KSDSs.

You do not need to specify the RECORD parameter on the DATASET statement for HIDAM primary indexes and secondary indexes. IMS automatically calculates the record sizes for these data sets during DBDGEN. They are reported in the output listing of DBDGEN. You should use these sizes for the RECORDSIZE parameter on the IDCAMS DEFINE statements when you define the data sets. If you specify a DATASET macro RECORD parameter smaller than the calculated size, it will not hold the index record and the index cannot be loaded. If you specify a RECORD parameter larger than the required size, you waste space in the data set.

The appropriate record sizes for PSINDEX data sets are reported in the DBDGEN output for these secondary indexes. You should use these sizes for the RECORDSIZE parameter on the IDCAMS DEFINE statements when you create these data sets.

You must specify the RECORD parameter on the DATASET statement for HISAM databases. As mentioned under “Hierarchical indexed sequential access method” on page 54, ideally a record in the primary data set should be large enough to contain an entire database record. In some cases, this could waste a lot of space. This is true when database records vary significantly in size. Small records will use only a small amount of the VSAM record. In this case, the primary data set record size should be large enough to hold the

majority of database records. The overflow data set record size should be large enough to hold the remainder of most other database records. Record sizes are specified on the RECORD parameter of the DATASET macro.

The CI size for the data component of index data sets (KSDSs) determines how many logical records are stored in a CI. Large CI sizes tend to be good for sequential processing. Multiple records are read or written with one I/O. Large CI sizes might not be a benefit for random processing. Typically, random processing accesses only one record per CI. For HISAM databases and indexes where sequential processing is most important, large CI sizes (16 K, 20 K, 24 K, or 28 K) should typically be used.

The size of the KSDS index component CI can be either explicitly specified in your IDCAMS DEFINE statement or allowed to default. The default causes VSAM to calculate the index CI size. If you use z/OS 1.3 or a later release, you should use the default calculation. It optimizes the index component CI size.

5.5.2 OSAM block sizes and VSAM ESDS CI sizes

OSAM block sizes and VSAM ESDS CI sizes affect the number of I/Os required to process a database and the sizes of buffers in IMS systems. Typically, block sizes and CI sizes should be large enough to hold entire database records. When database record sizes are very large or vary greatly, this might not be possible. Then, you should attempt to make the block or CI size large enough to hold the frequently accessed segments in the database records.

For sequential processing, large sizes are typically good. They reduce the number of I/Os that are required to process the database. Block or CI sizes of at least 16K should typically be used for databases which will be used with sequential processing. For random processing, large sizes are not as beneficial. They increase the overhead of I/O processing and the space required for database buffers. If a complete database record is stored in a small block or CI, there might be no benefit in using a larger size.

OSAM block size specifications

For PHDAM and PHIDAM databases, you specify OSAM block sizes during partition definition. You do this either with the Block Size field in the HALDB Partition Definition utility or with the BLOCKSIZE parameter on the DBRC INIT.PART or CHANGE.PART command.

For HDAM and HIDAM databases, you specify block sizes with the SIZE parameter on the DATASET statement of DBDGEN. Alternatively, DBDGEN calculates these block sizes from the specification of the BLOCK parameter on the DATASET statement. The value specified on the BLOCK parameter includes only space that is used for segments. DBDGEN adds space for a FSEAP and HDAM RAPs to calculate the block size from the BLOCK specification. Most users specify SIZE rather than BLOCK, because SIZE is the actual block size. Using even block sizes allows the non-HALDB OSAM data set to be 8 GB maximum. If an odd block size is used, then the data set is restricted to 4 GB maximum.

When picking a block size, make sure that it is a valid OSAM buffer size, see table Table 5-1 for a list of valid sizes and DASD utilization.

Table 5-1 Number of blocks per track on a IBM 3390 disk drive

Block size of:	Number of blocks per track	8 gigabyte limit	% utilization
512	49	22 827 CYLs	44.3
1 024	33	16 947 CYLs	59.6

Block size of:	Number of blocks per track	8 gigabyte limit	% utilization
2 048	21	13 316 CYLs	75.9
4 096	12	11 651 CYLs	86.7
6 144	8	11 651 CYLs	86.7
8 192	6	11 651 CYLs	86.7
10 240	5	11 185 CYLs	90.4
12 288	4	11 651 CYLs	86.7
14 336	3	13 316 CYLs	76.1
16 384	3	11 651 CYLs	86.7
18 432	3	10 357 CYLs	97.6
20 480	2	13 981 CYLs	72.3
22 528	2	12 710 CYLs	79.5
24 576	2	11 651 CYLs	86.7
26 624	2	10 755 CYLs	94.0
28 672	1	19 973 CYLs	50.6
30 720	1	18 642 CYLs	54.2

VSAM ESDS CI size specifications

For VSAM database data sets, IMS always uses the CI size specified in the RECORDSIZE parameter of the IDCAMS DEFINE statement. It does not use the CI size specified by the SIZE or BLOCK parameters on the DATASET statement of DBDGEN. For documentation reasons, we recommend that you specify a SIZE parameter that matches the CI size that you will use. This can eliminate confusion which might occur if the size in the DBDGEN differed from that actually used. When picking a CI size, make sure that it is a valid IMS VSAM buffer size. See Table 5-2 for a list of valid sizes and DASD utilization.

Table 5-2 Number of VSAM CONTROLINTERVALs per track on a IBM 3390 disk drive

CI size of:	Number of control intervals per track	4 gigabyte limit	% utilization
515	49	11 414 CYLs	44.3
1 024	33	8 474 CYLs	59.6
2 048	21	6 658 CYLs	75.9
4 096	12	5 826 CYLs	86.7
8 192	6	5 826 CYLs	86.7
12 288	4	5 826 CYLs	86.7
16 384	3	5 826 CYLs	86.7
20 480	2	6 991 CYLs	72.3
24 576	2	5 826 CYLs	86.7

CI size of:	Number of control intervals per track	4 gigabyte limit	% utilization
28 672	1	9 987 CYLs	50.6

If you want to change the CI size for a VSAM data set, it is not necessary to do a DBDGEN. You only have to change the CI size in the IDCAMS DEFINE before reloading the database.

HALDB partition definition does not include a capability to specify VSAM CI sizes. They are specified only with IDCAMS DEFINE statements.

5.5.3 FREQ parameter on the SEGM statement

The SEGM statement for HISAM, SHISAM, INDEX, and PSINDEX databases includes the FREQ parameter. This parameter is optional. You do not need to specify it. It is ignored for PSINDEX, INDEX, and SHISAM databases. The FREQ parameter specifies the frequency of a segment in a HISAM database. This is the average number of these segments per its parent. For root segments, the documentation states that FREQ is the number of roots in the database; however, specifying FREQ for roots does not affect the output of DBDGEN.

For HISAM, DBDGEN uses the FREQ specifications on dependent segments to calculate suggested logical record sizes and CI sizes. These are listed in the output of DBDGEN. DBDGEN always produces the same suggestions for the prime and overflow data sets. If you know the distribution of your database record sizes, you can choose logical record and CI sizes which are better for your databases. If you do not know them, you are unlikely to know the frequency of segment occurrences. For these reasons, you do not need to specify this parameter.

You never need to specify the FREQ parameter. IMS ignores it for PSINDEX, INDEX, and SHISAM databases. It is not valid for PHDAM, PHIDAM, HDAM, and HIDAM databases.

5.5.4 Recommendation summary for block sizes, CI sizes, and record sizes

The recommendations for block sizes, CI sizes, and record sizes are:

- ▶ Do not specify the RECORD parameter on the DATASET statement for INDEX databases (secondary indexes and HIDAM primary indexes).
- ▶ Always use the output listing of DBDGEN to determine the RECORDSIZE parameters to use for the IDCAMS DEFINE for KSDSs. These include data sets for PSINDEX, INDEX, PHIDAM, HIDAM, and HISAM databases.
- ▶ If you use z/OS 1.3 or a later release, you should specify the data component CI size for KSDSs. Do this with an IDCAMS DEFINE statement. Do not specify the index component CI size. Let the system calculate it.
- ▶ The CI size for VSAM ESDS and KSDS data sets is not controlled by DBDGEN parameters. It is determined by the RECORDSIZE parameter on the IDCAMS DEFINE statement. To avoid confusion, specify the SIZE parameter on the DATASET statement for VSAM data sets to match the CI size specified with IDCAMS.
- ▶ For HDAM and HIDAM OSAM data sets, specify the SIZE parameter on the DATASET statement, not the BLOCK parameter.
- ▶ Do not specify the FREQ parameter on SEGM statements.

5.6 Free space

Free space in a database affects the processing done by IMS to retrieve and insert segments. Free space in a database increases the likelihood that an insert of a segment will go in the same block as the segment from which it is chained. This tends to reduce the number of I/Os required for inserting and retrieving segments. On the other hand, adding free space increases the number of blocks that must be read for sequential processing, because it spreads data across more blocks.

Free space is created when a database is initially loaded, when it is reorganized, and when segments are deleted.

5.6.1 Specifying free space

Free space in HIDAM databases is specified with the FRSPC parameter on the DATASET statement. The first subparameter specifies the *free block frequency factor* (FBFF). A value of “x” specifies that every xth block in the data set is left as free space by a load or reorganization. The second subparameter specifies the *free space percentage factor* (FSPF). A value of “y” specifies that y percent of each block is left as free space by a load or reorganization. Different data sets in the same database can have different free space specifications.

Free space in PHIDAM databases is specified in the partition definitions. These parameters are the free block frequency factor and the free space percentage factor that are described in the previous paragraph. Each data set in a partition uses the same free space parameters. The values are specified either in the HALDB Partition Definition utility or with the FBFF and FSPF parameters of the DBRC INIT.PART and CHANGE.PART commands.

Free space in HDAM and PHDAM databases can be specified by using the same parameters used with HIDAM and PHIDAM; however, you should not do this. Instead, you should use other parameters to control how space is used in HDAM and PHDAM databases. If you specify a free block frequency factor, a reorganization cannot place any database records in the free blocks even though some records would be chained from the RAPs in these blocks. If you specify a free space percentage factor, each block in the RAA would have space that could not be used by a reorganization. Some root segments as well as their dependents could be forced to the overflow area. A better way to specify the free space characteristics for these databases is through the use of randomization parameters. Randomization parameters are explained in 5.7, “Randomization parameters” on page 71.

Free space in PHIDAM primary indexes, HIDAM primary indexes, and secondary indexes is specified with the FREESPACE parameter on the IDCAMS DEFINE statements. It is not defined in DBDs or HALDB partition definitions. The FREESPACE parameter has two subparameters. The first specifies the percentage of free space in each CI. The second specifies the percentage of free space in each CA. You should specify free space when you expect to insert new root segments in PHIDAM and HIDAM databases or when you expect to insert new secondary index entries.

The recommended amount of free space depends on the amount of inserts. For example, if you expect to insert 5% more root segments in a HALDB partition between reorganizations of the index, you should specify at least 5% free space in the CIs. You should also specify free space in the CAs. This creates empty CIs in the CA which are available for splitting CIs without forcing the split of the CA. Free space is created uniformly across the data set. If the inserts will be concentrated in a small part of the key range, free space might not be beneficial. In this case, most of the free space will not be used. Instead, VSAM will have to

continually split CIs and CAs to create room for the new entries. Unfortunately, you cannot do much to reduce these splits.

5.6.2 HD space search algorithm

IMS has an HD space search algorithm. IMS uses this algorithm when inserting segments in PHDAM, PHIDAM, HDAM, and HIDAM databases. The algorithm is documented in *IMS Version 9: Administration Guide: Database Manager*, SC18-7806, under “How the HD Space Search Algorithm Works.” In general, IMS attempts to place a new segment in the same block or CI with the segment or RAP from which it is chained. If there is not space in this block or CI, IMS may attempt to place the segment in the second most desirable block. This is the nearest block or CI that was left free when the database or partition was loaded or reorganized.

The search does not use the second most desirable block in two cases. First, if free block frequency is not specified in the database or partition definition, there is no second most desirable block. Second, HDAM and HIDAM DBD definitions can override the use of the second most desirable block in the space search. This is done by specifying SEARCHA=1 on the DATASET macro. Since the purpose of specifying the free block frequency factor is to create free blocks or CIs for later inserts, SEARCHA=1 should not be specified except for HDAM. This parameter is included to provide compatibility with old IMS releases that did not have the second most desirable block step in the HD space search algorithm.

There is no option equivalent to SEARCHA=1 for PHDAM and PHIDAM databases. They always include the search of the second most desirable block when the free space percentage factor is specified.

5.6.3 Recommendation summary for free space

The recommendations for free space are:

- ▶ Use the “free block frequency factor” and/or the “free space percentage factor” to create free space in HIDAM and PHIDAM databases.
- ▶ Use randomization parameters to create free space in HDAM and PHDAM databases. Do not specify fbff and fspf for them.
- ▶ Use the IDCAMS DEFINE FREESPACE parameter to create free space in PHIDAM primary indexes, HIDAM primary indexes, and secondary indexes.
- ▶ Specify SEARCHA=1 on the DATASET macros for HDAM databases.
- ▶ Specify SEARCHA=2 on the DATASET macros for HIDAM databases.

5.7 Randomization parameters

Randomizers and their parameters are specified in the DBD for HDAM. The DBD for PHDAM only specifies the default values. The definition of the PHDAM partition specifies the values used for the partition. Different partitions in a PHDAM database can have different randomization parameter values.

The randomization parameters on the DBD statement have the following syntax:

- ▶ RMNAME=(*mod*, *anch*, *rbn*, *bytes*)
- ▶ *mod* is the randomizer module name. It must be specified.

- ▶ *anch* is the number of root anchor points per block. It defaults to 1. The maximum value is 255.
- ▶ *rbn* is the number of blocks in the root addressable area. It must be specified.
- ▶ *bytes* specifies the maximum number of bytes of a database record that can be stored in the root addressable area in a series of inserts without a call to another database record. The default is no maximum.

For PHDAM, these values can be defined with the HALDB Partition Definition utility (PDU) or the DBRC INIT.PART or CHANGE.PART command. The DBRC command parameters are:

- ▶ RANDOMZR(*name*) is the randomizer module name.
- ▶ ANCHOR(*value*) corresponds to the anch parameter in the DBD.
- ▶ HIBLOCK(*value*) corresponds to the rbn parameter in the DBD.
- ▶ BYTES(*value*) corresponds to the bytes parameter in the DBD.

5.7.1 Randomizer

IMS supplies several randomizers. DFSHDC40 is appropriate for most databases and HALDB partitions. We recommend you use DFSHDC40, unless you have specific knowledge of your key distribution and you *do not* want the keys randomly spread across the RAPs. DFSHDC40 spreads roots randomly across the RAPs in a HDAM database or a PHDAM partition. You do not need to do any analysis of key distributions when you use DFSHDC40.

DFSHDC40 converts the key into a randomly chosen four-byte binary number. This number is chosen independently of the number of RAPs in the database or partition. Then, DFSHDC40 multiplies this number by the number of RAPs in the database or partition and divides the result by the maximum value. This assigns keys to RAPs in an order that is unaffected by the number of RAPs. So if you unload a database or partition and change the number of RAPs, the reload is still a sequential process.

In some cases, you might want to use a different randomizer. This should only occur when you have knowledge of the key distribution and you do not want a random distribution of the keys across the database or partition. For example, you might want to assign the keys to RAPs in key sequence. The IBM product, *IMS Sequential Randomizer Generator*, can create randomizers that do this.

5.7.2 Number of RAPs

The total number of RAPs for a HDAM database or PHDAM partition is the product of the number of RAPs per block times the number of blocks in the root addressable area. When more RAPs are used, the probability of a long chain of roots from a RAP is diminished. A good general rule is that the total number of RAPs in a database or partition should be at least twice the number of roots. When DFSHDC40 is used, the roots are distributed across the RAPs in a Poisson distribution. When the total number of RAPs is twice the number of roots, the distribution has the following characteristics:

- ▶ 79% of roots are the first root on their RAP chain.
- ▶ 18% of roots are the second root on their RAP chain.
- ▶ 3% of roots are the third root on their RAP chain.
- ▶ 0.38% of roots are past the third root on their RAP chain.

When the total number of RAPs is three times the number of roots, the distribution has the following characteristics:

- ▶ 85% of roots are the first root on their RAP chain.
- ▶ 13% of roots are the second root on their RAP chain.

- ▶ 1.5% of roots are the third root on their RAP chain.
- ▶ 0.13% of roots are past the third root on their RAP chain.

Since a RAP is only four bytes, the space used by RAPs is rarely significant.

5.7.3 Size of root addressable area

The root addressable area (RAA) is the set of OSAM blocks or VSAM CIs that holds RAPs. Other blocks in the data set contain the overflow area. When the RAA is too small, the segments in a database record are often stored in overflow. Reading them requires reading the RAP block and reading blocks in the overflow area. When the RAA is too large, there is unnecessary unused space in the RAA. A sequential read of the database or partition has to read more blocks. A good general rule is that the RAA should be 35% larger than the size of all of the segments in the HDAM database or PHDAM partition. This creates approximately 25% free space. Larger sizes may be desirable because they may help avoid the need for reorganizations. They increase the probability that IMS will place segments in the block containing the RAP from which they are chained.

5.7.4 The BYTES parameter

The bytes parameter specifies the maximum number of bytes of a database record that may be stored in the root addressable area in a series of inserts without a call to another database record. This includes inserts that are done by reorganizations. When you reorganize a database or partition any segment which causes the database record to exceed this parameter is placed in overflow. This reserves space for segments of other database records. It helps avoid situations where exceptionally large database records force other database records into overflow. A good general rule is that the bytes parameter should be twice the average database record size. This assumes that all of the segments in the database are in the first data set group. This general rule might not apply to some databases. If you are aware of your database record size distributions, you might want to use other values for bytes. The bytes parameter does not affect any segments that are in other data set groups.

5.7.5 Specifying randomization parameters for PHDAM

The randomization parameters used for PHDAM partitions are defined in the partition definitions. The DBD for PHDAM only specifies the default values. If you change a PHDAM DBD, the randomization values used by its partitions are not changed. To modify the parameters for a partition, you must make changes to the partition definition, which is stored in the RECONs. You can do this with either the HALDB Partition Definition utility (PDU) or a DBRC CHANGE.PART command.

5.7.6 Recommendation summary for randomization parameters

The general recommendations for randomization parameters follow. You might be aware of reasons for using different values. We recommend:

- ▶ Use DFSHDC40 for the randomization module.
- ▶ The total number of RAPs in a database or partition should be at least twice the number of roots.
- ▶ The root addressable area (RAA) size should be 35% larger than the size of all of the segments in the first data set of an HDAM database or a PHDAM partition.
- ▶ The bytes parameter should be twice the average database record size.

5.7.7 Monitoring HDAM databases

To monitor a HDAM database, periodically run a pointer checker utility, such as IBM *IMS High Performance Pointer Checker for z/OS*, depending on database activity, to check the packing density of the database.

90% or more of the roots should be in their home block (HB). This just means that the roots are in the block to which they actually randomized. Look at Example 5-1 for reference.

10% or less of roots in the HB - 1 or HB + 1 is acceptable.

If roots are present in "BEYOND" or "OVERFLOW", the database randomizing parameters should be tuned, and then the database reorganized. When HDAM attempts to retrieve a root, it takes the root's key and passes it through a hashing algorithm and "randomizes" it. It then comes up with the RAP in the block that the root is supposed to be in. IMS then goes to that block and looks for the key. If the root is not present in that block, then it must chain through subsequent blocks until it finds it. If roots are not stored close to the home block, I/O can become a problem.

Example 5-1 High performance pointer checker: distribution of root segments

IMS HIGH PERFORMANCE POINTER CHECKER FOR z/OS "DB RECORD DISTRIBUTION STATISTICS REPORT"
5655-K53 DATE: 09/16/2006 TIME: 18.15.10 FABPMAIN

DBNAME: DHZBCP1 DB#: 00E DSG#: 01 DDNAME: DHZBCP1 DSNAME: PGMZ1.DHZBU.DHZBCP1.G5186V00

TOTAL NUMBER OF SEGMENTS (ROOTS + DEPENDENTS) IN THE DATA SET = 539313
MAXIMUM ROOTS PER BLOCK = 12

DISTRIBUTION OF ROOT SEGMENTS (HDAM/PHDAM ONLY)

LOCATION	NUMBER OF ROOTS	PERCENTAGE
HOME BLOCK - (11-)	10	0.0 %
HOME BLOCK - 10	0	0.0 %
HOME BLOCK - 9	0	0.0 %
HOME BLOCK - 8	0	0.0 %
HOME BLOCK - 7	0	0.0 %
HOME BLOCK - 6	0	0.0 %
HOME BLOCK - 5	0	0.0 %
HOME BLOCK - 4	0	0.0 %
HOME BLOCK - 3	0	0.0 %
HOME BLOCK - 2	0	0.0 %
HOME BLOCK - 1	0	0.0 %
HOME BLOCK - 1	0	0.0 %
HOME BLOCK	189,795	100.0 %
HOME BLOCK + 1	0	0.0 %
HOME BLOCK + 2	0	0.0 %
HOME BLOCK + 3	0	0.0 %
HOME BLOCK + 4	0	0.0 %
HOME BLOCK + 5	0	0.0 %
HOME BLOCK + 6	0	0.0 %
HOME BLOCK + 7	0	0.0 %
HOME BLOCK + 8	0	0.0 %
HOME BLOCK + 9	0	0.0 %
HOME BLOCK + 10	0	0.0 %
HOME BLOCK + (11-)	8	0.0 %
OVERFLOW	0	0.0 %

TOTAL 189,813 100.0 %

In order to tune the database you need to know how long the average database record is. In Example 5-2, the average database record length for the DHZDBCP1 database is 159 bytes.

Example 5-2 High performance pointer checker: database record statistics

IMS HIGH PERFORMANCE POINTER CHECKER FOR z/OS "DATABASE STATISTICS REPORT"													
5655-K53					DATE: 09/16/2006			TIME: 18.15.10		FABPMAIN			
DBNAME: DHZDBCP1 DB#: 00E													
DATABASE RECORD STATISTICS													

SEGMENT					<---- SEGMENT LENGTH ---->		<-AVERAGE OCCURRENCES->		AVG LENGTH	CUMULATIVE			
SC	LV	DG	NAME	OCCURRENCES	PRFX	+	DATA	=	TOTAL	PER ROOT	PER PARENT	/DB RECORD	LENGTH
					----		-----		-----			-----	-----
01	01	01	DHZSBC00	189,813	10		55.0		65.0	1.0		65.0	65.0
02	02	01	DHZSBCMD	349,500	6		45.0		51.0	1.8	1.8	93.9	158.9
--					----		-----		-----			-----	-----
TOTALS				539,313	AVERAGE DB RECORD LENGTH = 158.9								
					AVERAGE DB RECORD PREFIX LENGTH = 21.0								
NOTE : 'V' INDICATES THAT 'DATA' SHOW AVERAGE VALUES FOR A VARIABLE LENGTH SEGMENT (IF ANY)													

In Example 5-2, the average database record length is 158.9 bytes on 189,813 root segments. If you have a tool that shows a histogram of the database record length with a running cumulative total and a percentage of total roots, then you would be able to find that the 95th percentile of the records are this size, say 300 bytes, and tune for that size and not the average database record length. See Example 5-3.

Example 5-3 Histogram of database record sizes

HISTOGRAM OF DATABASE RECORD LENGTH				
BYTES	OCCURRENCEY	%	CUM %	
- 100	0	0.0%	0.0%	
101 - 200	1,035,115	66.9%	66.9%	
201 - 300	234,918	15.2%	82.1%	
301 - 400	100,143	6.5%	88.6%	
401 - 500	31,870	2.1%	90.7%	
501 - 600	30,262	2.0%	92.6%	
601 - 700	15,548	1.0%	93.6%	
701 - 800	15,247	1.0%	94.6%	
801 - 900	25,592	1.7%	96.3%	
901 - 1,000	26,474	1.7%	98.0%	
1,001 - 2,000	25,565	1.7%	99.6%	
2,001 - 3,000	5,573	0.4%	100.0%	
3,001 - 4,000	6	0.0%	100.0%	
4,001 - 5,000	0	0.0%	100.0%	
5,001 - 6,000	0	0.0%	100.0%	
6,001 - 7,000	2	0.0%	100.0%	
7,001 - 8,000	0	0.0%	100.0%	
AVERAGE DATABASE RECORD SIZE = 25				

In the histogram in Example 5-3 on page 75, the average database record length is 251 bytes. But the 95th percentile would be 900 bytes. We recommend that you tune for the 95th percentile.

5.7.8 Loading or reloading HDAM databases

Blocks must be preformatted in HDAM prior to IMS storing records in them. When the first record is inserted into a block, that block, along with all prior blocks, is formatted. This can be a source of significant I/O. To avoid this situation, a high-value (x'FFFF') record can be inserted to ensure all of the blocks are formatted prior to the user accessing the database. This should always be done if the randomizer used is DFSHDC40.

When reloading, if you are reallocating the database equal to or smaller than the original, all of your blocks should already be preformatted so that you do not have to do anything additional. If you are reallocating it larger, or the database was incorrectly loaded the first time, a high-value record needs to be loaded. Check with the programmers for that database to determine if you have to delete the high-value record after you are done, or whether their programs check for it and ignore it.

If, all of a sudden, an unusually high number of roots appear in overflow, make sure that a job did not run (DL/I) using a DBD that has a different RAA size (test DBD), which added records out to overflow. You cannot access these. HD reload probably ABENDs with an "LB" (duplicate keys). HD unload picks up the records, because unload does not use the DBD; it just sequentially unloads the segments.

5.8 Fixed length as opposed to variable length segments

You can define segments as either fixed length or variable length. You define a fixed length segment by specifying one value for the *BYTES=* parameter on the SEGM statement for the segment. You define a variable length segment by specifying two values for the *BYTES=* parameter. A specification of *BYTES=(2000,100)* defines a variable length segment with a maximum size of 2 000 bytes and a minimum size of 100 bytes. The number of bytes specifies the size of the data portion of the segment. It does not include the size of the segment prefix.

The maximum size for a segment is limited by the record size of the data set, which contains the segment type. The minimum size of variable length segments is the minimum size stored in the data set. If the segment is sequenced, the minimum size must be large enough to include the key field with one exception. If the segment is compressed and key compression is used, the minimum size can be as small as four bytes.

Variable length segments begin with a two-byte length field. Programs use this field to determine the size of the segment. The size specified by an application program can be smaller than the minimum size. For example, a program might create a segment with "20" in the length field even when the minimum size is 30 bytes. In this case, IMS stores an extra 10 bytes in the data set. Later retrievals of the segment return a 20 byte segment to application programs. The minimum size that a program specifies must be large enough to include the key field for sequenced segments.

If you have a variable amount of data that could be placed in one segment, you have two basic choices in your database design. You could use a variable length segment or you could use a fixed length segment large enough to hold the maximum amount of data. The following explains the advantages and disadvantages of these choices.

5.8.1 Variable length segment

A variable length segment is the natural way to store a variable amount of data. It uses the minimal amount of space but could require extra I/Os. If a replace call increases the size of a segment, the larger segment might not fit in the same space that the smaller segment occupied. For HISAM, this could cause the segment to move to another logical record in the overflow data set. It could also force IMS to move other segments in the database record. This was discussed under “Hierarchical indexed sequential access method” on page 54. This makes variable length segments less attractive for HISAM databases.

For HD access methods, IMS does not move the prefix when you replace a segment. If the larger segment does not fit in the previous location, it uses another technique. It splits the segment by moving the data portion of the segment elsewhere and adding a pointer in the prefix to the data portion. This was discussed under 5.2.3, “HD access methods” on page 60. Splitting the segment can cause extra I/Os because the prefix and the data portions of a segment might be in different blocks.

If you do not change the length of the segment after you insert it in the database, you should define a variable length segment. This results in optimal space usage. Because you do not change the length, you avoid the overhead caused by movement in HISAM or by split segments with HD access methods. If you change the length of the segment, your choice is not obvious. A variable length segment causes extra overhead for moving the segment, but it optimizes the use of space. A good practice with variable length segments is to specify a minimum size, which avoids the splitting of segments by most replace calls. For example, you could make the minimum size large enough to hold 80 or 90 percent of the segments. Only the very large segments would be subject to being split.

5.8.2 Fixed length segment

A fixed length segment must be large enough to hold the maximum amount of data for a segment. The importance of this disadvantage depends on the range for the segment. If the maximum amount of data is much larger than the typical amount, this can have a significant effect. It could make database records span multiple blocks unnecessarily. Fixed length segments are easily handled by replace calls. If a replace call adds more information to the segment, it does not grow in size. So the replaced segment can always fit in the previously occupied space. This avoids the HISAM movement or HD split segment considerations.

If you update the segment frequently, you might want to use a fixed length segment. This depends on the trade-off between the extra space required for the segment as opposed to the potential I/O savings.

Fixed length segments, which are compressed, have the space usage characteristics of variable length segments. Replacements of these segments can force movement of them in HISAM or split segments with HD access methods. See 5.12, “Compression” on page 84 for a further explanation of compression.

5.8.3 Recommendations for fixed as opposed to variable length segments

The recommendations for choosing between fixed length and variable length segments are:

- ▶ If segments are rarely replaced and contain variable amounts of data, you should define them as variable length.
- ▶ If segments are frequently replaced and the amount of data stored in the segments varies within a small range, you should define the segments as fixed length.

- ▶ If segments are frequently replaced and the amount of data stored in the segment varies over a large range, your choice between fixed length and variable length segments is a trade-off between space use and I/O requirements. You can use a minimum size to lessen the number of split segments.
- ▶ If a variable length segment is frequently replaced, consider specifying a minimum size that accommodates most segment occurrences.
- ▶ If a HISAM segment is frequently replaced with a different size segment and other segments follow it in the hierarchy, do not define it as variable length. Either use an HD access method or define the segment as fixed length large enough to hold the maximum size.

5.9 Pointer options

IMS allows you to specify the types of pointers that are used between segments in HDAM, PHDAM, HIDAM, and PHIDAM databases. Only use the pointers that are needed. The fewer the number of pointers, the less maintenance that IMS has to perform, and therefore, the faster the application. This section provides advice about choosing the pointers for certain types of segments.

5.9.1 Hierarchic as opposed to child and twin pointers

IMS provides two types of pointer schemes: hierarchic pointers or child and twin pointers.

Hierarchic pointers can be specified as either forward or both forward and backward. A hierarchic forward pointer points to the next segment in the database record hierarchy. Hierarchic backward pointers point from a dependent segment to the previous dependent segment in the database record.

Figure 5-2 shows an example of hierarchic pointing with forward pointers. Each dependent segment has only one pointer. It points to the next segment in the hierarchy. To get from the root segment A to its first D child segment, IMS must traverse all of the B and C segments that are children of A. Root segments have a hierarchic pointer to their first dependent segment and another pointer to the next root. Hierarchic backward pointers point from a dependent segment to a previous dependent segment.

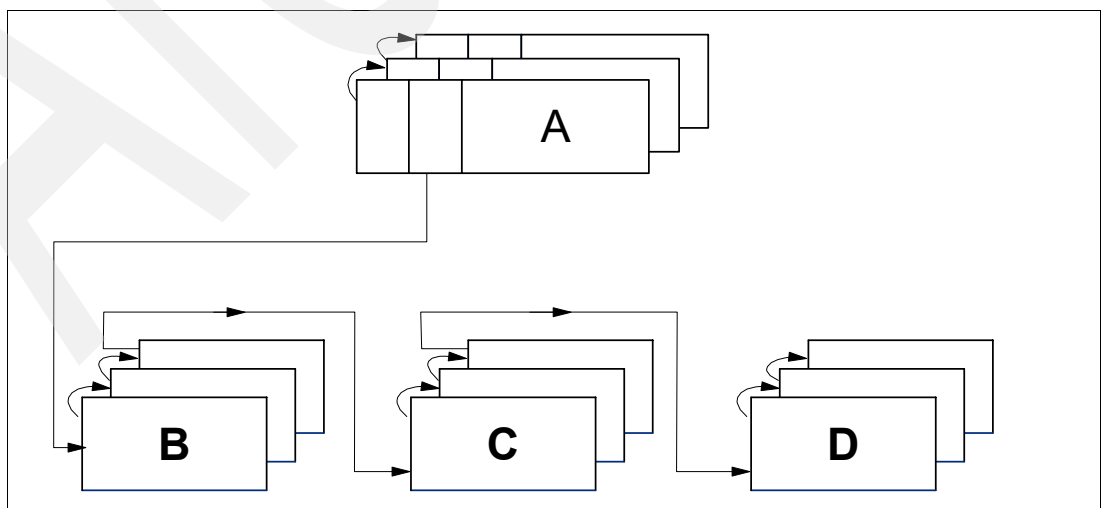


Figure 5-2 Hierarchic pointers

Child and twin pointers are an alternative to hierarchic pointers. Child pointers can be either child first or child first and last. A child first pointer in a segment points to the first child of a segment type. A child last pointer points to the last child of a segment type. If a segment has multiple children types, it has multiple child first pointers and, possibly, multiple child last pointers. Twin pointers can be either forward or both forward and backward. They point to other segments of the same type, which are under the same parent.

Figure 5-3 illustrates the use of child first and twin forward pointers. Segment A has three child first pointers. The first points to the first B segment under A. The second points to the first C segment. The third points to the first D segment. Twin pointers point to the next segment of the same type. There are twin pointers between root segments and between dependent segments.

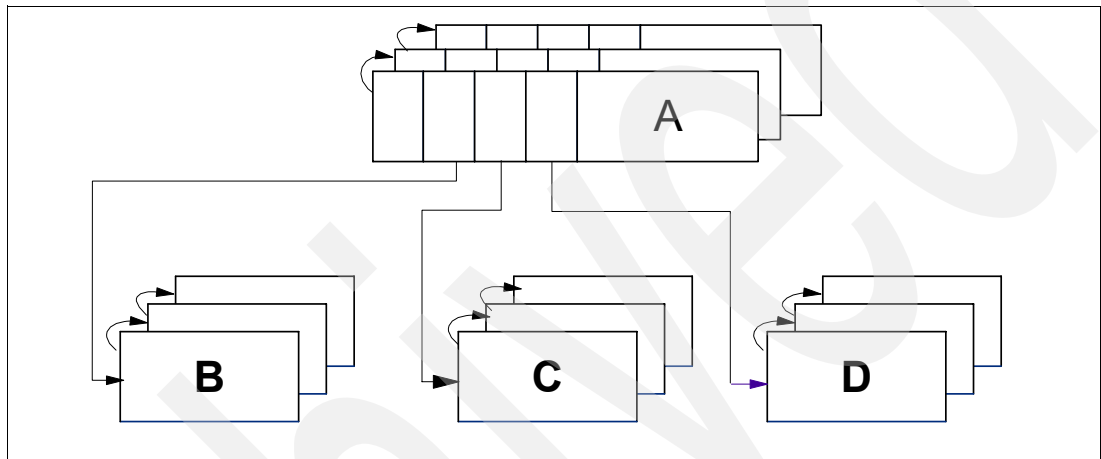


Figure 5-3 Child and twin pointers

For most IMS databases, child and twin pointers are preferable to hierarchic pointers. Hierarchic pointers take less space, because each dependent segment has only one pointer and each root segment has only two pointers. They have the disadvantage of usually requiring more accesses to find the desired segment. This is especially true when there are a large number of segments in a database record. Hierarchic pointers are not available with PHDAM and PHIDAM databases.

5.9.2 Forward only as opposed to forward and backward pointers

Hierarchic and twin pointers can be forward only (one forward pointer) or both forward and backward (a forward pointer and a backward pointer). Backward pointers are useful when segments are deleted from a chain and the previous segment in the chain has not been accessed. This can occur when a segment is entered through a logical relationship. If the backward pointer does not exist, IMS must find the previous segment in the twin chain or hierarchic chain. It does this by following the parent pointer to the segment's parent and following either the hierarchic or child and twin pointers until it reaches the previous segment in the chain. It recognizes the previous segment, because its pointer points to the segment being deleted.

The inclusion of backward pointers requires more space and more processing for inserts and deletes. Both the previous and next segments in the chain must have their pointers adjusted when a segment is inserted or deleted. Because the only need for backward pointers is for deletion by a logical path, backward pointers should not be used unless the database satisfies all of the following conditions:

- The database participates in a logical relationship.

- ▶ Segments are deleted using the logical relationship path.
- ▶ There are typically many segments in the chain which contains the segment to be deleted.

5.9.3 HIDAM and PHIDAM root segments

For HIDAM root segments, you can specify no twin pointers, forward only twin pointers, or both forward and backward twin pointers. When you specify no twin pointers, roots are accessed using the index. When you specify both forward and backward twin pointers, roots are accessed by the twin forward pointer when proceeding from a previous root segment with get next (GN) processing. This eliminates the need for accessing the index. When you specify forward only twin pointers, they are not used for accessing segments. IMS maintains them, but does not use them. This creates overhead with no benefits; therefore, you should never define forward only pointers for HIDAM root segments. PHIDAM does not allow the specification of forward only twin pointers on root segments. It only allows either no twin or both forward and backward twin pointers.

When you specify both forward and backward twin pointers for HIDAM or PHIDAM roots, IMS must maintain the twin chain when roots are inserted and deleted. It must adjust the pointers in the previous and following roots. This overhead can be significant when there are frequent insertions and deletions of roots.

The overhead of using the index to access the next root in a HIDAM database or PHIDAM partition *might* not be significant. An index CI typically holds many index entries. This means that accessing the next root typically does not require reading another index CI. For this reason, you should not specify twin forward and backward pointers for most HIDAM or PHIDAM root segments.

If you are always adding the next higher key to the database, then TB is a better choice than NOTWIN, because of how IMS locks on the high-values record and the highest key in the database. If the updates are always somewhere in the middle of the database, then NOTWIN is a better choice.

There is a reason why some installations define twin forward and backward pointers on (P)HIDAM roots. If the index is damaged, the use of twin forward and backward pointers allows you to unload the database or partition if the first pointer in the index is good. Unload uses the twin forward pointers to find successive roots in the database. You can use the unload file to reload the database. This recreates the index. Without these pointers, roots must be found with the index. This would prevent the successful unloading of the database with a damaged index. In this case, you would need to recover the index from an image copy and log records or you would need to rebuild the index by using a tool, such as *IBM IMS Index Builder for z/OS*.

5.9.4 Unsequenced dependent segments

When you define a dependent segment, you have the option of specifying a sequence field. IMS maintains segments with sequence fields in key sequence. IMS orders segments without sequence fields by a combination of the rules defined for the segments and the order in which application programs insert the segments.

Some applications rely on segments being returned in key sequence. Others do not depend on this sequencing. If applications do not require that segments are in key sequence, it is best not to specify a sequence field. Then IMS does not necessarily have to follow the twin chain when inserting the segment.

For segments without sequence fields, the location of the inserted segment depends on the last subparameter of the *RULES=* parameter on the SEGM statement for the segment. The possible values are FIRST, LAST, and HERE. LAST is the default. HERE causes IMS to insert the segment at the current location. This is the least overhead. FIRST causes IMS to insert the segment at the beginning of the twin chain. IMS finds this location with minimal overhead, because there is a pointer in the parent to the beginning of the twin chain. LAST can cause significant overhead with long twin chains. You can avoid this by specifying a Physical Child Last pointer from the parent to this segment type. Then IMS can go directly to the last segment in the twin chain. So if you specify LAST or use it as the default, you should define a physical child last pointer from the parent when there might be a long twin chain. This advice only applies to segments without sequence fields.

5.9.5 Defining hierarchical, physical twin, and physical child pointers

You define hierarchic pointers by specifying PTR=H or PTR=HB on SEGM statements. Use PTR=HB to specify both forward and backward pointers.

You define twin pointers by specifying PTR=T or PTR=TB on SEGM statements. Use PTR=TB to specify both forward and backward pointers.

You define physical child pointers in the SEGM statement for the child. The second subparameter of the PARENT parameter is either SNGL or DBLE. SNGL causes IMS to place a physical child first pointer in the parent segment. DBLE causes IMS to place both a physical child first (PCF) and a physical child last (PCL) pointer in the parent segment. The PCL pointer is helpful if there are a significant number of Segment Search Arguments (SSAs) that use the last command code, “*L”. SNGL is the default. Example 5-4 shows an illustration of a SEGM statement for a child segment whose parent will contain both physical child first and physical child last pointers to it.

Example 5-4 Definition of physical child first and physical child last pointers

```
SEGM NAME=EXPR,BYTES=20,PTR=T,PARENT=((NAME,DBLE))
```

5.9.6 Recommendation summary for pointer options

The recommendations for pointer options are:

- ▶ Use child and twin pointers instead of hierarchic pointers.
- ▶ Do not specify twin backward pointers for dependent segments unless you satisfy the criteria for deletes with logical relationships.
- ▶ Never specify twin forward only pointers for HIDAM roots.
- ▶ Never specify twin forward and backward pointers for HDAM roots.
- ▶ Specify no twin pointers for HIDAM and PHIDAM roots.
- ▶ If you specify RULES=(,LAST) or use last as the default for segments without sequence fields, you should define a physical child last pointer from the parent if there might be a long twin chain.

5.10 SCAN= parameter on the DATASET statement

The *SCAN=* parameter on the DATASET statement affects searches for free space. It is the number of cylinders that IMS scans when searching for space to insert a segment in a HDAM or HIDAM database. When a segment is inserted, IMS tries to place it in the “most desirable

block.” Generally, this is the block that contains the segment or RAP from which the inserted segment will be chained. Exceptions are explained in the “How the HD Space Search Algorithm Works” section of the *IMS Version 9: Administration Guide: Database Manager*, SC18-7806, manual. If IMS does not find space in this block, it uses the HD space search algorithm to find space in another block. In general, the algorithm tries to find space in a nearby block on the same cylinder. If it does not find available space there, the SCAN parameter limits the number of adjacent cylinders that IMS examines when it looks for space. If IMS does not find space within this limit, it inserts the segment at the end of the data set.

SCAN=0 causes IMS to search for blocks in the buffer pool which are on the cylinder containing the most desirable block. SCAN=1 causes IMS to scan the pool for blocks on the two adjacent cylinders also. SCAN=2 causes IMS to scan the pool for blocks on the two cylinders which are on either side of the cylinder containing the most desirable block. IMS must scan its buffer pool once for each cylinder within the scan range. So with SCAN=3, IMS might scan the buffer pool seven times, once for each cylinder within the limit.

You should always specify SCAN=0. The default is SCAN=3. The default was set many years ago when cylinders were much smaller and buffers pools contained fewer buffers. Today, a 3390 cylinder holds over 0.5 megabytes, and many buffer pools have thousands of buffers. Multiple scans of these pools can be costly. In any case, there is no particular advantage in putting the segment two or three cylinders away. It can just as easily be handled when placed at the end of the data set.

You cannot specify SCAN for HALDB databases. The space search algorithm for them always operates as though SCAN=0 were specified.

Recommendation summary for the SCAN parameter

Always specify SCAN=0 on DATASET statements for HDAM and HIDAM databases.

5.11 Multiple data set groups

PHDAM, PHIDAM, HDAM, and HIDAM databases can have multiple data set groups. You can use multiple data set groups to store different segment types in different data sets. You can store multiple segment types in the same data set, but all instances of the same segment type are stored in the same data set. Figure 5-4 on page 83 is an illustration of the use of multiple data set groups.

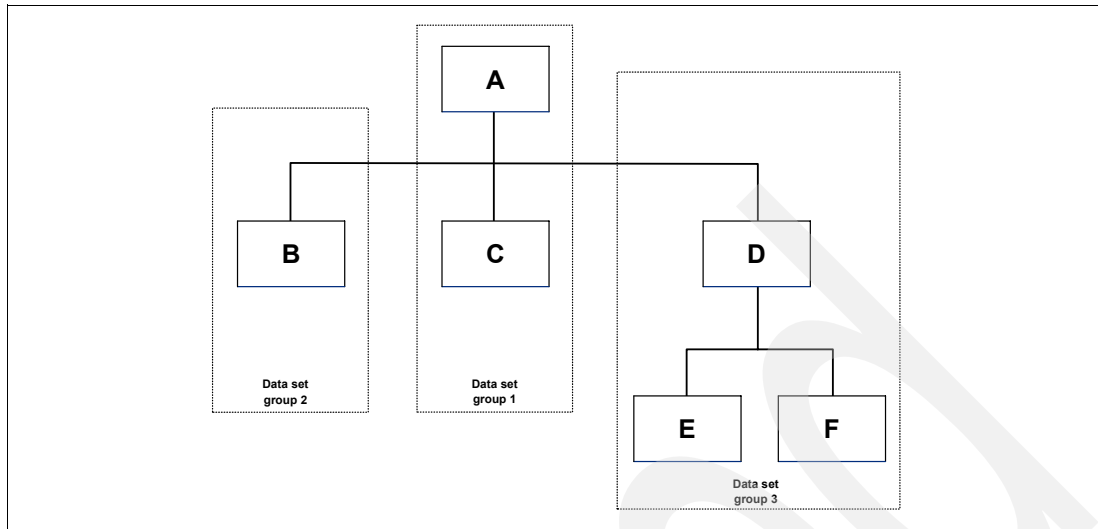


Figure 5-4 Multiple data set groups

In this illustration, segment types A and C are stored in the first data set group. Segment type B is stored in data set group 2. Segment types D, E, and F are stored in data set group 3.

You can use data set groups for various purposes.

First, you can use them to move infrequently used segments to their own data sets. Then the other segments in the database are stored in fewer blocks and might require fewer I/Os to access. This could be especially important in sequential jobs which read all of some segment types but do not read other segment types. In Figure 5-4, data set group 3 might have been created for this purpose.

Second, you can create data set groups to reduce the number of blocks which are read to access a segment. In Figure 5-4, data set group 2 might have been created for this purpose. If there are many instances of segment B, moving segment B to its own data set group increases the probability that the C segments will be in the same block with the root segment. This is useful if C segments are frequently accessed, but B segments are rarely accessed.

Third, you can use data set groups for space management reasons. With HD access methods, IMS keeps a bitmap indicating which blocks have room for the largest segment allowed in the data set. The space search algorithm uses this bitmap when looking for space for inserts. Segment types with very large maximum sizes can make the bitmap misleading. The bitmap might indicate that there is not space in a block for the largest segment in the data set, but there might be plenty of free space for smaller segments. Some shops isolate extremely large segment types in their own data set. This is beneficial for space search in other data sets. The bitmaps in the other data sets are not skewed by the large segment types. This technique is especially beneficial when there are lots of inserts of the smaller segment types.

Fourth, you can use data set groups to provide more capacity for HDAM or HIDAM databases. You cannot partition these databases. If you use only one data set, the database would be limited to 8 gigabytes with OSAM or 4 gigabytes with VSAM. Because you can define up to 1 001 partitions for PHDAM and PHIDAM databases, you do not need multiple data set groups to provide database capacity for them. Usually, it is preferable to convert a HDAM database to PHDAM or a HIDAM database to PHIDAM when additional capacity is needed.

You need to know the access patterns that applications use before you can make the best decisions about the use of multiple data set groups. Multiple data set groups spread database records across blocks in different data sets. If you use them unnecessarily, you might cause IMS to do additional I/Os. You should define multiple data groups only when you have a reason to do so.

For HDAM and HIDAM databases, you use DATASET statements to specify multiple data set groups. IMS assigns segments to the data set specified on the preceding DATASET statement.

For PHDAM and PHIDAM databases, you assign a segment to a data set by specifying the DSGROUP= parameter on the SEGM statement.

Recommendation summary for multiple data set groups

The recommendations for the use of multiple data set groups are:

- ▶ Define multiple data set groups only when you have a reason to do so. You should use them primarily to reduce I/Os for databases with certain characteristics.
- ▶ You can use multiple data set groups to provide additional capacity for HDAM and HIDAM databases, but conversion to PHDAM or PHIDAM is preferable.
- ▶ Do not use multiple data set groups with PHDAM or PHIDAM databases to provide capacity. Instead, create more partitions for these databases. Multiple data set groups might be appropriate for PHDAM or PHIDAM databases to reduce I/Os.

5.12 Compression

You can compress segments by using Segment Edit/Compression routines. This reduces the amount of space that is required on DASD for the segments. It does not affect the view of the segments by application programs. You can use compression with PHDAM, PHIDAM, HISAM, HDAM, and HIDAM databases. You cannot use compression with PSINDEX, SHISAM, or INDEX databases.

You specify compression with the *COMPRTN*= parameter on the SEGM statement. You can use different specifications for different segment types. You can compress some segment types in a database while not compressing others. You can use different compression routines for different segment types.

IMS supplies a sample compression routine, *DFSCMPX0*, which implements run length encoding. This reduces all strings of four or more repeating bytes to three bytes. It can give excellent results for segments which contain many blanks, zeroes, or other values which appear in consecutive bytes. IMS also supplies a facility for creating compression routines which use special system hardware for compressing data. This is the Hardware Data Compression facility. You give sample data to the facility and it produces a compression dictionary which is optimized for the sample data. You can also purchase products which supply compression routines and provide ease of use in maintaining and administering compression. The *IMS Hardware Data Compression Extended for z/OS* product (program number 5655-E02) in IBM DB2 and IMS Tools portfolio provides these capabilities.

Compression can significantly reduce the amount of space that is required for storing a database. This can substantially reduce the number of reads and writes that are required for processing the database. The reduction can be significant when the entire database, area, or partition is processed. On the other hand, compression can also substantially increase the CPU time that is required to process a database if you are not using Hardware Data

Compression. The use of compression is a trade-off between storage, I/Os, and CPU processing.

A more complete explanation of the use and implementation of compression with IMS is given in *A Guide to IMS Hardware Data Compression*, WP100416. This guide is available at: <http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100416>

5.12.1 Key compression as opposed to data compression

If you implement compression, you have the option of specifying whether you want key compression or only data compression. With key compression, the entire segment past the prefix is compressed. With data compression, only the data following the key is compressed. Key compression cannot be used with HISAM root segments. They must use data compression.

Obviously, key compression produces greater space savings. But there is a cost. When a database call needs to examine the key of a segment, IMS must invoke the exit routine when the key is compressed. The exit routine does not have to expand the entire segment, but it must expand the data through the key. This is not required when only the data is compressed. An example is a get call which is qualified on the key where the segment is in a long twin chain. IMS might have to look at many segments before it finds one that satisfies the call. With key compression, this would require the expansion of many segments. With data compression, only the segment which satisfies the call would be expanded.

The compression option you choose should depend on the amount of potential savings from key compression as opposed to the extra processing it requires. This depends on the size of the key, the location of the key in the segment, and the type of processing done against the segment.

5.12.2 COMPRTN= parameter

The *COMPRTN=* parameter on the SEGM statement is used to specify a segment edit/compression routine. The syntax of the *COMPRTN=* parameter is the following:

- ▶ For full function fixed length segments
`COMPRTN=(Routine name,DATA or KEY,INIT,size,PAD)`
- ▶ For full function variable length segments
`COMPRTN=(Routine name,DATA or KEY,INIT)`

There are five positional subparameters:

- ▶ Routine name

The first subparameter, routine name, identifies the name of the routine.

- ▶ DATA and KEY

The second subparameter indicates if all of the segment, including the key, is to be compressed or if only the data past the key is to be compressed. DATA is the default. It indicates that only the data past the key is to be compressed. KEY indicates that the key and all of the data is to be compressed. KEY cannot be specified for HISAM root segments.

- ▶ INIT

The third subparameter, INIT, is optional. It indicates that the routine is driven at initialization and termination. This allows the routine to do some special processing, such

as loading a table at initialization and deleting it at termination. Your exit routine requirements determine if this parameter must be specified.

► **Size and PAD**

The fourth and fifth subparameters are valid only for fixed length segments. The fourth subparameter has two possible meanings. It is either an “increment” size or a “PAD” size. The fifth subparameter determines the meaning of the fourth subparameter. If the fifth subparameter is omitted, the fourth subparameter is an increment size. If PAD is specified for the fifth subparameter, the fourth subparameter is a PAD size.

An increment size should not be specified. The following is an explanation of its meaning and why you should not specify it. An increment size is the number of bytes that the routine can add to a fixed length segment's size. This capability is provided because a compression routine algorithm could increase the size of a segment instead of decreasing it. Increment values of 1 to 32 767 can be specified. If an increment size less than 10 is specified or if no value is specified, an increment size of 10 is used. When PAD is specified, an increment size of 10 is used. The increment size is not required for any compression routines supplied by IMS or IBM products, such as *IMS Hardware Data Compression Extended*. They never add more than one byte to a segment. Specification of increment sizes is rarely required for any routines, because most compression routines never increase a segment's size by more than 10 bytes.

A PAD size is used to specify a minimum size that the segment will be when written to DASD. When compression reduces the segment's length to less than the PAD size, IMS pads the segment to the PAD size before writing it to DASD. PAD sizes are never required, but can be recommended for performance reasons.

The use of PAD is explained more thoroughly in *A Guide to IMS Hardware Data Compression*, WP100416. It is available at:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100416>

5.12.3 Recommendation summary for compression

The recommendations for the use of compression are:

- Evaluate the use of compression. The space saved might not be worth the extra processing required to compress and expand segments.
- Evaluate the use of key compression as opposed to data compression. Key compression might not be advisable for segments which have long twin chains and which are frequently processed by calls qualified on key fields.
- It might be wise to specify a PAD value for full function fixed length segments which are compressed. This can reduce the likelihood that a segment's prefix and its data are placed in separate blocks.
- Never specify the fourth subparameter (size) for COMPRTN= if you do not also specify the fifth subparameter (PAD).

5.13 Encryption

Because encryption uses the Segment Edit/Compression exit routine on the SEGM statement, all of the considerations for compression apply to encryption.

When you encrypt segments by using the Segment Edit/Compression exit, it does not affect the view of the segments by application programs. In other words, as long as you are using IMS to view the data, it is decrypted. You can use encryption with (P)HDAM, (P)HIDAM, and

HISAM databases. You cannot use encryption with PSINDEX, SHISAM, or INDEX databases.

You specify encryption with the COMPRTN= parameter on the SEGM statement. You can use different specifications for different segment types. You could encrypt some segment types in a database while not encrypting others. You do not have to use the same encryption routine for every segment type.

Encryption can substantially increase the CPU time that is required to process a database. When segments are encrypted, IMS must invoke an exit routine each time a segment is inserted, replaced, or read. The use of encryption is a trade-off between data security and CPU processing.

A more complete exposition of the use of encryption with IMS for the product *IBM Data Encryption Tool for IMS and DB2 Databases* (program number 5799-GWD), can be found on the DB2 and IMS Tools Web site:

<http://www.ibm.com/software/data/db2imstools/db2tools/ibmencrypt.html>

For more information, refer to the IBM Redbook *IBM eServer zSeries 990 (z990) Cryptography Implementation*, SG24-7070.

5.14 Secondary indexes

You can define secondary indexes for PHDAM, PHIDAM, HDAM, HIDAM, and HISAM databases. They are used primarily to provide an alternate key for retrieving records in a database. You can also use them to provide an alternate sequence for sequential processing.

You define secondary indexes for PHDAM and PHIDAM databases with ACCESS=PSINDEX on the DBD statement. Define secondary indexes for HDAM, HIDAM, and HISAM databases with ACCESS=INDEX on the DBD statement.

The indexed databases must include LCHILD and XDFLD statements for their secondary indexes.

5.14.1 Secondary index keys

The key for a secondary index is built by using one to five fields in the index source segment. These fields can be noncontiguous. They are specified with the SRCH= parameter on the XDFLD statement in the indexed database.

Unique keys as opposed to non-unique keys

Secondary indexes for HALDB (PHDAM and PHIDAM) databases must have unique keys. Unique keys are not required for non-HALDB databases, but they are highly recommended. You define unique keys by specifying U in the third subparameter of the NAME parameter in the FIELD statement for the index key field. You define non-unique keys by specifying M. U is the default.

You can create unique keys by defining subsequence fields. *Subsequence fields* are added to the sequence fields so that a unique key can be created. These keys are the keys of the secondary index KSDS. Application programs do not use subsequence fields when they define a search using a secondary index. They use only the secondary index sequence fields. Subsequence fields are defined with the SUBSEQ= parameter on the XDFLD statement in DBDGEN.

For example, if you have an employee database, you might want to create a secondary index based on employee last name. Of course, there might be duplicate last names among the employees. Adding a subsequence field, such as employee serial number, would create unique keys. In some cases, there might not be a field in the source segment which provides uniqueness. IMS supplies system-related fields, which you can use to create uniqueness. The system-related fields are the concatenated key field and the "/SX" field.

A *concatenated key field* contains the concatenated key of the target segment. The size of this field depends on the size of the concatenated key. A "/SX" field is a unique value generated by IMS. For non-HALDB databases, it is the 4-byte relative byte address (RBA) of the target segment. For HALDB databases, it is the 8-byte Indirect List Key (ILK) of the target segment. In most cases, the "/SX" field requires less space than the concatenated key field. System-related fields are defined with special names in the NAME= parameter of the FIELD statement. A concatenated key field is defined with a field name beginning with /CK. A "/SX" field is defined with a field name beginning with /SX.

Non-unique keys have two disadvantages. First, they require a second data set for the secondary index, the overflow data set. The second data set is an ESDS which contains the duplicate keys. Second, each index entry requires an extra four bytes for the pointers which chain possible duplicate entries. Since a /SX field requires only four bytes in non-HALDB databases, it requires no more space than the duplicates pointer.

Application programs never see subsequence fields unless they process the secondary index as a database.

5.14.2 Direct as opposed to symbolic pointers

Secondary indexes use three types of pointers. Non-HALDB secondary indexes use either direct or symbolic pointers. HALDB secondary indexes use an extended pointer set (EPS).

You can use direct pointers with HDAM and HIDAM databases. They are four byte RBAs. They point at the target segment. You can use symbolic pointers with HDAM, HIDAM, and HISAM databases. Symbolic pointers are the concatenated key of the target segment. They can be up to 255 bytes, depending on the size of the concatenated key. When using a secondary index with symbolic pointers, IMS must traverse the target database record from the root to the target segment. This might require accessing multiple OSAM blocks or VSAM CIs. For this reason, direct pointers are more efficient when accessing target segments. This is especially true when the target is several levels below the root in the database structure. For indexes into HDAM databases, symbolic pointers cause IMS also to traverse from the RAP to the root. This makes direct pointers even more attractive.

Symbolic pointers have a significant advantage over direct pointers when reorganizing the indexed database. If you use direct pointers for a secondary index, you must rebuild it when you reorganize the indexed database. This is required because the old direct pointers are no longer valid, they point to the wrong places. If you use symbolic pointers, the secondary index remains good when you reorganize the indexed database because the pointers are the concatenated key of the target segment. Therefore, you do not have to rebuild the secondary index.

The use of direct pointers as opposed to symbolic pointers for HDAM and HIDAM databases is a trade-off. Direct pointers have two advantages. They are smaller and they provide more efficient access to target segments. Symbolic pointers have the advantage of not requiring rebuilds after reorganizations of the indexed database.

HALDB secondary indexes always use an extended pointer set (EPS). They do not use the direct or symbolic pointers which non-HALDB secondary indexes use. The EPS contains an

internal direct pointer to the target. The self-healing pointer process updates this pointer after reorganizations of the indexed database. You do not have to rebuild a HALDB secondary index when you reorganize the indexed database. The EPS tends to give HALDB secondary indexes the advantages of both symbolic pointers and direct pointers. Access is direct, but you do not have to rebuild the secondary index when you reorganize the indexed database.

You select symbolic pointers by specifying PTR=SYMB on the LCHILD statement in the indexed database. You select direct pointers by specifying PTR=INDX on the LCHILD statement. You must specify PTR=INDX on the LCHILD statement for HALDB.

5.14.3 Shared secondary indexes

Multiple secondary indexes for non-HALDB databases can reside in the same data set. This is called *shared secondary indexes*. You should *never* use shared secondary indexes. When multiple indexes share the same data set, a rebuild of any of the indexes requires a rebuild of all of them. Shared secondary indexes are used when a CONST= parameter is defined on the XDFLD statement in a DBDGEN.

5.14.4 Duplicate data

You can include duplicate data fields in secondary index entries. These are fields in the secondary index which contain data from the source segment in the indexed database. Duplicate data fields are defined with the DDATA= parameter on the XDFLD statement in DBDGEN. Duplicate data fields are not part of the key of the secondary index. Duplicate data is only available to application programs which process the secondary index as a database, not as an index. That is, the DBDNAME= parameter in the PCB of the PSB references the secondary index, not the indexed database. Duplicate data fields make the secondary index larger. You should define them only when applications are designed to process the secondary indexes as databases. However, applications which take advantage of duplicate data can be more efficient. This occurs because they do not have to access the indexed database to have access to the data in the duplicate data fields. An application program never sees duplicate data unless it is processing the secondary index as a database.

5.14.5 User data

You can include user data fields in secondary index entries. These fields are not explicitly defined in the DBD. They are created when the size of the secondary index entry (BYTES= parameter on the SEGM statement) is larger than that required to hold the explicitly defined fields. IMS does not maintain user data fields. User programs, which process the secondary index as a database, maintain these fields. User data fields are rarely used. If you rebuild the secondary index, IMS does not rebuild the user data fields. The data is lost. For this reason, non-HALDB secondary indexes, which use direct pointers, should never have user data fields. HALDB secondary indexes and non-HALDB secondary indexes, which use symbolic pointers, do not need to be rebuilt after reorganizations; therefore, they might be able to use user data fields. An application program never sees user data fields unless it is processing the secondary index as a database.

5.14.6 Sparse indexing

Sparse indexing allows you to build secondary indexes, which do not have entries for every source segment in the indexed database. This can be useful with some applications. They might want a secondary index with entries for only certain segments. A sparse index can simplify the application and reduce the size of the secondary index.

You can create a sparse index with either of two techniques. First, you can specify the NULLVAL= parameter on the XDFLD statement in the DBDGEN. If the indexed field or fields contain this value in every byte, IMS does not create an index entry for this source segment. Second, you can specify the EXTRTN= parameter on the XDFLD statement. IMS calls the specified exit routine for every insert, delete, or replace of the source segment. The exit routine can inspect the source segment and decide whether an index entry should be created for it. You can combine both techniques. If you specify both the NULLVAL= parameter and the EXTRTN= parameter, IMS creates the secondary index entry only if neither technique suppresses the entry.

For more information about coding sparse indexes, look at the chapter, “Secondary Index Database Maintenance Exit Routine”, in the *IMS Version 9: Customization Guide*, SC18-7817. There are two assembler examples of sparse indexing in the appendix: “Source segment” on page 238 and “Index segment” on page 241.

5.14.7 Recommendation summary for secondary indexes

The recommendations for secondary indexes are:

- ▶ Use unique keys for secondary indexes. You may use /SX fields to create uniqueness.
- ▶ Never use shared secondary indexes.
- ▶ Specify duplicate data fields only when applications are designed to use them.
- ▶ Define space for user data fields only when applications are designed to use them and when rebuilds of the secondary index are not required.

5.15 Fast Path performance considerations

This section covers topics related to Fast Path processing.

IMS Fast Path includes two database organizations:

- ▶ Data Entry Database (DEDB)
- ▶ Main Storage Database (MSDB)

DEDBs are similar to HDAM but have some significant differences that provide even higher performance, capacity, and availability. MSDBs cannot be used in a data sharing environment, and we do not discuss them here. We recommend DEDB Virtual Storage Option instead of MSDB.

5.15.1 Virtual Storage Option (VSO)

VSO provides the user with the best features of data entry databases and the access time and parallelism of main storage databases. VSO data resides in a z/OS data space in virtual storage, so response times for DL/I calls are close to memory access times.

5.15.2 Field (FLD) calls support

The FLD call can be used to access a field within a segment in a DEDB. The more relaxed locking requirements of the FLD call (as opposed to GHU and REPL calls) provide a higher degree of parallelism. More transactions can update the data concurrently without incurring contention, and thus, this call is ideally suited to data that is heavily updated.

5.15.3 Shared Virtual Storage Option (SVSO)

IMS uses the Coupling Facility to allocate shared VSO data instead of z/OS data spaces. This makes VSO DEDBs accessible for all the IMS systems in a data sharing group.

5.15.4 DEDB general performance considerations

In this and following sections, we cover general performance considerations of DEDBs, sequential processing on DEDBs, sequential dependents sharing, and performance tips for VSO and SVSO implementation as well.

A DEDB is a hierarchical database designed to provide efficient storage and access to large volumes of data with a high level of availability.

As is the case with DL/I databases, some performance problems do occur with DEDBs and can be solved by following the recommendations we give below.

5.15.5 DASD or channel contention for I/O on DEDB

One possible solution to the problem of DASD contention with DEDBs is to alter the use of area data sets. It might be possible to identify the parts of the database that are heavily accessed (called *hot spots*) and move these records either to another existing area or to a new area.

We recommend that you spread the hot spots around, if possible, so that the activity is evenly dispersed between the area data sets on fast DASD devices.

5.15.6 OTHREAD contention

When buffers that are waiting to be written start queuing for OTHREADs, buffer contention increases because the locks on the data in the buffers are not released until the buffers are written to DASD. If the problem is on the buffer side, we recommend that you split the area across several DASD volumes.

However, the problem could be an output thread shortage. We also recommend that you set the OTHR system execution parameter to a value big enough so that the write buffers are not queued because of insufficient numbers of SRBs.

The OTHREAD Analysis section of the IMS Performance Analyzer Fast Path Analysis report gives useful information about the value to specify for the OTHR parameter. If the Max Value column of the Active OTHREADs section is close or equal to the OTHR parameter, we recommend that you increase this parameter value.

The output thread processing for VSO and SVSO DEDBs is somewhat different. Refer to “VSO output threads” on page 94 for more details.

5.15.7 Increased I/O or CI contention for independent or dependent overflow

If your system experiences I/O or CI contention for the Independent Overflow (IOVF) or the Dependent Overflow (DOVF) parts, we recommend that you reorganize the database. We recommend that you monitor the usage of IOVF and DOVF periodically to ensure that the performance of the databases is not impacted by I/O or CI contention.

5.15.8 Overflow Buffer Allocation (OBA) latch wait

If your system experiences waits for the overflow buffer allocation latch, we recommend that you increase the value of the Normal Buffer Allocation parameter for the dependent regions. This reduces the OBA usage and consequently the latch conflicts for it.

When doing so, you should also take care when increasing the value of the DBBF system execution parameter. If you are not careful, ABENDs can occur when starting dependent regions because of a shortage of Fast Path buffers. Refer to 5.15.17, “IMS Fast Path buffers” on page 101 for recommendations for this parameter.

The Latch Conflict Statistics of IMS Performance Analyzer Internal Resource Usage report show the number of IWAITs for every latch. The OBA latch IWAIT numbers should always be zero or close to zero.

5.15.9 DEDB sequential processing

If you have applications that use a PCB to sequentially process (read-only scans or updates) one or more areas of a DEDB, we recommend that you consider using High Speed Sequential Processing.

This is requested by using PROCOPT=Hx. There are no special hardware requirements to implement HSSP.

Sequential processing means physical sequential processing, typically driven by unqualified GN calls on the root segments. However, with care, it can also be the result of qualified GN or GU root calls. The essential aspect of the processing is that many or all CIs in the base section of each unit of work are accessed.

When using HSSP, IMS allocates private buffers to the application and uses both chained reads and look-ahead buffering. The update writes are done by standard OTHREADs and hence are also asynchronous and use chained write I/Os.

A standard option with HSSP is to take an asynchronous image copy while the HSSP application is running.

HSSP provides superb sequential performance by exploiting significant amounts of page fixed storage for the private buffers.

5.15.10 I/O error toleration support for DEDB

When one or a few CIs suffer I/O errors, the data in the CI in error remains available to all applications in the system which experienced the write error. Any attempt to access the CI from another data sharing IMS results in an “AO” status code. The area can be recovered without any requirement for a data outage.

5.15.11 DEDB using the Virtual Storage Option

Virtual Storage Option (VSO) is a high-performance option for DEDBs and the preferred alternative to MSDBs for holding data in memory. By means of VSO, IMS provides the user with the best features of DEDBs and the response time and parallelism of MSDBs.

The definition, access, and operations on a DEDB using VSO function are exactly the same as they are for an ordinary DEDB, but the data is kept in virtual storage using a z/OS data space. Thus, programs do not wait for I/Os when segments are accessed.

Local VSO DEDB performance tips

Here we provide performance tips related to segment level locking, lock contention, I/O reduction, VSO output threads, and IMS system checkpoint.

Segment level locking

Locking at segment level is implemented for any VSO DEDB whose characteristics exactly match those of an MSDB, that is:

- ▶ Root-only hierarchy
- ▶ Fixed length segment
- ▶ PCB PROCOPT=G or R
- ▶ VSO option in DBRC
- ▶ No compression

If implementing the VSO option for an existing DEDB, then we recommend that you consider modifying the DEDB, whenever possible, so that it matches the segment level locking requirements. However, it might not be possible to take an existing DEDB and modify it so that it meets the requirements of segment level locking.

Lock contention

One of the benefits obtained from a VSO DEDB when compared to a non-VSO DEDB is that a CI lock held for updating can be released earlier because of the differences in the output thread processing:

- ▶ With a non-VSO DEDB, a CI lock has to be held until the CI is written out to DASD. Because Fast Path writes out the CIs asynchronously for performance reasons, this means that the CI lock can be held for a relatively long time.
- ▶ With VSO areas, the lock is released as soon as the updated CI has been copied to the data space, which occurs early in sync point phase 2. This considerably reduces lock contention.

Table 5-3 summarizes the lock level in effect for every case.

Table 5-3 Locking level for GU and GN DL/I calls on VSO DEDB

PROCOPT	Segment level locking	VIEW=MSDB	Lock level
A	N	N	EXCLUSIVE
A	N	Y	READ
A	Y	N	EXCLUSIVE
A	Y	Y	EXCLUSIVE
G	N	N	READ
G	N	Y	READ
G	Y	N	READ
G	Y	Y	READ
GR	N	N	EXCLUSIVE
GR	N	Y	READ
GR	Y	N	EXCLUSIVE
GR	Y	Y	READ

If you migrate an MSDB to VSO DEDB, we recommend that you include VIEW=MSDB in the corresponding PCBs, in order to avoid possible deadlock situations and performance impact.

If you implement the VSO option for an existing non-VSO DEDB, keep in mind that if you code VIEW=MSDB in the PCB the READ locks are released soon after the GU calls complete. This behavior is quite different from the non-VSO DEDB method, and your application programs might not be prepared for this event or they might experience an integrity exposure.

With VIEW=MSDB PCBs holding locks on VSO DEDBs as MSDBs do, and locking at the segment level, there is not a significant difference between VSO DEDB locking rules and the ones used with MSDBs. This means that, from a lock contention and parallelism point of view, VSO DEDBs are comparable to MSDBs and considerably better than non-VSO DEDBs.

The FLD call works for a VSO in just the same way as for an MSDB. That is to say, FLD calls are processed at sync point and are sorted by resource.

I/O reduction

Some IMS systems have DEDB areas that experience very high I/O rates. Implementing such areas with VSO provides major benefits, because the read I/Os are eliminated and the write I/Os are optimized; updates to a CI from multiple transactions are applied with a single I/O. We strongly recommend that you implement VSO for small and highly volatile DEDBs. If your system has small and highly volatile databases implemented as MSDBs, we recommend that you migrate them to VSO DEDBs as well.

VSO output threads

Periodically, all updated CIs are written out from the data spaces to the area data sets during a process that is also called *Output Thread*.

This OTHREAD process for VSO DEDB is more efficient than the non-VSO DEDB process. Locks on CIs are released earlier during sync point processing, and the I/Os are performed in chains of 200 KB. Also, multiple updates to the same CI are only written back to DASD once. See Table 5-4 for a comparison of OTHREAD processing.

At IMS system checkpoint, the OTHREAD process is started too, but it is also asynchronous, and so it does not impact IMS system checkpoint duration. If you migrate your MSDB to VSO DEDB and specify your system to no longer use MSDBs (MSDB parameter left to null in DFSPBxx member of IMS.PROCLIB), then your IMS system checkpoint elapsed time should be reduced.

Table 5-4 OTHREAD processing for VSO and non-VSO DEDBs

Non-VSO DEDB	VSO DEDB
Updated CI in FP buffer	Updated CI in FP buffer
SYNC	SYNC
	CI Write to Data Space
	Lock Release
Asynchronous Physical Logging → OTHREAD	Asynchronous Physical Logging
Database Data set Write I/Os	Asynchronous OTHREAD (Wait for Logging)
Lock Release	Database Data set Write I/Os

VSO PRELOAD options

Any DEDB can be moved to virtual storage by specifying the VSO keyword on the DBRC commands INIT.DBDS or CHANGE.DBDS.

VSO areas are mapped linearly to z/OS data spaces. During IMS startup, two 2 GB data spaces are acquired:

- ▶ One has the disabled reference (DREF data space) option specified. That means that its pages can reside in memory but are never paged out to DASD.
- ▶ The other data space does not have the DREF option (non-DREF data space) and might experience paging.

An area can be defined to DBRC with either the PRELOAD or the NOPREL attributes:

- ▶ Any area defined with the PRELOAD option is read into the DREF data space following the IMS initialization system checkpoint. Only the direct part of the area is loaded.
- ▶ If an area has the NOPREL option, then each direct part CI is allocated a position in the non-DREF but no automatic load takes place. Instead, the first time a CI is requested after the area open, IMS reads it from DASD and copies it to the data space.

SDEP CIs are never loaded into the data space.

CIs from PRELOAD areas always remain in central storage. For medium or small areas with a high access rate where most of the CIs in the area are accessed with similar frequency (there are zero or negligible hot spots), we recommend that you implement the PRELOAD option.

When choosing the PRELOAD option, keep in mind that PREOPEN is implicitly assumed for this option. This means that VSAM area data sets are allocated and opened immediately when IMS starts or a **/START AREA** command is issued.

CIs from NONPRELOAD areas can be paged out to page data sets if there is a storage constraint. If you have big VSO areas where only a small part of the area is actually accessed, we recommend that you implement the NOPREL option so that only space for needed CIs is allocated.

VSO DEDB performance will more likely degrade with paging than non-VSO DEDBs doing I/Os to the actual data sets. We recommend that you make conscious use of the VSO option for DEDBs, that is, for those databases where the area is not too big and which need high performance.

IMS system checkpoint

When VSO DEDBs databases are used, the content of the data space is written to DASD at IMS system checkpoint time only for those VSO DEDBs that have had update processing. To do so, an output thread is started at every system checkpoint. But this output thread is an asynchronous process, and its elapsed time does not impact the IMS system checkpoint duration.

A considerable reduction in the IMS system checkpoint elapsed time (tens of seconds, depending on how large the MSDBs are) can be achieved by migrating MSDBs to VSO DEDBs and specifying the MSDB parameter in the DFSPBxx member of IMS.PROCLIB as null.

During IMS system checkpoint, transaction processing almost stops, which has several negative consequences. If this is the case for your system, we recommend that you migrate from MSDBs to VSO DEDBs.

Monitoring VSO performance

The VSO Activity Summary: SHARELVL 2/3 section of IMS Performance Analyzer VSO Statistics report provides information that can be used to monitor the performance of the VSO areas. Refer to Example 5-5 for a sample of this report. You can determine how well VSO is performing by comparing the number of requests that are performed against the data space to the number of requests that need DASD access.

Example 5-5 IMS Performance Analyzer VSO Statistics report (VSO Activity Summary: SHARELVL 2/3)

IMS Performance Analyzer VSO Activity Summary: SHARELVL 2/3 - IM1B											
From 25Sep2006 13.15.48.33			To 25Sep2006 13.32.33.58			Elapsed= 0 Hrs 16 Mins 45.251.273 Secs					
Database Name	Area Name	--- IMS from/to CF ---		-----VSO CF from/to DASD-----			-----	Lookaside-Pool Buffer			
		Gets	Puts	Gets	Puts	Castouts	Searches	Hits	Pct	Hit Valid	Pct
DISTDB	AREADI01	4919	24712	0	395	3	48696	8053	16.5	8053	16.5
ITEMDB	AREAIT01	30145	1002	501	0	3	236444	32802	13.9	32573	13.8
WAREDB	AREAWH01	0	12100	0	40	3	45769	12418	27.1	12418	27.1
System Totals		35064	37814	501	435	9	330909	53273	16.1	53044	16.0

You can also monitor the areas that are actually loaded into data spaces and the amount of storage they use. The *AREASIZE* column shows the number of 4 K pages reserved for a certain area in the data space. This amount is the maximum space needed to allocate the whole area (all its CIs) and depends on the DBD. If the area is not PRELOADED, the actual amount of storage allocated for the area can be sensibly less.

Example 5-6 shows a **/DIS FPVIRTUAL** command.

Example 5-6 /DIS FPVIRTUAL command

```
IM1BDIS FPVIRTUAL
DFS4444I DISPLAY FROM ID=IM1B
      DATASPACE MAXSIZE(4K) AREANAME AREASIZE(4K) OPTION
      001      524238      DREF
      NO AREAS LOADED INTO DREF DATASPACE 001.
+ AREANAME STRUCTURE      ENTRIES CHANGED AREA CI# POOLNAME OPTIO
NS
+ AREAWH01 IMOB_AREAWH01A 0000075 0000020 00000075 WAREPOOL PREO,
PREL
+ AREAWH01 IMOB_AREAWH01B 0000075 0000020 00000075 WAREPOOL PREO,
PREL
+ AREAIT01 IMOB_AREAIT01A 0002448 0000000 00002520 ITEMPPOOL PREO,
PREL
+ AREAIT01 IMOB_AREAIT01B 0002448 0000000 00002520 ITEMPPOOL PREO,
PREL
+ AREADI01 IMOB_AREADI01A 0000440 0000185 00000440 DISTPOOL PREO,
PREL
+ AREADI01 IMOB_AREADI01B 0000440 0000185 00000440 DISTPOOL PREO,
PREL
      *2006271/171202*
```

5.15.12 Shared Virtual Storage Option

Block-level data sharing of VSO DEDB areas allows multiple IMS subsystems to concurrently read and update VSO DEDB data. VSO DEDBs supporting block-level data sharing are commonly called *Shared VSO (SVSO) DEDBs*.

The main elements that participate in the SVSO implementation are:

- ▶ Cache structures in the Coupling Facility

Store-in cache structures are used to contain copies of the control intervals accessible to all the IMSs in the data sharing group.
- ▶ Local cache buffers

Each IMS has a set of buffers that contains a local copy of the control intervals.
- ▶ Permanent storage

DEDB area data sets are needed to contain a non-volatile copy of the data in DASD.

5.15.13 Local buffer pool definitions

The private local buffer pools can be defined by means of the DEDB statement of member DFSVSMxx on IMS.PROCLIB. Multiple DEDB statements can be used in order to define various buffer pools, each one with different attributes. The attributes that can be specified in the DEDB statement include:

- ▶ The name of the buffer pool
- ▶ The number of buffers for the primary and secondary allocations
- ▶ The maximum number of buffers that can be allocated
- ▶ The buffer pool buffer size
- ▶ Whether to assign the buffer pool to a certain area
- ▶ The LKASID option

An area can use a certain buffer pool if the CI size for the area matches the buffer pool buffer size, it has the same LKASID option specified in the RECON, and the buffer pool is not implicitly assigned to another area.

Two or more areas with the same characteristics can also share a buffer pool.

When starting an area, if no buffer pool is defined or there is not an available buffer pool that matches the CI size for the area, IMS creates a default buffer pool for it. Default buffer pools can also be shared among different areas.

We recommend that you keep the following suggestions in mind when defining local buffer pools:

- ▶ Do not allow the default buffer pools to be created. Include at least one DEDB statement for each different CI size in the DFSVSMxx member of IMS.PROCLIB.
- ▶ Specify a number of buffers large enough to hold the ones needed for the maximum number of concurrent requests. Specify them as primary allocation buffers, so that they are page fixed. Then, specify a secondary allocation to cover unexpected high load situations, and a maximum number of buffers large enough so that it is never reached (if a buffer is not available, IMS waits for a buffer to become free, which is undesirable). Paging of primary or secondary buffers does not occur. When secondary buffers are allocated, they are page fixed.

If your applications use PCB with PROCOPT=GO to access SVSO areas, Fast Path might steal local buffers. That means that extra buffers need to be defined for the local buffer pool.

Also, if the buffer pool is not large enough, the LKASID option can be ineffective. So, it is important for the primary allocation to be large enough.

A buffer containing committed changes that has not been written to the Coupling Facility (CF) cannot be reused until an OUTPUT THREAD writes the buffer to the CF and completes. Therefore, a lack of OTHREADS can result in more buffers being needed for the local buffer pools. We recommend that you set the OTHR parameter to the maximum value (255) if there is any doubt about how to fine-tune their number, because they are cheap to define.

- ▶ If the application's access rate pattern for different SVSO areas is similar, you could allow two or more areas to share the same buffer pool as long as they have the same CI size and LKASID options. It is better from an administration point of view. However, if the two areas both have very high access rates, then you should give the high-access-rate areas dedicated buffer pools and only allow the low-access-rate areas to share pools.

If one of the areas sharing a pool is accessed much more than the others, then it consumes most of the buffers, and the performance of the accesses to the other areas can be degraded. If the access profiles are not similar for the different areas in your system, we recommend that you assign dedicated buffer pools for each area.

- ▶ Normally, we always recommend the LKASID option unless the area is highly updated from different IMSs in the data sharing group.
- ▶ For those areas where the read/write ratio is very high, we recommend that you assign them a dedicated buffer pool and enable the LKASID option, so that the major part of the reads are resolved within the local IMS (without accessing the CF).
- ▶ We also recommend that you monitor the usage of these buffer pools and adjust the allocation values and the LKASID option appropriately.

LKASID option

IMS maintains the buffers in the local buffer pool chained off three or four queues:

- ▶ Available queue

These are the buffers available for use. It does not matter what they contain, because they are considered as empty buffers.

- ▶ Requestor queue

These are the buffers in use by an application program. These buffers count toward the NBA/OBA limit.

- ▶ Output Thread queue

These are the buffers with committed updates that are waiting for an OTHREAD to be written to the CF.

- ▶ LKASID queue (optional)

This queue only applies for pools with the LKASID option. This is discussed in this section.

When the LKASID option is the choice, the local buffers used by the application are not returned to the available queue after application sync point. Instead, they are chained off the lookaside queue.

We recommend that you select the LKASID option. It provides better performance, because CF accesses are saved, especially if the read/write ratio is high and most of the requests for CI are resolved locally without accessing the CF.

In order to take advantage of the LKASID option, the buffer pool has to be large enough (primary plus secondary) to hold all of the current requests for buffers from executing transactions (requestor queue), for scheduled but incomplete output threads (output thread queue), and with enough additional buffers to provide a sufficient number of buffers that remain on the lookaside queue.

How efficiently the LKASID option performs depends on the ratio `valid_hits/number_of_searches`. If this ratio is not a big number, it is not worth paying the cost of searching the lookaside queue, and you might consider choosing NOLKASID.

This ratio can be obtained by means of either IMS Performance Analyzer VSO Statistics report, the `/DIS POOL FPDB` command, or the DBFULTA0 utility.

There are three reasons for the LKASID option to be inefficient:

- ▶ A shortage in the buffer pool
- ▶ The application database access profile
- ▶ INOPREL area and CF structure size shortage

Be aware that the Valid ratio shown in the output for the `/DIS POOL FPDB` command has a slightly different meaning than the Hits Valid ratio shown in the IMS Performance Analyzer VSO Statistics report. For the command, Valid means the percentage of times a buffer found in the pool had valid data. Therefore, you must use the displayed for Hits and Valid together to obtain the ratio of efficiency for the LKASID option. For example, if Hits is 40%, and Valid is 75%, a buffer was found in the pool 40% of the time, and of that 40%, 75% of the buffers found had valid data, that is, 30% of the requests found buffers on the LKASID queue with valid data. So, IMS had to read data from CF approximately 70% of the time. The Hits Valid ratio shown in IMS Performance Analyzer VSO Statistics report is 30%.

DBFULTA0 gives the number of cross-invalidated hit buffers instead, that is, the percentage of invalid buffers from those that were found in the lookaside queue.

Monitoring the local buffer pools is necessary to set the local buffer pool allocations and the LKASID option appropriately.

Coupling Facility structure definitions

Each VSO DEDB cache structure in the shared storage of a Coupling Facility represents one or more VSO DEDB areas. A VSO DEDB cache structure can be a single-area structure or a multi-area structure. A single-area structure can contain data from only one DEDB area. A multi-area cache structure can hold data from multiple areas. Both single-area and multi-area cache structures conform to the characteristics of the areas for which they are created. Both types of cache structures are also non-persistent: they are deleted after you close the last area connected to them.

To determine the structure size, you can use the z/OS Coupling Facility structure sizer tool (CFSizer). CFSizer is a Web-based application that calculates the structure size based on the input data that you provide. To use the CFSizer tool, go to:

<http://www.ibm.com/servers/eserver/zseries/cfsizer/>

5.15.14 PRELOAD | NOPREL option

The *PRELOAD* option causes the whole area to be read from DASD and written to the CF as soon as IMS restarts or an `/STA AREA` command is issued. When this is the choice, the CF structure size has to be large enough to contain all the CIs of the direct portion of the area; otherwise, the area cannot be started.

As a general rule, we recommend that you use NOPREL and specify a CF structure size large enough to contain the CIs of the area that are actually active. If there are hot spots in the database, as soon as the applications start accessing those CIs, they are placed in the CF and remain there for the rest of the IMS session (or until the area is closed or a /VUNLOAD command is issued). No unnecessary space is wasted in the CF with NOPREL and an appropriate CF structure size. The first reference to each CI might have a worse response time because DASD accesses are also needed (CF structures are just a cache).

PRELOAD is only recommended if almost all the CIs in the area contain data, they are all referenced with similar frequency, and it is required to have very good performance right from the start.

Also, you might consider using PRELOAD for those areas with medium load but where extremely high performance is a requirement and all the CIs are randomly accessed. In this case, PRELOAD maintains all the CIs in the CF.

5.15.15 Block level locking and root-only DEDBs

Local VSO DEDB areas use segment level locking, which makes them equivalent to MSDBs. This means that two segments could be accessed simultaneously from different dependent regions without incurring lock contention even if they were in the same CI and the access intent was exclusive.

With SVSO areas, the locking unit is the CI. If, after moving from local VSO to SVSO, you experience an increase in lock contention, we recommend that you try to reduce contention problems at a maximum by means of:

- ▶ Reducing the CI size

Different CI sizes can be specified for each area and local buffer pool. We recommend that you try to adjust the CI size to the segment size wherever it is possible, so that only one segment fits in a CI.

- ▶ Expanding the area

You can also increase the size of the area in order to have more available RAP CIs. This causes segments to be more sparsely distributed in the database and reduces the chances of having two or more segments in the same CI. If you do so, you should also be aware that, if PRELOAD option is the choice, the CF structure size needs to be appropriately increased. Keep in mind that increasing the area size can result in lots of empty CIs; you might consider if PRELOAD is the best option in this case.

We also recommend that you adjust the PROCOPT values of the database PCBs to the actual type of access that the programs perform. If you overly specify the PROCOPTs, you can experience lock contention problems. The only time locking is different between VSO and SVSO is if VSO is eligible for segment level locking. If you have lock contention with VSO, you will have the same contention with SVSO. The difference is that with SVSO, you use the IRLM to resolve the contention. If there is contention, then using the IRLM to resolve it is not appreciably less efficient than using PI to resolve it if data sharing is not used at all.

If you implement the VSO option for an already existing SHARELEVEL(3) DEDB, no particular increase in lock contention is expected to occur.

5.15.16 Sequential dependent sharing (shared SDEPs)

The sequential dependent (SDEP) segment function provides the user with a time-sequenced insert capability for a portion of a DEDB area that has SDEP defined.

SDEP segments are physically stored in chronological sequence in SDEP CIs into the last part of the DEDB area data set. Therefore, they are a unique segment type to the DEDB. They are also chained off the root as a second dependent segment.

SDEP segments cannot be replaced or deleted by applications, only inserted (and also retrieved but with poor performance), so utilities are provided to sequentially read, process, and delete SDEP segments in an area (SCAN and DELETE utilities). This kind of segment is designed for quick insert and is ideally suited for journal type applications.

SDEP buffers are kept in main storage and written out to DASD by an output thread once the buffers are filled to capacity.

5.15.17 IMS Fast Path buffers

The following IMS control region and dependent region EXEC parameters related to Fast Path buffers are important for performance tuning:

- ▶ Control region EXEC parameters (DFSPBxx member of IMS.PROCLIB)
 - DBBF: the total number of buffers
 - DBFX: the system buffer allocation
- ▶ Dependent region EXEC parameters
 - NBA: normal buffer allocation
 - OBA: overflow buffer allocation

We recommend that you use the following formula in order to calculate the number of Fast Path database buffers required:

DBBF = Number of open areas that have SDEP segments
+ Sum of NBA for all concurrently active FP programs
+ Largest OBA allocation for any of the concurrently active FP programs,
including any specified by CICS for DBCTL
+ DBFX
+ Sum of all Fast Path buffers used by CICS(CNBA)

If the number of database buffers requested by DBBF is not large enough, then an area open or a region initialization fails.

5.15.18 Normal buffer allocation

When a dependent region is started with NBA specified in its execution parameters, it causes the NBA number of buffers to be made available for the region in the Fast Path buffer pool. This number of buffers needs to be sufficient to handle the processing of the vast majority of programs running in that region. These buffers are page fixed when the region starts.

All CIs locked at the exclusive level remain locked until the buffer is released. Buffers that have not been updated are released when either:

- ▶ The NBA limit is reached (and buffer stealing occurs).
- ▶ The program reaches sync point.

Updated buffers are released only when the OTHREADS have completed.

Example 5-7 IMS Performance Analyzer Fast Path Resource Usage and Contention Report

IMS Performance Analyzer

Fast Path Resource Usage and Contention - IM1B

From 25Sep2006 13.20.02.24 To 26Sep2006 17.04.58.94 Elapsed= 27 Hrs 44 Mins 56.699.936 Secs

Transact Code	Routing Code	Count	---DEDB Calls---				---ADS I/O---				--VSO Activity--				-Common Buffer- Usage				Contentions				LGNR Total	Stat #CI	Totl Sync	Tran Rate
			Reads	Updates	Reads	Updates	Reads	Updates	Reads	Updates	Reads	Updates	Reads	Updates	Avg	Max	Wts	Stl	Tot	Tot	CI/	Sec				
TPCCB1	BMP1DEB1	1	97	97	97	97	10	10	10	10	0	0	0	0	10	10	0	0	0	0	0	0	0	0	0	0
TPCCB2	BMP1DEB2	1	120	120	120	120	10	10	10	10	0	0	0	0	10	10	0	0	0	0	0	0	0	0	0	0
TPCCB4	BMP1DEB4	1	102	102	102	102	10	10	10	10	0	0	0	0	10	10	0	0	0	0	0	0	0	0	0	0
TPCCB6	BMP1DEB6	1	99	99	99	99	10	10	10	10	0	0	0	0	10	10	0	0	0	0	0	0	0	0	0	0
TPCCB9	BMP1DEB9	1	110	110	110	110	10	10	10	10	0	0	0	0	10	10	0	0	0	0	0	0	0	0	0	0
TPCCN	*SF=L	62	22	31	20	29	14	20	0	0	5	11	0	0	26	36	0	0	0	0	0	0	0	0	62	0
TPCCN	TPCCN	6373	22	32	21	31	14	22	11	20	5	14	1	1	25	39	0	0	0	0	1	0	0	0	0	0
TPCCN	*Total*	6435	22	32	21	31	14	22	11	20	5	14	1	1	25	39	0	0	0	0	1	0	0	0	62	0
TPCCO	*MPP	777	7	15	0	0	1	4	0	0	0	0	0	0	1	4	0	0	0	0	0	0	0	0	0	0
TPCCP	TPCCP	6050	2	2	3	3	1	2	1	1	0	1	2	2	4	4	0	0	0	0	0	0	0	0	0	0
TPCCS	*MPP	569	410	485	0	0	290	369	0	0	0	1	0	0	10	10	0	999	0	0	1	0	0	0	0	0
System Totals		13836	28	485	11	120	19	369	6	20	3	14	1	2	14	39	0	999	0	0	2	0	0	62	0	0

The IMS Performance Analyzer Fast Path Resource Usage and Contention report shows the number of buffers actually used by each Fast Path transaction (see Example 5-7 on page 101).

We recommend that you try to aim for at least 99% of transactions acquiring all their buffer needs from NBA.

You can also use the IMS Performance Analyzer Fast Path Transaction Exception Log report. This report produces similar information to that obtained with the DBFULTA0 utility, but with enhanced data filtering, time precision, and additional fields, such as USERID. Refer to Example 5-8.

Example 5-8 IMS Performance Analyzer Fast Path Transaction Exception Log report

IMS Performance Analyzer																										
Fast Path Transaction Exception Log																										
Log 08Sep2006 16.59.23.79																										
Sync	Point	S	Transact	Routing	P	User	PST	Queue	--Transit Times (Msec)--				Output	--DB Call--		--ADS--		--VSO--		Buf	--DB Wait--					
Time	E		Code	Code	I	ID	ID	Count	In-0	Proc	Out-0	Total	(sec)	DEDB	MSDB	Get	Put	Get	Put	Use	CI	UW	OB	CB		
16:57:58.48			TPCCN	TPCCN		WH001002	11	6	137	17	0	154	0.00	35	0	13	0	0	1	23	0	0	0	0		
16:57:58.50			TPCCN	TPCCN		WH000406	11	5	59	19	0	78	0.00	35	0	13	0	0	1	23	0	0	0	0		
16:57:58.52			TPCCN	TPCCN		WH000602	11	7	79	21	0	100	0.00	55	0	19	0	0	1	34	0	0	0	0		
16:57:58.54			TPCCN	TPCCN		WH001904	11	6	71	15	0	86	0.00	27	0	9	0	0	1	16	0	0	0	0		
16:57:58.54			TPCCP	TPCCP		WH000702	6	0	0	5	0	5	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.55			TPCCN	TPCCN		WH001706	11	5	87	16	0	103	0.00	39	0	10	0	0	1	21	0	0	0	0		
16:57:58.59			TPCCN	TPCCN		WH000304	11	6	104	33	0	137	0.00	39	0	13	0	0	1	24	0	0	0	0		
16:57:58.61			TPCCN	TPCCN		WH001308	11	5	112	16	0	128	0.00	27	0	10	0	0	1	18	0	0	0	0		
16:57:58.61			TPCCP	TPCCP		WH000608	6	1	0	4	0	4	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.62			TPCCP	TPCCP		WH000610	6	0	5	5	0	10	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.63			TPCCN	TPCCN		WH000308	11	4	97	19	0	116	0.00	55	0	17	0	0	1	32	0	0	0	0		
16:57:58.67			TPCCN	TPCCN		WH000506	11	4	118	36	0	154	0.00	59	0	17	0	0	1	33	0	0	0	0		
16:57:58.67			TPCCP	TPCCP		WH000504	6	1	0	31	0	31	0.00	5	0	1	0	0	2	4	1	0	0	0		
16:57:58.68			TPCCP	TPCCP		WH000902	6	0	22	6	0	28	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.68			TPCCN	TPCCN		WH000706	11	4	155	17	0	172	0.00	43	0	13	0	0	1	25	0	0	0	0		
16:57:58.70			TPCCN	TPCCN		WH001208	11	4	115	17	0	132	0.00	39	0	12	0	0	1	23	0	0	0	0		
16:57:58.72			TPCCP	TPCCP		WH000804	6	0	0	5	0	5	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.73			TPCCN	TPCCN		WH001204	11	4	116	20	0	136	0.00	59	0	17	0	0	1	33	0	0	0	0		
16:57:58.73			TPCCP	TPCCP		WH001510	6	1	0	5	0	5	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.73			TPCCP	TPCCP		WH001604	6	0	6	3	0	9	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.75			TPCCN	TPCCN		WH001210	11	3	98	21	0	119	0.00	51	0	16	0	0	1	30	0	0	0	0		
16:57:58.77			TPCCN	TPCCN		WH000208	11	3	82	15	0	97	0.00	23	0	10	0	0	1	17	0	0	0	0		
16:57:58.78			TPCCP	TPCCP		WH001304	6	0	0	5	0	5	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.79			TPCCN	TPCCN		WH000210	11	3	69	19	0	88	0.00	47	0	14	0	0	1	27	0	0	0	0		
16:57:58.79			TPCCP	TPCCP		WH001404	6	0	0	5	0	5	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.80			TPCCP	TPCCP		WH001602	6	0	0	4	0	4	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.81			TPCCN	TPCCN		WH000408	11	2	84	19	0	103	0.00	47	0	14	0	0	1	27	0	0	0	0		
16:57:58.82			TPCCP	TPCCP		WH001004	6	0	0	4	0	4	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.83			TPCCN	TPCCN		WH002002	11	1	44	15	0	59	0.00	27	0	10	0	0	1	18	0	0	0	0		
16:57:58.86			TPCCN	TPCCN		WH001704	11	1	51	25	0	76	0.00	59	0	19	0	0	1	35	0	0	0	0		
16:57:58.86			TPCCP	TPCCP		WH001708	6	2	0	34	0	34	0.00	5	0	1	0	0	2	4	1	0	0	0		
16:57:58.87			TPCCP	TPCCP		WH000206	6	1	25	4	0	29	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.87			TPCCP	TPCCP		WH000508	6	0	20	5	0	25	0.00	5	0	1	0	0	2	4	0	0	0	0		
16:57:58.87			TPCCN	TPCCN		WH000802	11	0	21	17	0	38	0.00	23	0	10	0	0	1	17	0	0	0	0		

16:57:58.88	TPCCP	TPCCP	WH000106	6	0	0	3	0	3	0.00	5	0	1	0	0	2	4	0	0	0	0
16:57:58.90	TPCCP	TPCCP	WH000110	6	0	0	3	0	3	0.00	5	0	1	0	0	2	4	0	0	0	0
16:57:58.92	TPCCN	TPCCN	WH001504	11	3	0	44	0	44	0.00	51	0	17	0	0	1	31	0	0	0	0
16:57:58.93	TPCCP	TPCCP	WH001010	6	0	0	4	0	4	0.00	5	0	1	0	0	2	4	0	0	0	0
16:57:58.94	TPCCP	TPCCP	WH000806	6	0	1	6	0	7	0.00	5	0	1	0	0	2	4	0	0	0	0
16:57:58.96	TPCCN	TPCCN	WH000102	11	3	24	31	0	55	0.00	63	0	18	0	0	1	35	0	0	0	0
16:57:58.97	TPCCP	TPCCP	WH001102	6	0	0	4	0	4	0.00	5	0	1	0	0	2	4	0	0	0	0

IMS Performance Analyzer

Fast Path Transaction Exception Summary

Log 08Sep2006 16.59.23.79

Transact Code	Routing Code	Resp Count	--Average Transit Time--				--Maximum Transit Time--				----- DB Calls -----				----- DB Waits -----							
			Input	Pgm	Output	Total	Input	Pgm	Output	Total	DEDB	MSDB	CI	UOW	OBA	CB	Avg	Mx	Avg	Mx	Avg	Mx
TPCCN	TPCCN	3787	402	26	0	428	1898	449	3	1920	43	63	0	0	2	38	0	0	0	0	0	0
TPCCP	TPCCP	3904	3	8	0	11	183	222	1	222	5	5	0	0	0	10	0	0	0	0	0	0
System Totals		7691	200	17	0	216	1898	449	3	1920	24	63	0	0	1	38	0	0	0	0	0	0

IMS Performance Analyzer

Fast Path Transaction Exception Recap

Log 08Sep2006 16.59.23.79

Total number of Fast Path transactions examined (Total Traffic) 7691
 Number of Fast Path exception transactions (Exception Traffic) 7691

Expectation Set used in the analysis . . . N/A in Library N/A

Breakdown of exceptions by type:

IFP transactions where the expectation was not met . . . 7691
 IFP transaction Sync failures 0
 IFP transactions where no dequeue record was found . . . 0
 Non-IFP transactions (including Sync failures) 0

Total Traffic Data Set N/A
 Exception Traffic Data Set . . . N/A

The number reported under the *Bufuse* column on the first line for every transaction is the total number of buffers used, irrespective of whether they were NBA only or NBA + OBA. The Buffers line in the report gives the detailed breakdown of the number used within NBA and OBA.

The report can provide a breakdown of buffer usage by type. It details the number of NBA, OBA, NRDB (Non-related Buffers for SDEP and MSDB use), number of times buffer stealing was invoked, number of times the program waited for a buffer to become available, number of buffers written with OTHREADs, as well as the number of buffer sets used by HSSP and the high-speed reorganization utility.

5.15.19 Overflow Buffer Allocation (OBA)

If your program requires more than its NBA, IMS can provide additional buffers. The number allowed is specified by the OBA parameter on the region procedure. However, IMS permits only a single program to access its OBA buffers at any point in time and uses the OBA latch to enforce this (generally, OBA is required only by update programs).

The OBA latch is released when the holding program reaches sync point. If the latch is unavailable because another program is using its OBA buffers, the region waits until the latch becomes available. At any time, only the largest OBA requested by a region is page fixed in the Fast Path buffer pool. We recommend that you allocate sufficient NBA for the majority of work units, so that OBA is rarely used.

You can monitor the OBA latch contention by means of the *Latch Statistics* section of IMS Performance Analyzer Internal Resource Usage report. We recommend that you investigate any value different from 0 total IWAITs in the OBA latch line.

5.16 Non-recoverable databases

A database can be registered with DBRC as non-recoverable. This does not log any after-images of any database changes that take place. The before-images are still logged, because they are required for dynamic backout. These before-images are kept only on the OLDS and are not archived to a SLDS.

A database can be registered with DBRC as non-recoverable by using the NONRECOV option on the INIT.DB or CHANGE.DB commands.

If a database is marked as non-recoverable, then image copies are not required after initial load (PROCOPT=L) or reorganizations of the database.

Typical uses of a non-recoverable database are with indexes, either primary or secondary, where you have an index rebuilding tool and also, databases that the application has decided if anything goes wrong with them, a DASD failure, for example, they are to be made empty. Non-recoverable databases are not backed up.

5.17 OSAM as opposed to VSAM

Some IMS full function database data sets can be either OSAM or VSAM ESDSs. These are HDAM data sets, PHDAM database data sets, HIDAM database data sets, and PHIDAM database data sets. You cannot use OSAM for HIDAM indexes, PHIDAM indexes, secondary indexes, or HALDB ILDSs. These data sets must be VSAM KSDSs.

OSAM is the preferred access method whenever it can be used. OSAM has several advantages.

OSAM has a superior way of writing multiple blocks. When an application sync point occurs, IMS sorts the altered OSAM blocks by physical location within volumes. IMS writes blocks which are on the same volume with chained writes. This reduces the processor time required on the z/OS system. IMS does the chained writes for different volumes in parallel. This reduces the elapsed time for the writes.

OSAM has a shorter processor instruction path length for most of its processing. This is especially significant in online systems. This means that OSAM typically requires less processor time than VSAM.

OSAM has OSAM sequential buffering. VSAM does not have a similar capability. When OSAM sequential buffering is invoked, IMS does anticipatory reads. For sequential processes, OSAM sequential buffering reads blocks into buffers before the application requires them. This eliminates the wait time for reads and can significantly reduce the elapsed time for sequential processing.

For HDAM and HIDAM, OSAM data sets can be up to 8 gigabytes in size. HDAM and HIDAM ESDSs are limited to 4 gigabytes.

You can reuse OSAM data sets. When you reorganize a database, you do not have to delete and redefine OSAM data sets. When you reorganize HDAM and HIDAM databases, you must

delete and redefine VSAM data sets. This VSAM requirement does not apply to PHDAM and PHIDAM databases.

There is a warning about the reuse of OSAM data sets. You should not reuse multivolume OSAM data sets. If you do not scratch and reallocate a multivolume OSAM data set before reusing it, an invalid end-of-file mark might be left on the last volume of the data set. This can lead to lost data. You can use HALDB to avoid database data sets that are so large that they must span multiple volumes.

5.17.1 Performance results on OSAM and VSAM

Tests were run in a controlled environment in the Silicon Valley Laboratory using 10 HIDAM databases. The first set of tests were run with the databases defined with VSAM, and then a second set of tests were run with OSAM using the same workload that was used in the first test.

Set one of the BMPs tested consisted of three BMPs each executing 2 000 000 total database calls. There were 10 qualified GHU calls performed along with 1 000 000 qualified GHN calls and 1 000 000 replace calls. Table 5-5 shows the results of BMP set one.

Table 5-5 BMP set one

Type	RMF CPU%	Average total task CPU time in seconds	Total elapsed time in minutes	Delta on total elapsed time
VSAM	50.33	168	8.71	
OSAM	51.87	136	6.31	27.53% reduction
OSAM SB	54.58	138	6.33	27.34% reduction

Set two of the BMPs tested consisted of four BMPs each executing 4 500 000 total database calls. There was one qualified GHU call performed along with 1 000 qualified GHN calls, 1 000 replace calls, and 4 000 000 GN calls. Table 5-6 shows the results of BMP set two.

Table 5-6 BMP set two

Type	RMF CPU%	Average total task CPU time in seconds	Total elapsed time in minutes	Delta on total elapsed time
VSAM	29.86	98	5.45	
OSAM	27.00	57	3.50	35.78% reduction
OSAM SB	78.62	61	1.16	78.59% reduction

5.17.2 Recommendation summary for OSAM as opposed to VSAM

Use OSAM data sets, not VSAM ESDSs, with PHDAM, PHIDAM, HDAM, and HIDAM databases.

5.18 Buffer life concept

To help determine if more OSAM or VSAM buffers might avoid some reads, you can use a *buffer life* concept.

Buffer life is a measure of how long a block or CI remains in the OSAM or VSAM buffer pool, before its buffer is used for reading another block or CI. If the block or CI is there for less than the life of a transaction or BMP synchronization interval, it is likely that repeated reads to the same block might be needed to complete the work, and this should be avoided.

For online transactions, it is often beneficial to keep data in the buffer pool across user think time. This allows successive transactions from the same user to use the same data without incurring the cost of rereading it. Because user think times are often in the range of 15 seconds to a minute, buffer lifetimes of more than a minute are recommended.

Buffer life is an average. Some blocks might remain in the pool for much longer times and some for much shorter times. For this reason, we recommend that buffer life is longer than average think times. Five to 10 minutes is a reasonable goal. If buffer lifetimes are over 15 minutes, then it seems unlikely that more buffers would lead to significantly fewer reads.

Buffer life for a OSAM or VSAM pool can be calculated with the following formula:

$$\text{Buffer life} = (\text{number of buffers} \times \text{elapsed time}) / \text{number of reads}$$

5.19 Overflow sequential access method (OSAM)

Overflow sequential access method (OSAM) is unique to, and is supplied with, IMS. IMS communicates with OSAM using the OPEN, CLOSE, READ, and WRITE macros.

An OSAM database has these characteristics:

- ▶ An OSAM data set can be read using either the BSAM or QSAM access method.
- ▶ An OSAM data set does not need to be formatted before use.
- ▶ An OSAM data set can use fixed-length records blocked or unblocked.
- ▶ An OSAM data set can span multiple volumes.
- ▶ An OSAM data set has an 8 GB size limit.

Detailed information about how data is organized in an OSAM database can be found in the *IMS Version 9: Administration Guide: Database Manager*, SC18-7806. For information about defining OSAM subpools, refer to *IMS Version 9 Installation Volume 2: System Definition and Tailoring*, GC18-7823.

5.19.1 Tuning OSAM buffers

There is one OSAM buffer pool, typically divided into multiple subpools. A *subpool* is a set of buffers that are the same size, and the OSAM buffer pool is allowed to have multiple subpools that all have same-size buffers. The definitions of the subpools are put in the DFSVSMxx member (DFSVSAMP DD statement for batch).

The first thing you need to do when tuning OSAM buffers is make sure that you have a valid buffer size. Refer to Table 5-1 on page 67 for valid OSAM buffer sizes. There is a minimum number of four buffers required for each subpool and a maximum of 32 767. You can have multiple subpools of the same size. This is useful, for example, for isolating lesser used databases from the effects of much busier and “buffer-greedy” databases.

IMS batch and utility jobs use the DFSVSAMP data set to specify buffers, and Transaction Manager and DB Control use the IMS.PROCLIB member, DFSVSMnn. Look at Example 5-9.

Example 5-9 OSAM buffers

```
IOBF=(bufsize,# buffers,fix1,fix2,id)
DBD=dbdname(dataset number,id)
```

- ▶ IOBF= is a required keyword for the OSAM subpool definition.
- ▶ bufsize specifies the length of the buffers in the subpool.
- ▶ # buffers specifies the number of buffers in the subpool.
- ▶ fix1 specifies the buffer long-term page fixing option. If you specify Y, all buffers and buffer prefixes associated with this subpool are long-term and page fixed at initialization of the subpool. We recommend that you always page fix buffer prefixes, because OSAM fixes buffers and prefixes at I/O time to prevent page faults. Rather than suffering the overhead of taking the page fault, paging in the storage pool, fixing the storage pool, and then unfixing it, it is better to fix them at initialization time from a performance perspective.
- ▶ fix2 specifies the buffer prefix long-term page fixing option. If you specify Y, all buffer prefixes associated with this subpool and the subpool header are long-term and page fixed at initialization of the subpool.
- ▶ id specifies a user-defined identifier to be assigned to a subpool.
- ▶ DBD= is used to assign the database data set to the OSAM buffer subpool with a matching ID.
- ▶ dataset number is determined by the order of the DATASET macros in the specified DBD for non-HALDB databases and for HALDBs, specify the data set number as an alphabetic character.

When an OSAM database is “opened” by IMS, the database’s data set is assigned to a subpool based on the DBD subpool ID assignment if one is there. If not, then IMS assigns it to a subpool based on the blocksize of the data set. See Example 5-10.

Example 5-10 OSAM pool assignment

```
IOBF=(04096,1024,Y,Y,PROD)
IOBF=(04096,0123,N,Y)
IOBF=(18432,0050,Y,Y,CUST)
DBD=DHZDBCP1(1,PROD)
DBD=GMZDCUP1(2,CUST)
```

In Example 5-10, OSAM pool assignment, database DHZDBCP1 has been assigned to a buffer pool called PROD. That pool has 1 024 4 K buffers (or 4 194 304 bytes of storage) assigned to it for its individual use. Also, database GMZDCUP1 has been assigned to a buffer pool called CUST. That pool has 50 18 K buffers (or 921 600 bytes of storage) assigned to it for its individual use.

To tune the OSAM buffer pool, you need to monitor it. A good tool for that is the *DC Monitor*. It is part of the base product of IMS and is turned on by issuing this command:

/TRACE SET ON MONITOR ALL

To turn the monitor off, issue this command:

/TRACE SET OFF MONITOR

We recommend that you run the monitor twice a day, during the two heaviest load times of the day. Example 5-11 is one of the OSAM buffer pool reports. There is one report for each pool.

Example 5-11 OSAM buffer pool statistics

IMS MONITOR	*** BUFFER POOL STATISTICS ***	TRACE START 2006 258, 13:43:12
-------------	--------------------------------	--------------------------------

DATA BASE BUFFER POOL		
	FIX PREFIX/BUFFERS	Y/Y
	SUBPOOL ID	PROD
	SUBPOOL BUFFER SIZE	4096
	TOTAL BUFFERS IN SUBPOOL	1024

	DIFFERENCE
1 NUMBER OF LOCATE-TYPE CALLS	1981
2 NUMBER OF REQUESTS TO CREATE NEW BLOCKS	18
3 NUMBER OF BUFFER ALTER CALLS	4467
4 NUMBER OF PURGE CALLS	1334
5 NUMBER OF LOCATE-TYPE CALLS, DATA ALREADY IN OSAM POOL	139661
6 NUMBER OF BUFFERS SEARCHED BY ALL LOCATE-TYPE CALLS	7006
7 NUMBER OF READ I/O REQUESTS	9797
8 NUMBER OF SINGLE BLOCK WRITES BY BUFFER STEAL ROUTINE	0
9 NUMBER OF BLOCKS WRITTEN BY PURGE	2635
10 NUMBER OF LOCATE CALLS WAITED DUE TO BUSY ID	0
11 NUMBER OF LOCATE CALLS WAITED DUE TO BUFFER BUSY WRT	6
12 NUMBER OF LOCATE CALLS WAITED DUE TO BUFFER BUSY READ	4129
13 NUMBER OF BUFFER STEAL/PURGE WAITED FOR OWNERSHIP RLSE	0
14 NUMBER OF BUFFER STEAL REQUESTS WAITED FOR BUFFERS	0
15 TOTAL NUMBER OF I/O ERRORS FOR THIS SUBPOOL	0
16 NUMBER OF BUFFERS LOCKED DUE TO WRITE ERRORS	0

QUOTIENT :	$\frac{\text{TOTAL NUMBER OF OSAM READS} + \text{OSAM WRITES}}{\text{TOTAL NUMBER OF TRANSACTIONS}}$	= 1.02
------------	--	--------

	DIFFERENCE
17 NUMBER OF BLOCKS READ FROM CF	0
18 NUMBER OF BLOCKS EXPECTED BUT NOT READ	0
19 NUMBER OF BLOCKS WRITTEN TO CF (PRIME)	0
20 NUMBER OF BLOCKS WRITTEN TO CF (CHANGED)	0
21 NUMBER OF BLOCKS NOT WRITTEN (STORAGE CLASS FULL)	0
22 NUMBER OF BLOCKS INVALIDATED WITH XI (VECTOR CALL RET)	0
23 NUMBER OF XI CALLS ISSUED	0

Report analysis

Example 5-11 on page 108 is an OSAM buffer pool report for a subpool with 1 024 buffers, each 4 KB in size.

There are numbers to the left of the line items in the report in Example 5-11 on page 108, but these numbers are for your reference for this section only and do not appear on an actual report:

► Line 2: NUMBER OF REQUESTS TO CREATE NEW BLOCKS

Each count represents a format logical cylinder operation that extended a database data set by one physical DASD cylinder.

One or more databases are likely to require reorganization to regain free space. Database DBDs might need to be changed to increase free space.

► Line 4: NUMBER OF PURGE CALLS

Each count means the database subpools were purged of altered buffers for a dependent region or batch job step.

► Line 5: NUMBER OF LOCATE-TYPE CALLS, DATA ALREADY IN OSAM POOL

Each count reflects the number of times a request for a block was satisfied from the subpool; that is, a read operation was not required. This line item is used to calculate a hit ratio for the subpool (discussed in “OSAM statistics” on page 110).

► Line 7: NUMBER OF READ I/O REQUESTS

The count reported is the number of requests that could not be satisfied from the contents of the subpool and required that a block be read from a database data set on DASD. Use tuning tactics to attempt to eliminate reads or, if they cannot be avoided, to perform them as efficiently as possible. This line item is used to calculate a hit ratio for the subpool (discussed in “OSAM statistics” on page 110).

► Line 8: NUMBER OF SINGLE BLOCK WRITES BY BUFFER STEAL ROUTINE

The count reflects the number of blocks written to database data sets as a result of altered buffer steals. You can eliminate altered buffer steals entirely by judiciously balancing the size (number of buffers) of the pool with timely application program sync points.

► Line 9: NUMBER OF BLOCKS WRITTEN BY PURGE

Because OSAM purge is the most efficient way to write altered OSAM buffers, the tuning tactic is to write all altered buffers with OSAM purge and eliminate all OSAM-altered buffer steals.

Buffer handler contention is represented by lines 10 through 14.

Contention means a request of the buffer handler waits until the condition indicated is reset. These waits, from an IMS Monitor perspective, are reported as NOT-IWAIT time.

► Line 10: NUMBER OF LOCATE CALLS WAITED DUE TO BUSY ID

Each OSAM buffer has a buffer prefix. Whenever a prefix must be changed (to indicate a changed attribute, for example), it is given an attribute of BUSY ID under the subpool latch. Once BUSY ID is set, no other task can access the prefix or its associated buffer contents. Any requests for the buffer prefix while it is busy must wait.

► Line 11: NUMBER OF LOCATE CALLS WAITED DUE TO BUFFER BUSY WRT

A request is made to access the contents of a buffer, but the contents of the buffer are in the process of being written to the proper database data set. The write must complete before any requestors can have access to the contents of the buffer.

► Line 12: NUMBER OF LOCATE CALLS WAITED DUE TO BUFFER BUSY READ

The explanation is the same as for Line 11, except the buffer is busy because a read is in progress (a previous requestor required the same OSAM block). NUMBER OF LOCATE CALLS WAITED DUE TO BUFFER BUSY READ can be thought of as a good form of contention, in that a single read can satisfy multiple requests of the buffer handler.

► Line 13: NUMBER OF BUFFER STEAL/PURGE WAITED FOR OWNERSHIP RLSE

The OSAM steal or purge routine cannot steal or purge a buffer that is owned by another requestor. Steal or purge must wait until ownership is released.

► **Line 14: NUMBER OF BUFFER STEAL REQUESTS WAITED FOR BUFFERS**

The buffer steal routine is required to steal a buffer to satisfy a read request. Counts indicate that the steal routine went through its complete algorithm and could not find a suitable buffer to steal.

OSAM statistics

Several statistics are of use when analyzing an OSAM database subpool. The following are some of the more common statistics used.

Hit ratio

A subpool hit ratio is calculated as follows:

$$\text{Hit Ratio} = \frac{\text{Line 5}}{\text{Line 5} + \text{Line 7}} \times 100 = \text{xx.x\%}$$

For Example 5-11 on page 108, Example 5-11 on page 108, the hit ratio calculation is as follows:

$$\text{Ratio} = \frac{139,661}{139,661 + 9,797} \times 100 = 93.4\%$$

By itself, a hit ratio means nothing. Hit ratios are generally useful when making run-to-run comparisons. If a change is made to a particular subpool and the hit ratio increases, it is highly likely that the increased hit ratio represents an improvement.

Reads per second and writes per second

Reads per second are simply “NUMBER OF READ I/O REQUESTS” (Line 7) divided by the monitor interval in seconds. The tuning goal is to reduce this number.

Writes per second are “NUMBER OF SINGLE BLOCK WRITES BY BUFFER STEAL ROUTINE” (Line 8) plus “NUMBER OF BLOCKS WRITTEN BY PURGE” (Line 9), all divided by the monitor interval in seconds. The tuning goal is to reduce this number.

Buffer life

This statistic is useful for online systems. It is calculated as follows:

$$\text{Buffer Life} = \frac{\text{No. of buffers in subpool} \times \text{Monitor Interval}}{\text{NUMBER OF READ I/O REQUESTS (Line 7)}}$$

Buffer life is the average life in seconds of the contents of a buffer in a subpool. When the average buffer life is in tens of seconds, then the subpool is holding buffer contents across a user's think time. When buffer life is low, such as less than 1 buffer-second per read, it is generally beneficial to dramatically increase the number of buffers in a subpool.

Use buffer life with reads per second. For example, if the number of buffers in a subpool is increased, the goal is for buffer life to increase and reads per second to decrease. If reads per second do not decrease, then the increase in the number of buffers is not beneficial and not worth the extra demand placed on real storage by the additional buffers.

Keep these things in mind as you tune

Things to consider when tuning:

- ▶ On the OSAM Buffer Pool Statistics report (Database Buffer Pool)
 - As you tune the buffers, the difference on the following lines should be going up:
 - NUMBER OF LOCATE-TYPE CALLS, DATA ALREADY IN OSAM POOL (Line 5)
 - NUMBER OF BLOCKS WRITTEN BY PURGE (Line 9)
 - As you tune the buffers, the difference on the following lines should be going down:
 - NUMBER OF REQUESTS TO CREATE NEW BLOCKS (Line 2)
 - NUMBER OF READ I/O REQUESTS (Line 7)
 - NUMBER OF SINGLE BLOCK WRITES BY BUFFER STEAL ROUTINE (Line 8)
- ▶ As you tune the buffers, there should be an increase in the percentage found in the pool (Hit Ratio).
- ▶ As you tune the buffers, there should be a decrease in the percentage of buffer flushes. *Buffer flushes* are defined as:

$$\text{Buffer flushes} = \frac{\text{Line 8}}{\text{Line 8} + \text{Line 9}} \times 100 = \text{xx.x\%}$$

For this example, the buffer flushes ratio calculation is as follows:

$$\text{Buffer flushes} = \frac{0}{2,635 + 0} \times 100 = 0\%$$

OSAM tuning general rules

These are good rules to follow:

- ▶ 60% or better hit ratio.
- ▶ Always page fix buffer prefixes.
- ▶ Page fix buffers if real storage permits.
- ▶ Isolate high activity databases in their own subpool.
- ▶ Match block size with a valid OSAM buffer size.
- ▶ For sequential-read BMPs and batch:
 - Use the largest block size possible.
 - Use sequential buffering.
 - Use 3990-3 cache controller.

5.19.2 OSAM data set notes

Do not use the BLOCK= parameter in the DBD source; use SIZE= parameter instead.

If you make minor changes to your DBD (for example, changing the number of RAPs) and neglect to update BLOCK=, you will not use the buffer size that you expected. This can be very bad for performance.

The OSAM data set can be allocated at either load time through JCL or preallocated using something similar to IEFBR14.

Do not specify any DCB parameters in the JCL when allocating the database data set. IMS creates the DCB information from the DBD information in DBDLIB. Always include secondary space allocation.

Preallocate only the number of volumes for OSAM data set extents that will be used during initial load or reload process. If a volume is not used, it does not have a valid end-of-file (EOF) mark, which causes OSAM to scan the entire DB to find the true EOF mark.

Do not reuse multivolume OSAM data set extents without first scratching and reallocating the space. There can be an invalid EOF that was left on the last volume. Inserts can be placed in the database after the old end-of-file marker. Some utilities, such as Image Copy, ignore anything after the new EOF.

OSAM data sets have the following characteristics:

- ▶ A maximum of 60 DASD extents.
- ▶ The maximum data set size is 8 gigabytes.
- ▶ Fixed length records, unblocked.

Here are some advantages of OSAM:

- ▶ Shorter path length than VSAM
- ▶ More choices for buffer sizes
- ▶ One I/O request writes multiple blocks to a track:
 - DB LOAD or DB RELOAD
 - Synchronization point

OSAM is good. VSAM is used for indexes.

5.19.3 OSAM sequential buffering

OSAM sequential buffering can be invoked in four ways. It can be invoked by an exit in IMS, in a PSB, by coding a control card in your JCL, or by having SBONLINE keyword included in DFSVSMxx IMS.PROCLIB member. In tests that have been run on a 721 437 record database (3 790 161 total segments) with an average database record length of 3 079 bytes, using get next processing, it takes 13 minutes without the use of sequential buffering. It only took five minutes with OSAM sequential buffering.

Sequential buffering can run in DLI, BMP, and MPP modes. To use sequential buffering in the BMP environment, the keyword SBONLINE must be in the DFSVSMnn member in IMS.PROCLIB.

Sequential buffering evaluates whether it should read ahead, based on how the program is accessing the data. If the evaluation shows that the program is randomly looking at the data, read ahead is not invoked. After another five hundred DL/I calls, sequential buffering evaluates the way that the program is accessing the data again. If this time, sequential buffering thinks you are sequentially going through the data, it turns on sequential buffering.

The exit for IMS is in SDFSRESL. If member DFSSBUX0 is there, then sequential buffering is invoked. IBM has supplied five sample exits in SDFSRESL that can be used. The first routine (DFSSBU1) disallows sequential buffering. Routines two through four (DFSSBU2 - DFSSBU4) allow sequential buffering. Exit routine nine (DFSSBU9) allows sequential buffering only for certain hours during the day. Look in the *IMS Version 9: Customization Guide*, SC18-7817, for the coding of these exits.

The second way to specify the use of sequential buffering is in the PSB on the PCB level. The keyword is "SB=COND". This tells IMS that you want to conditionally use sequential buffering. The default is "SB=NO". Review the *IMS Version 9 Utilities Reference: System*, SC18-7834 manual for more information.

Sequential buffering can be requested in the JCL that you are executing by a “//DFSCTL” statement. Through this control card, you can tell IMS to use sequential buffering, the databases on which to use it, the PSB on which to use it, or the DD name on which you want to use sequential buffering.

You can also tell IMS how many buffer sets to use. Buffer sets are what sequential buffering uses to hold data. Normally, IMS reads in one block at a time. Sequential buffering reads in 10 blocks at a time into these sets of buffers. The default buffer set is 4 and can be raised up to 25 sets. Sequential buffering fills 10 buffers and passes the data to OSAM buffers as they are requested. While that data is being given to OSAM, sequential buffering might read ahead another 10 buffers, and wait until OSAM starts requesting those buffers. Sequential buffering tries to keep one set ahead of OSAM, so you should base your selection of the number of buffer sets on the number of tasks that can be using sequential buffering at any given time. Example 5-12 on page 113 is the syntax of the control statement. SBPARM starts in column one.

Example 5-12 DFSCTL control statement

```
SBPARM  ACTIV=COND,
        BUFSETS=16
```

Look in the *IMS Version 9 Installation Volume 2: System Definition and Tailoring*, GC18-7823, manual for more information about what can be coded on this card.

Another statement to code in your JCL is the “//DFSSTAT” statement. This statement gives information about what sequential buffering did or did not do. You would only code this statement to see if a job is a good candidate for sequential processing. If the statistics show that this job is doing sequential processing, then you would turn on sequential buffering and use this output as an indicator to see how much sequential buffering helped. Look at the statistics in Example 5-13 to see if sequential buffering would be helpful.

Example 5-13 PST accounting statistics

```
*** PST ACCOUNTING STATISTICS ***
DB GU CALLS                0
DB GN CALLS                3,790,161
DB GNP CALLS               0
DB GHU CALLS               0
DB GHN CALLS               0
DB GHNP CALLS              0
DB ISRT CALLS              0
DB DLET CALLS              0
DB REPL CALLS              0
DB CALLS (TOTAL)          3,790,161
DB DEQ CALLS               0
```

Because this job is doing more get next than get unique calls, this program is a good candidate for sequential buffering. The next report to look at is “SEQUENTIAL BUFFERING SUMMARY FOR THE APPLICATION” in Example 5-14. This report tells you if sequential buffering was turned on.

Example 5-14 Sequential buffering summary for the application

```
*** SEQUENTIAL BUFFERING SUMMARY FOR THE APPLICATION ***

DFSSBUXO DISALLOWED USAGE OF SB:          NO
DFSSBUXO REQUESTED CONDITIONAL SB ACTIVATION: YES
AT LEAST ONE SB= KEYWORD IN PSB:          YES
```

AT LEAST ONE SBPARM CONTROL STMT FOR APPLICATION:	YES
SBPARM CONTROL CARD(S) READ FROM //DFSCTL:	YES
AT LEAST ONE SBPARM PSB= SPECIFIED THAT MATCHED PSB:	NO
AT LEAST ONE SBPARM DB= SPECIFIED THAT MATCHED DB:	NO
AT LEAST ONE SBPARM PCB= SPECIFIED THAT MATCHED PCB:	NO
AT LEAST ONE SBPARM DD= SPECIFIED THAT MATCHED DD:	NO

Because sequential buffering was not turned off and AT LEAST ONE SB= KEYWORD IN PSB says YES (this is a BMP run), sequential buffering is used in this run.

To find how much I/O was saved, find the report and look for “NUMBER OF SEARCH REQUESTS ISSUED BY OSAM BH: SEARCH”, Example 5-15 on page 114, and obtain the number across from it (in this case, 95 240). This is the number of I/O requests that would have happened if sequential buffering was not used. Next, you need to add together these two numbers. “NUMBER OF READ I/O: RANDOM READ” and “NUMBER OF READ I/O: SEQUENTIAL READ.” This is the number of I/O that was incurred when the job ran. (Here it is 1 600 + 9 466 = 11 066.) You then take this number and subtract it from the number in the search to get the number of I/Os that was saved. Here we have 95 240 - 11 066 = 84 174 I/Os saved by using sequential buffering. If you were to multiply this number by the time that it takes to do an I/O, you would have a good idea of how much time was saved (84 174 X 0.0053 = 446.1222 seconds saved, or almost 7.5 minutes). This job’s I/O time would have been 504.772 seconds (95 240 X 0.0053) without sequential buffering, yet with sequential buffering, its time was 58.6498 seconds (11 066 X 0.0053).

Example 5-15 NUMBER OF SEARCH REQUESTS ISSUED BY OSAM BH: SEARCH

NUMBER OF SEARCH REQUESTS ISSUED BY OSAM BH:	
SEARCH	95,240
NUMBER OF READ I/O:	
RANDOM READ	1,600
SEQUENTIAL READ	9,466
NUMBER OF BLOCKS READ:	
TOTAL NUMBER BLOCKS READ	96,260
NBR BLOCKS READ AT RANDOM	1,600 PCT OF TOTAL: 1.66
NBR BLOCKS READ SEQUENTIALLY	94,660 PCT OF TOTAL: 98.33
PERCENT READ PER SEARCH REQUEST	11.61
NUMBER OF SEQUENTIAL I/O ERRORS	0

Another number to look at is “NBR BLOCKS READ SEQUENTIALLY.” If this number is high, then sequential buffering probably helped speed up this run. If “PERCENT READ PER SEARCH REQUESTS” is low, then sequential buffering helped speed up this run.

Example 5-16 Sequential buffering (SB) DETAIL STATISTICS (PAGE A)

*** SB DETAIL STATISTICS (PAGE A) ***

PSB	PFZP9990
DB	PFZDABP1
PCB	
DB-PCB NBR	44
DSG-CB NBR	2
DD	PFZDABP1
DB-ORG	HIDAM
DD-TYPE	*PSDATA

NBR OF BUFSETS

16

```
** NUMBER OF SEARCH REQUESTS ISSUED BY OSAM BH:
    SEARCH                                     94,366

** NUMBER OF READ I/O:
    TOTAL                                     10,193
    RANDOM READ                             727
    SYNCHRONOUS SEQUENTIAL READ              3
    OVERLAPPED SEQUENTIAL READ               9,463

** NUMBER OF BLOCKS READ:
    TOTAL                                     95,387
    RANDOM READ                             727 PCT OF TOTAL: .76
    SYNCHRONOUS SEQUENTIAL READ              30 PCT OF TOTAL: .03
    OVERLAPPED SEQUENTIAL READ               94,630 PCT OF TOTAL: 99.20

** AVERAGE I/O WAIT TIMES (MILLIS):
    RANDOM READ                             3.47
    SYNCHRONOUS SEQUENTIAL READ              63.66
    OVERLAPPED SEQUENTIAL READ               4.89
```

The report in Example 5-16 on page 114 is for each database that IMS reviewed for sequential buffering. The “NBR OF BUFSETS” is the number of buffer sets that you designate in the “DFSCTL” statement. You can also find the number of I/Os saved by subtracting “NUMBER OF READ I/O: TOTAL” from the “NUMBER OF SEARCH REQUESTS ISSUED BY OSAM BH: SEARCH.”

If “NUMBER OF BLOCKS READ: RANDOM READ” is high, and you were going through the database sequentially, then the database probably needs to be reorganized. If “NUMBER OF BLOCKS READ: OVERLAPPED SEQUENTIAL READ” is high, then sequential buffering helped speed up the job.

Sequential buffering is beneficial in online image copy, HD reorganization unload, partial database reorganization, surveyor, database scan, and database prefix update.

5.20 Virtual storage access method (VSAM)

VSAM is one of several DASD access methods used in z/OS. IMS uses two of the five storage methods in VSAM: the key-sequenced data set (KSDS) and the entry-sequenced data set (ESDS).

5.20.1 Tuning VSAM buffers

VSAM Local Shared Resource (LSR) pool contains buffers used by the VSAM index and data components. Buffers of equal length are combined into subpools. The definitions of the subpools are put in the DFSVSMxx member (DFSVSAMP DD statement for batch) and are allocated by CI size.

The first thing you need to do when tuning VSAM buffers is make sure that you have a valid buffer size. Refer to Table 5-2 on page 68 for valid VSAM buffer sizes. There is a minimum number of three buffers required for each subpool and a maximum of 32 767. All subpools are allocated on 4 096 page boundaries. What this means is that when you add up the space you have allocated for your subpool, it must be divisible by 4 096. Subpools can be specified as

either “D” for data (only the data portion of a KSDS or ESDS can use this pool) or “I” for index (only the index portion of a KSDS can use this pool) or none at all. You can have multiple subpools of the same size, but if you have an index only subpool, you must have a data only subpool too. This is useful for isolating lesser used databases from the effects of much busier and “buffer-greedy” databases.

IMS batch and utility jobs use the DFSVSAMP data set to specify buffers, and Transaction Manager and DB Control use the IMS.PROCLIB member, DFSVSMnn. Refer to Example 5-17.

Example 5-17 VSAM buffers

```
POOLID=id, FIXDATA=, FIXINDEX=, FIXBLOCK=
VSRBF=buffer size, number of buffers, type, HS, HSn
DBD=dbdname(dataset number, id, ERASE=, FREESPACE=)
```

The fields in Example 5-17 on page 116 are:

- ▶ POOLID= is a required keyword for VSAM shared resource pool definition. It must begin in the first position of the control statement. The total number of POOLIDs must not exceed 16.
- ▶ id is a one- to four-character alphanumeric field that specifies a user identifier assigned to a shared resource pool that is used with the DBD statement to direct a given data set to a specific shared pool.
- ▶ FIXDATA= is either YES or NO and specifies the data shared resource pool long-term page fixing option.
- ▶ FIXINDEX= YES causes all buffers in the index shared resource pool to be long-term page fixed at initialization of the shared resource pool. A NO does not long-term page fix the buffers.
- ▶ FIXBLOCK= specifies the I/O-related control block's long-term page fixing option. If YES is specified, all I/O-related control blocks are long-term page fixed at initialization of the shared resource pool. If NO is specified, no I/O-related control blocks are long-term page fixed. We recommend that you always specify YES for this option.
- ▶ VSRBF= is the keyword for VSAM subpool definition.
- ▶ buffer size is a three- to five-digit number specifying the buffer size for this subpool.
- ▶ number-of-buffers is a one- to five-digit number (3 to 32 767) specifying the number of buffers in this subpool. If you specify a number of buffers less than the minimum number required, IMS increases the number to the minimum and issues a warning message.
- ▶ type is a one-character field that specifies whether the subpool in the shared resource pool is an index subpool (I) or a data subpool (D). This parameter is optional. When you do specify D or I in any VSRBF statement, you *must* also specify a type of I for each size that might be required to handle index buffering requests.
- ▶ HS is for HS0 or HSR. These parameters optionally specify the kind of action IMS should take if Hiperspace™ (extended storage on z/OS) buffering for this subpool is not available. HS0 Indicates that Hiperspace buffering is optional, and IMS can continue initialization without Hiperspace buffering. HSR Indicates that Hiperspace buffering is required, and IMS must terminate if Hiperspace buffering is not available.
- ▶ HS_n= is an optional one- to eight- digit number *n* ranging from (3 to 16 777 215) that specifies the number of Hiperspace buffers to build for this subpool. HS0, HSR, and HS_n are valid only on 4 K and larger boundaries.
- ▶ DBD= specifies, if coded, that the data set having a matching ID parameter, as defined on the POOLID shared pool definition statement, is to be directed to the indicated shared

resource pool. Then, a subpool within the assigned shared resource pool is assigned to the data set based upon buffer length.

- ▶ **dataset** number identifies the specific data set of a data set group within a database (identified by the **dbdname** parameter) that requests assignment of a specific shared pool. The number is an IMS internally assigned value between 1 and 10. For data organizations such as primary index, unique secondary index, and HISAM without dependent segments, the primary data set of the data set group is assigned data set number 1. No secondary data set of the data set group exists for these data organizations.

For HALDBs, specify the data set number as an alphabetic character. The valid data set characters defined for HALDB partition data sets are A through J, L, and X. (When HALDB Online Reorganization is used, data sets M through V and Y are automatically directed to the same shared resource pools for data sets A through J and X. The specification of M through V and Y are not valid.)

- ▶ **ERASE=** Is either a YES or a NO and indicates the treatment of logical records that are deleted. YES indicates that the deleted record should be erased. NO indicates that the deleted record should not be erased, but should be marked as a deleted record. This applies only to KSDS data sets.
- ▶ **FREESPACE=** is either a YES or a NO indicating the treatment of the defined free space percent in the KSDS. If YES is specified, the defined free space should be preserved. If NO is specified, the defined free space should not be preserved.
- ▶ The word **OPTIONS** starting in position one identifies the **OPTIONS** statement.
- ▶ **BGWRT=** YES or NO, which specifies whether the background write function of the buffer handler is to be active. You can also activate background write by coding **BGWRT=(YES,n)** or omitting the parameter. *n*: is a two-digit number from 10 to 99 specifying the percentage of buffers in each subpool to be considered as candidates for writing by the background write function. For an explanation of background write, see “VSAM background write” on page 118 or “Determining Which VSAM Options to Use” in *IMS Version 9: Administration Guide: Database Manager*, SC18-7806.
- ▶ **VSAMPLS=LOCL** specifies that the VSAM shared resource pools are to be built.

For more information about these parameters and many more, consult the *IMS Version 9 Installation Volume 2: System Definition and Tailoring*, GC18-7823, manual.

In Example 5-18, VSAM pool assignments, there are two pools assigned, **POOLID=1** and **POOLID=SZ**. **POOLID=1** is a general use pool where a 2 K pool is split with a data component and a corresponding index component. The 4 K pool has the same split as the 2 K pool. There are also an 8 K and a 12 K pool specified. The second pool is called **SZ** for the **KCDDSZP1** database's indexes; **KCDDSZ11**, **KCDDSZS1**, and **KCDDSZS2**. Since the application reads these three indexes sequentially as stand-alone databases, their data CI size was allocated at 24 K with an index CI size of 4 K. At a later time, two more indexes were discovered being read sequentially (**SASDDHS1** and **GRVDGMS6**) and were added to the same buffer pool.

Example 5-18 VSAM pool assignments

```
OPTIONS,BGWRT=(YES,45),VSAMPLS=LOCL
POOLID=1,FIXDATA=NO,FIXINDEX=NO,FIXBLOCK=YES
VSRBF=02048,0500,D
VSRBF=02048,0100,I
VSRBF=04096,1436,D
VSRBF=04096,0100,I
VSRBF=08192,0256
VSRBF=12288,0016
POOLID=SZ,FIXDATA=NO,FIXINDEX=NO,FIXBLOCK=YES
```

```
VSRBF=24576,0512,D,HS0,HS19000
VSRBF=04096,0256,I,HS0,HS04688
DBD=KCDDSZI1(1,SZ,ERASE=Y,FREESPACE=N)
DBD=KCDDSZS1(1,SZ,ERASE=Y,FREESPACE=N)
DBD=KCDDSZS2(1,SZ,ERASE=Y,FREESPACE=N)
DBD=SASDDHS1(1,SZ,ERASE=Y,FREESPACE=N)
DBD=GRVDGMS6(1,SZ,ERASE=Y,FREESPACE=N)
```

5.20.2 VSAM background write

This is a VSAM-only feature that uses a lower-priority asynchronous task to write out updated buffers whenever a subpool becomes completely full of updated buffers. The objective is to make buffers eligible for stealing so that applications do not have to wait on a space write before being able to read in a new CI.

The BGWRT task is dispatched as an asynchronous low-priority task in either the IMS control region (with LSO=Y) or the DL/I separate address space (with LSO=S).

The BGWRT task processes each VSAM subpool in turn. For each subpool, it examines the specified percentage of buffers on the least recently used (LRU) chain. Any modified buffers it finds are written to DASD.

Background write does not prevent reuse of these buffers. If a subsequent request requires the data in the buffer before the buffer manager needs the buffer for a new block, the buffer is used to satisfy the request and is placed on the top of the buffer-use chain.

5.20.3 VSAM hiperspace buffers

Hiperspace is a z/OS facility for storing data, specifically geared for high-performance reads and writes, in memory. IMS provides support for VSAM Hiperspace buffering capability.

Hiperspace buffering offers potential DL/I performance improvements through VSAM I/O avoidance. This is achieved by reducing the requirement for DASD I/O as VSAM finds reassessed buffers in Hiperspace.

The following benefits can be obtained by using VSAM Hiperspace:

- ▶ Significant improvements in VSAM buffer hit ratio
- ▶ VSAM I/O reduction
- ▶ Reduction in internal elapsed time and region occupancy

When considering VSAM Hiperspace implementation, you should weigh a number of factors to assess potential benefits. We recommend that you evaluate each of the following as it applies to your situation:

- ▶ A primary consideration is to determine the amount of your overall database I/O activity that is VSAM. OSAM databases are not eligible for Hiperspace, so only the percentage of your total read I/O that is VSAM can benefit.

Hiperspace operates on 4 096 blocks so only buffer sizes of 4 KB or larger can use Hiperspace.

- ▶ Hiperspace benefits read access only; it offers no benefit to database updates.
- ▶ Adequate available memory is important. If you plan to specify a large number of Hiperspace buffers, be sure you have enough storage to contain them. Hiperspace is never migrated to “auxiliary” storage. Hiperspace does not have to be backed up, because, if the system becomes constrained and starts stealing storage frames from

Hiperspace (possibly causing a read or write to fail), then IMS can always revert to using DASD to retrieve the lost data.

► VSAM database candidates for Hiperspace usage include these:

- Small, heavily referenced databases.

Less Hiperspace is required to cache a larger percentage of a small database.

- Databases with skewed access patterns.

A database that has a small percentage of the total amount of data and is frequently accessed is a good candidate.

- High read:write ratio databases.

Hiperspace benefits read access; therefore, databases that have less update activity have a greater potential for I/O reduction.

- Large index data sets.

Index data sets are typically very good candidates, because they are relatively small and are frequently accessed.

To tune the VSAM buffer pool, you need to monitor it. A good tool for that is the IMS Monitor. It is part of the base product of IMS and is turned on by issuing this command:

```
/TRACE SET ON MONITOR ALL
```

To turn the monitor off, issue this command:

```
/TRACE SET OFF MONITOR
```

We recommend that you run the monitor twice a day, during the two heaviest load times of the day. Example 5-19 is one of the VSAM buffer pool reports. There is one report for each pool.

Example 5-19 VSAM buffer pool statistics

IMS MONITOR	*** BUFFER POOL STATISTICS ***	TRACE START 2006 258, 13:43:12
VSAM BUFFER POOL		
	FIX INDEX/BLOCK/DATA	N/N/Y
	SHARED RESOURCE POOL ID	SZ
	SHARED RESOURCE POOL TYPE	D
	SUBPOOL ID	1
	SUBPOOL BUFFER SIZE	24576
	NUMBER HIPERSPACE BUFFERS	19000
	TOTAL BUFFERS IN SUBPOOL	512
	DIFFERENCE	
1	NUMBER OF RETRIEVE BY RBA CALLS RECEIVED BY BUF HNDLR	0
2	NUMBER OF RETRIEVE BY KEY CALLS	1898
3	NUMBER OF LOGICAL RECORDS INSERTED INTO ESDS	0
4	NUMBER OF LOGICAL RECORDS INSERTED INTO KSDS	4524
5	NUMBER OF LOGICAL RECORDS ALTERED IN THIS SUBPOOL	9489
6	NUMBER OF TIMES BACKGROUND WRITE FUNCTION INVOKED	0
7	NUMBER OF SYNCHRONIZATION CALLS RECEIVED	3843
8	NUMBER OF WRITE ERROR BUFFERS CURRENTLY IN THE SUBPOOL	0
9	LARGEST NUMBER OF WRITE ERRORS IN THE SUBPOOL	0
10	NUMBER OF VSAM GET CALLS ISSUED	3264

11	NUMBER OF VSAM SCHBFR CALLS ISSUED	0
12	NUMBER OF TIMES CTRL INTERVAL REQUESTED ALREADY IN POOL	112995
13	NUMBER OF CTRL INTERVALS READ FROM EXTERNAL STORAGE	12163
14	NUMBER OF VSAM WRITES INITIATED BY IMS/ESA	16210
15	NUMBER OF VSAM WRITES TO MAKE SPACE IN THE POOL	0
16	NUMBER OF VSAM READS FROM HIPERSPACE BUFFERS	2367
17	NUMBER OF VSAM WRITES TO HIPERSPACE BUFFERS	2884
18	NUMBER OF FAILED VSAM READS FROM HIPERSPACE BUFFERS	0
19	NUMBER OF FAILED VSAM WRITES TO HIPERSPACE BUFFERS	113

QUOTIENT : $\frac{\text{TOTAL_NUMBER_OF_VSAM_READS} + \text{VSAM_WRITES}}{\text{TOTAL_NUMBER_OF_TRANSACTIONS}} = 2.14$

Report analysis

Example 5-19 on page 119 is a VSAM Buffer Pool report for a subpool with 512 buffers, each 24 KB in size. The numbers to the left of the line items in the report are for your reference only and do not appear on the actual report. The line descriptions are:

► **Line 3: NUMBER OF LOGICAL RECORDS INSERTED INTO ESDS**

Counts in this field indicate the IMS-maintained logical end-of-file (EOF) marker has been moved to extend one or more ESDSs. In VSAM terms, the EOF is moved one CA at a time.

These counts indicate that one or more data sets are extended to satisfy requests for space as a result of inserts or variable length replaces where the replaced segment had to be split. One or more databases are likely to require reorganization to regain free space. Database DBDs might need to be changed to increase free space specifications.

► **Line 6: NUMBER OF TIMES BACKGROUND WRITE FUNCTION INVOKED**

Counts in this field indicate that background write is active. If the count field is zero, background write might or might not be turned off. If the count field is zero and altered buffer steals are occurring (see Line 15 of Example 5-19 on page 119), it is likely that background write is turned off.

Background write is activated by control card input (//DFSVSAMP DD statement in batch or DFSVSMxx member for online systems). By default, background write is active. If turned off, we recommend you turn it on (see 5.20.2, “VSAM background write” on page 118).

► **Line 7: NUMBER OF SYNCHRONIZATION CALLS RECEIVED**

This count is incremented for every transaction sync point and every BMP CHKP and SYNC call. Only when every transaction updates buffers in a subpool does the count match the number of buffer purge calls.

► **Line 11: NUMBER OF VSAM SCHBFR CALLS ISSUED**

SCHBFR is an IMS macro that might be translated as a search buffer. A large number indicates the subpool is frequently being scanned in a search for space. If this is a batch monitor, the VSAM statistics report identifies which database data sets are incurring the SCHBFR calls and, thus, are good candidates for reorganization or re-specification of free space.

Note: An ISRT call typically causes multiple SCHBFR requests to be issued. Even when SCHBFR finds a buffer in the subpool, IMS examines it to see if there is sufficient room for the segment. If not, another SCHBFR is issued to continue the scan of the subpool.

► Line 12: NUMBER OF TIMES CTRL INTERVAL REQUESTED ALREADY IN POOL

This count indicates the number of times a request for a CI was satisfied from the subpool and therefore a read operation was not required. It is used to calculate a hit ratio for the subpool (discussed under 5.20.4, “VSAM statistics” on page 121 at the end of this list).

► Line 13: NUMBER OF CTRL INTERVALS READ FROM EXTERNAL STORAGE

The count reports the number of requests that could not be satisfied from the contents of the subpool and had to be read in from a database data set on DASD.

Use tuning tactics to attempt to eliminate reads or, if they cannot be avoided, to perform the reads in as efficient a manner as possible. This line item is used to calculate a hit ratio for the subpool (discussed in 5.20.4, “VSAM statistics” on page 121).

► Line 14: NUMBER OF VSAM WRITES INITIATED BY IMS/ESA

The count reports the number of CIs written to database data sets as a result of application program sync points (including CHKP and SYNC calls) and background write.

► Line 15: NUMBER OF VSAM WRITES TO MAKE SPACE IN THE POOL

The count reflects the number of CIs written to database data sets as a result of altered buffer steals.

► Lines 16 and 17: NUMBER OF VSAM READS FROM HIPERSPACE BUFFERS and NUMBER OF VSAM WRITES TO HIPERSPACE BUFFERS

The reads (Line 16) reflect a request of the buffer handler that was satisfied from Hiperspace. The writes (Line 17) reflect the movement of CIs from the virtual storage subpool buffers to Hiperspace. Every read from DASD or Hiperspace into the virtual storage subpool results in a write to Hiperspace.

The ratio of writes and reads to and from Hiperspace is an indication of its effective use. Reasons for using Hiperspace rather than virtual storage buffers include these:

- A VSAM subpool in virtual storage is limited to 32 KB buffers. But there is no practical limit to the number of Hiperspace buffers that can be used.
- From a buffer handler point of view, SCHBFR (NUMBER OF VSAM SCHBFR CALLS ISSUED, Line 11) requests consume CPU resource; the more buffers in a subpool, the longer the path length. SCHBFR requests do not search Hiperspace buffers. Therefore, if SCHBFR requests are perceived to be a problem, placing most of the buffers in Hiperspace reduces the path length required to satisfy the requests.

For HIDAM, a better solution is to provide enough free space so that the most desirable and second-most desirable CIs have enough space to satisfy requests and then to schedule timely database reorganizations to buy back that free space when required.

VSAM reads from Hiperspace buffers are used to calculate a hit ratio for the subpool (discussed in 5.20.4, “VSAM statistics” on page 121).

► Lines 18 and 19: NUMBER OF FAILED VSAM READS FROM HIPERSPACE BUFFERS and NUMBER OF FAILED VSAM WRITES TO HIPERSPACE BUFFERS

Counts in either of these two fields indicate that storage is overextended. Hiperspace, as implemented by VSAM, does not guarantee that Hiperspace buffers are not lost.

5.20.4 VSAM statistics

Several statistics are of use when analyzing a VSAM database subpool. The following are some of the more common statistics used:

► **Hit Ratio**

A subpool hit ratio is calculated as follows:

$$\text{Hit Ratio} = \frac{\text{Line 12} + 16}{\text{Line 12} + \text{Line 13} + \text{Line 16}} \times 100 = \text{xx.x\%}$$

For this example, the hit ratio calculation is as follows:

$$\text{Hit Ratio} = \frac{112,995 + 2,367}{112,995 + 12,163 + 2,367} \times 100 = 90.5\%$$

By itself, a hit ratio means nothing. Hit ratios are generally useful when making run-to-run comparisons. If a change is made to a particular subpool and the hit ratio increases, it is highly likely that the increased hit ratio represents an improvement. This is a good thing, because as you tune the buffers, there should be an increase in the percentage found in the pool. That is, the hit ratio should go up.

► **Reads per second and writes per second**

Reads per second are simply “NUMBER OF CTRL INTERVALS READ FROM EXTERNAL STORAGE” (Line 13) divided by the monitor interval in seconds. The tuning goal is to reduce this number.

Writes per second are “NUMBER OF VSAM WRITES INITIATED BY IMS/ESA” (Line 14) plus “NUMBER OF VSAM WRITES TO MAKE SPACE IN THE POOL” (Line 15), all divided by the monitor interval in seconds. The tuning goal is to reduce this number.

► **Buffer life**

This statistic is useful for online systems. It is calculated as follows:

$$\text{Buffer Life} = \frac{\text{No. of buffers in subpool} \times \text{Monitor Interval}}{\text{NUMBER OF CTRL INTERVALS READ FROM EXTERNAL STORAGE (Line 13)}}$$

Buffer life is the average life in seconds of the contents of a buffer in a subpool. When the average buffer life is in tens of seconds, then the subpool is holding buffer contents across a user's think time. When buffer life is low, such as less than 1 buffer-second per read, it is generally beneficial to increase the number of buffers in a subpool dramatically.

► **Buffer flushes/washes**

A subpool buffer flushes/washes ratio is calculated as follows:

$$\text{Buffer flushes/washes} = \frac{\text{Line 15}}{\text{Line 14} + \text{Line 15}} \times 100 = \text{xx.x\%}$$

For this example, the flushes/washes ratio calculation is as follows:

$$\text{Buffer flushes/washes} = \frac{0}{16,210 + 0} \times 100 = 0\%$$

As you tune the buffers, there should be a decrease in the percentage of buffer flushes/washes.

► **Total I/O**

Total I/O for this buffer subpool is the sum of Line 13 + Line 14 + Line 15.
12,163 + 16,210 + 0 = 28,373

As you tune the buffers, the difference on the following lines should be going up:

- NUMBER OF TIMES CTRL INTERVAL REQUESTED ALREADY IN POOL (Line 12)
- NUMBER OF VSAM WRITES INITIATED BY IMS/ESA (Line 14)

As you tune the buffers, the difference on the following lines should be going down:

- ▶ NUMBER OF CTRL INTERVALS READ FROM EXTERNAL STORAGE (Line 13)
- ▶ NUMBER OF VSAM WRITES TO MAKE SPACE IN THE POOL (Line 15)

For VSAM tuning general rules, see Appendix A.7, “VSAM tuning general rules” on page 229.

5.20.5 Tuning VSAM data sets

VSAM data sets use a *Control Interval (CI)* for the basic unit of storage. A CI is formed by physical records, usually just one. A CI is a contiguous area of storage that VSAM uses to store data records and control information that describes the records. A CI is the unit of information that VSAM transfers between the storage device and the processor during one I/O operation. CIs are contained in a *control area (CA)*. A CA is formed by two or more CIs put together into fixed-length contiguous areas of storage. A VSAM data set is composed of one or more CAs.

The control area is determined by the allocation for the data CONTROLINTERVALSIZE. VSAM takes the smaller allocation size between the primary and the secondary units as the CA size if the allocation is in TRACKS. Otherwise, the CA size is one cylinder regardless of the allocation.

CI and CA splits occur as a result of data record insertions. If a record is to be inserted (in key sequence) and there is insufficient free space in the CI, the CI is split. Approximately half of the records in the CI are transferred to a free CI provided in the CA, and the record to be inserted is placed in the original CI. If there are no free CIs in the CA and a record is to be inserted, a CA split occurs. Half of the CIs are sent to the first available CA at end of the data component. This movement creates free CIs in the original CA, then the record to be inserted causes a CI split.

CI splits are a part of the cost of doing business with VSAM. The index component of a KSDS must be able to index all of the data CIs in the data component of the KSDS. To determine the number of CIs that must be indexed, take the CA size and multiply it by the number of tracks per cylinder, then multiply that number by the number of CIs per track. This yields the number of CIs that must be indexed per CA. If you do not have a large enough index CI size, then all of the data CIs are not addressable, and premature CA splits occur. CA splits are expensive, because of the number of physical I/Os that are required to accomplish the split. We recommend that there is enough free space in the KSDS data set to prevent CA splits between reorganizations.

For more information about index CI sizes, read 5.5.1, “Index CI sizes and record sizes” on page 66.

For VSAM data set tuning general rules, see Appendix A.7.1, “VSAM data set tuning general guidelines” on page 230.

For ESDS performance guidelines, see Appendix A.7.2, “ESDS performance guidelines” on page 230.

For KSDS performance guidelines, see Appendix A.7.3, “KSDS performance guidelines” on page 231.

5.21 Improve GSAM performance

There are three ways to improve the throughput of GSAM databases: using PROCOPT=LS or GS and no DCB buffer information; coding the use of chained scheduling; or, using a BUFNO greater than one.

If you are using a GSAM/BSAM database and not doing “random” reads (that is, not using RSAs), then we recommend using a BUFNO greater than one.

If you code DCB=OPTCD=C, chained scheduling, or PROCOPT=GS or PROCOPT=LS, then multiple buffers are used and IMS determines the number as follows:

- ▶ If TAPE, the number is a minimum of 3 but the greatest of 3, 5, 7, or 9 as long as the total number of bytes is fewer than 64 000.
- ▶ If DASD, the number is the number of blocks on 2 tracks, plus 1 buffer, but with a maximum of 10 buffers.

If none of the above, one buffer is used.

It is always advisable to specify a BUFNO parameter greater than 1. This causes GSAM to use its multiple buffering I/O facility (BUFFIO). This is even better than chained scheduling. It is a “QSAM-like” facility for use with BSAM. BUFNO > 1 causes IMS to use multiple buffers for reads and writes. It also invokes a “read-ahead” function. If BUFNO=1, this function is not used, even if PROCOPT=GS or LS is specified. If BUFNO is greater than one, it is used, even with PROCOPT=G or PROCOPT=L. If BUFNO is not specified, the PROCOPT specification determines if it is used.

5.22 When to reorganize

You should reorganize your database in the following circumstances:

- ▶ Database performance has deteriorated. This can happen either because segments in a database record are stored across too many blocks or CIs, or because you are running out of free space in your database.
- ▶ There are too many physical I/Os to DASD.
- ▶ The database structure has changed. For example, you should reorganize a HALDB partition after changing its boundaries or high key.
- ▶ The (P)HDAM randomizer has changed.
- ▶ The HALDB Partitions Selection exit routine has changed.
- ▶ When the OSAM or VSAM data set goes into extents.
- ▶ When the data portion of a VSAM data set High-Used RBA keeps increasing.
- ▶ When the index portion of a VSAM data set keeps having CI and CA splits.
- ▶ When you start to run out of free space in the database.
- ▶ When roots start not to randomize to the home block in a (P)HDAM database, and start to go to the beyond area or to overflow.



Transaction manager performance

In this chapter, we:

- ▶ Examine the process from scheduling to first IMS call
- ▶ Consider the effect of certain program load and IMS transaction macro parameters
- ▶ Review IMS variable pool parameters on performance

6.1 Scheduling to first IMS call

Before deciding what performance data to gather, or what parameter to adjust, consider what tasks are actually performed in scheduling your work:

1. Input message processing

The IMS control of the transaction begins with the transaction:

- Placed by VTAM in an IMS receive-any (RECANY) buffer
- Moved to a buffer acquired in the High I/O Pool (HIOP)
- Edited by MFS routines, IMS basic edit, or intersystem communication (ISC) edit (depending on terminal type and bypass MFS option, as appropriate)
- Allocated a position on one of the message queue data sets
- Moved to the message queue pool and enqueued on the scheduler message block (SMB)

The time that the message spends in these pools, in MFS processing, and being moved to the message queue buffers affects response time. Individual transaction I/O to the format library affects the message queue. A major factor in determining response time is whether the respective pools are large enough for the current volume of transactions flowing into input queuing. In particular, if the message queue pool is too small, overflow to the message queue data sets occurs.

2. Message classification

This is the call to the z/OS WLM to obtain a WLM service classification for the incoming message.

3. Input queuing

This is the time spent on the input queue or in the message queue buffers waiting for a message region to become available. In a busy system, this time can become a major portion of the response time. The pattern of programs scheduled into available regions and the region occupancy percentage are important and should be closely monitored.

4. Scheduling

Because of class scheduling, regions can be idle while transactions are still on the queue. The effects of scheduling parameters can be:

- Termination of scheduling as a result of PSB conflict or message class priorities
- Termination of scheduling as a result of intent conflict
- Extension of scheduling by I/Os to IMS.ACBLIB for intent lists, PSBs, or DMBs
- Pool space failures in either the PSB or DMB pools

5. Init PB call (activate the WLM delay monitoring environment)

Activate the WLM delay monitoring environment for the message when it is placed into the dependent region. The WLM PB is initialized with the Service Classification and transaction name, message arrival time, program execution start time (current time), user ID, and so forth.

6. Program load and initialization

After scheduling, several processing events occur before the application program can start:

- Content supervision for the dependent region
- Location of program libraries and directories to them
- Program fetch from the program library
- Program initialization up to the time of the first DL/I call to the message queue

For monitoring, you can obtain the overall time for the above activities. The number of I/Os should be checked periodically.

7. Message queue GU

This is the GU call to the message queue. It is chosen as a measuring point because the event is recorded on the system log and is used as a starting point for iterations of processing when more than one message is serviced at a single scheduling of the program. Program control can now be considered effectively passed from IMS to the application.

6.2 Program load options

The following load options can affect performance:

► COBOL options:

- LIBKEEP (causes the run-time library routines to remain in memory)
- RESIDENT (requests library routines be located dynamically at run time, not link-edited)
- RENT (program is re-entrant)
- NODYNAM (no dynamic calls)

- DBLDL is a parameter (default 20, maximum 9999) in the EXEC statement of the DFSMPR procedure that maintains a list of BLDL entries in the dependent region containing the directory index for programs. It is maintained on a most active, most recently used basis, and reduces the I/O to the program directory. Programs with entries in this list have a lower schedule to first DL/I call elapsed time than infrequently used programs.

For a region that is used to test new or changed programs, set the DBLDL parameter to 0, ensuring that the most current version of the program is loaded for each execution. Specifying the DOPT parameter disables quick reschedule.

- LLA is the recommended method for managing dynamically loaded program libraries. For IMS application programs, it saves load modules in a data space, so that the modules can be retrieved more efficiently than through DASD. LLA monitors the frequency with which modules are accessed and readjusts its data space population to optimize the probability of having a module in the data space when it is requested.
- LRR (Library Routine Retention) is an LE function that provides a performance improvement for those applications or subsystems with the following attributes:
- The application or subsystem invokes programs that require Language Environment®.
 - The application or subsystem is not Language Environment-conforming. That is, Language Environment is not already initialized when the application or subsystem invokes programs that require Language Environment.
 - The application or subsystem, while running under the same task, repeatedly invokes programs that require Language Environment.
 - The application or subsystem is not using Language Environment preinitialization services.

Restrictions:

Note these restrictions:

- Language Environment library routine retention is not supported to run on CICS.
- Language Environment library routine retention is not supported to run in an XPLINK (Extra Performance Linkage) environment.
- Run-Time Library Services (RTLS) cannot be used with library routine retention.

- Preload is commonly used for high activity programs that do not use a large amount of virtual storage. You specify the names of the modules to be loaded in member DFSMPLxx of IMS.PROCLIB. When the message processing region is initiated, you specify the suffix xx as a parameter in the EXEC statement of the DFSMPR procedure. Preloaded application programs are then branched to directly rather than through a FETCH program, so preloaded programs should have a schedule to first DL/I call elapsed time that is less than those that use the FETCH program. However, page fault serialization could cause some application program elapsed time to increase.

Note: If the PSB name and the application program name are the same, this can trigger abend U0921 or other unpredictable results.

Restriction: Do not preload application programs for batch regions executed using either the DLIBATCH or DBBBATCH procedures. In these environments, preloading application programs offers no advantage.

Guidelines for the most effective implementation of program preload are:

- All commonly used PL/I, VS Pascal, or COBOL subroutines and application program subroutines should be preloaded into each dependent region. If these subroutines are reentrant, they should be put into the pageable link pack area (PLPA) so that only one copy resides in real storage. Match subroutine usage with the region preload lists to ensure that the appropriate modules are preloaded.
 - Application programs are candidates for preload when they account for a high percentage of a region's transaction volume. Preload is most effective when the transaction arrival rate for the preloaded program is adequate to keep the working set of the program in real storage. If a system is constrained by real initializing z/OS and IMS parameters for tuning storage contention, preload only subroutines and very high volume application programs, because preloading can increase the paging rate. In addition to deciding to preload them, consider class scheduling of high volume transactions.
 - Depending on the transaction arrival rate, it can be advantageous to preload in only one or two regions and class schedule the transactions accordingly. The use of the preload option for system performance improvements is highly dependent on the availability of real storage and the arrival rate of the candidate transactions.
 - Application program overlay is sometimes a viable alternative to preload. If an all-purpose application program is coded to interrogate the input transaction data and execute only a portion of the application program code for the transaction subcode, program overlay I/O might be more efficient than loading or paging through the entire program. Preload is a better alternative if the transaction arrival rate is sufficient to maintain a working set in main storage.
- Program fetch:
 - Order PGMLIB in descending frequency of use.

- Use full-track blocking, because this minimizes Start I/O (SIO) operations and seek times.

Note that Virtual fetch is no longer supported by the operating system.

- ▶ SRCH specifies whether the IMS control region searches the JPA and the LPA before the STEPLIB or JOBLIB when loading a module. If the SRCH parameter is set to 1, then IMS searches the JPA and the LPA. If it is set to 0, then IMS does a standard search.

If multiple IMS systems or IMS DL/I batch jobs execute concurrently in the same z/OS LPAR, some virtual storage can be saved by referencing certain modules in LPA, so that they are shared by all the jobs rather than each job having its own copy. This is feasible only if all the IMS systems are at the same release and level.

If IMS modules are moved into LPA and the control region has a JOBLIB or STEPLIB DD statement, we recommend that you set the SRCH parameter to 1. This can result in two benefits: a save in storage and a save in I/Os and CPU time, because the modules do not need to be loaded.

6.3 Transaction macro parameter options

The following transaction macro parameter options can affect performance:

- ▶ MAXRGN is a keyword on the TRANSACT macro and specifies the number of MPP regions that can be scheduled for this transaction code. Use it when you want to prevent one or more transactions from monopolizing all available regions.
- ▶ SEGNO is a keyword defined on the TRANSACT macro and sets the maximum number of output message segments allowed for each input message processed by the scheduled program. Use it to protect available message queue space from being used up by a program output loop.
- ▶ PARLIM is a keyword defined on the TRANSACT macro and specifies the number of messages that should be enqueued before another region is scheduled. This value is multiplied by the number of regions already scheduled for this transaction. If the result is fewer than the number of messages enqueued, another region is scheduled for the transaction, unless MAXRGN is exceeded. If the region cannot be scheduled for internal reasons (database intent), the next transaction within the class is scheduled.

Note: The text above is valid for a unshared queues environment. In a shared queues environment, the successful consecutive GU count is used instead of the enqueue count. Refer to 11.9.7, “FF scheduling differences” on page 209 for more information.

- ▶ PROCLIM is a keyword defined on the TRANSACT macro and specifies the number of messages that an application program can process in a single schedule. For message-driven programs, PROCLIM is the limit for one transaction and uses the real elapsed time, typically used because the terminal is set in response mode by the transaction. The count parameter is ignored.
- ▶ PRTY is a keyword defined on the TRANSACT macro and has three values as follows:
 - The first value (NORMAL) specifies the priority assigned to this transaction when the number of input transactions enqueued and waiting to be processed is less than the limit count value. The valid specification range is from 0 through 14 (the default is 1).
 - The second value (LIMIT) specifies the priority to which this transaction is raised when the number of input transactions enqueued and waiting to be processed is equal to or greater than the limit count value. The valid specification range is from 0 through 14 (the default is 1).

- The third value (LIMIT COUNT) specifies the number that, when compared to the number of input transactions queued and waiting to be processed, determines whether the normal or limit priority value is assigned to this transaction. The limit count value can range from 1 through 65 535. The default is 65 535.

When the limit priority is used, and the priority is raised to the specified limit priority value, the priority is not reduced to the normal priority until all messages enqueued for this transaction code are processed. If you do not want the limit priority for this transaction, code equal values for the normal and limit priorities, and a limit count of 65 535.

- SCHDTYP is a keyword defined on the APPLCTN macro, and describes programs that operate in message processing regions, Fast Path message-driven program regions, batch message processing regions, CCTL threads, or batch processing regions.

SCHDTYP can specify PARALLEL (this application program can be scheduled into more than one message region or batch message region simultaneously) or SERIAL (default).

Given that altering SCHDTYP from SERIAL to PARALLEL (or reverse) requires a MODBLKS system generation, consider defining all resources as SCHDTYP=PARALLEL. You can then use MAXRGN to dynamically control parallel processing, where MAXRGN=1 for SERIAL processing required, or MAXRGN=x (where x is greater than one) for PARALLEL processing, for greater workload flexibility, as required.

- WFI is a keyword on the TRANSACT macro and specifies that IMS allows the program to remain in main storage after it has processed the available input messages.

The pseudo WFI (pseudo wait-for-input) option allows an MPP region to remain scheduled until another input message appears. With pseudo WFI, unnecessary application program termination and rescheduling can be eliminated:

- PWFI is usually more appropriate (it is a region parameter, WFI is a TRANSACT option).
- Consider IMS class scheduling (but do not fall into the “single” class trap).
- Consider isolating high volume transactions to their own classes where possible (if not practical, then use PARLIM/MAXRGN to minimize “thrashing”).

6.4 IMS parameters

This section describes some of the IMS execution parameters that have an effect on IMS performance.

Note that during IMS initialization, the DFS1929i message displays the system parameters that are actually in effect. The values are taken either from the IMS system generation, or from the DFSPBxxx member or from the PARM specification in the JCL EXEC statement, but they are the values that are actually used and can be useful information for problem determination.

6.4.1 ARC parameter

ARC parameter specifies the number of online log data sets (OLDS) that must be filled and closed before IMS starts automatic archiving. Set this parameter to the default value (01).

6.4.2 BSIZ parameter

BSIZ parameter should be set properly to avoid unnecessary ECSA usage. Although the BSIZ must be at least as large as the largest defined DEDB CI size, it should not be oversized.

6.4.3 CPLOG parameter

CPLOG parameter specifies the IMS system checkpoint frequency. The value is the number of the IMS log records written between system-generated checkpoints. The valid values are from 1 000 to 16 000 000 (the default value is 500 000). Set the parameter to CPLOG=500K (the default), then check the IMS system checkpoint frequently and compare it to the average time that your system takes to fill up an OLDS, so that at least one system checkpoint is taken on each OLDS.

If needed, change the CPLOG value dynamically using the **/CHANGE CPLOG** command and modify the CPLOG parameter in the DFSPBxxx member of IMS.PROCLIB with the new value.

6.4.4 DBBF parameter

The DBBF parameter on the DFSPBxx member of IMS.PROCLIB specifies the maximum number of buffers in ECSA for Fast Path databases. If DBBF is not specified, then the BFALLOC parameter of FPCTRL macro in IMS system generation is taken; if BFALLOC is not specified either, the default IMS value is 10. The maximum value is 65535.

Within the DBCTL system, consider what the proper size is for the Fast Path DEDB buffer pool, because it affects the system ECSA setting and you do not expect your system has an ECSA shortage due to too big buffer pool allocation.

With IMS Version 8 and above, FPBUFF=LOCAL is introduced, which enhances the FDBR initialization processing to GETMAIN the DEDB buffer pools in the FDBR private region rather than ECSA. This relieves some ECSA constraint used by FDBR.

The following recommended formula was introduced in *IMS V7 Performance Monitoring And Tuning Update*, SG24-6404:

DBBF = Number of open areas that have SDEP segments
+ Sum of NBA for all concurrently active FP programs
+ Largest OBA allocation for any of the concurrently active FP programs
(including any specified by CICS for DBCTL)
+ DBFX
+ Sum of all Fast Path buffers used by CICS(CNBA)

6.4.5 DBFP parameter

DBFP=0 means page fix and free DBBF buffers for each schedule and termination, DBFP=1 means page fix when allocated and used, so the number only increases. DBFP=*n* (2 to 3 600) means the same as 1 but every *n* seconds page free buffers that are not needed by currently scheduled applications.

If many IMS dependent regions are started and terminated with NBA and OBA specified in a short time period, which would be the case with many BMPs, use DBFP=1. This helps reduce the number of page fix and page free operations performed by z/OS.

When you use DBFP=1, the unfixed value on /DIS POOL FPDB drops to a small value, which might not meet NBA for a new region, but it does not mean that new region cannot be started. For example:

AVAIL = 11982 WRITING = 4 PGMUSE = 0 UNFIXED = 14

6.4.6 DBFX parameter

The DBFX parameter (default is 4) specifies an additional system buffer allocation, which is page fixed at the start of the first Fast Path region. The DBFX allocation is needed because DEDB writes are deferred until after sync point processing, to allow for asynchronous processing where the DEDB updates are held until the associated log buffer is written.

DBFX specifies the number of buffers that are page fixed at control region initialization time. That number does not change even if you use DBFP=1. If your DBFX value is small, you could run into dependent regions having to wait for buffers in the output thread.

Actually, there is no generally recommended value you can use to determine if the DBFX setting is correct. But /DIS POOL FPDB shows the active counts (WRITING). You also might want to check the buffer waits in the Fast Path Resource Usage and Contention section of the Performance Analysis report. If the buffer wait numbers are large or keep increasing, you might consider increasing DBFX.

Example 6-1 shows a /DIS POOL FPDB command output.

Example 6-1 DIS POOL command output

```
?DIS POOL FPDB
DFS4445I CMD FROM MCS/E-MCS CONSOLE USERID=JOUK03: DIS POOL FPDB IM1B
DFS4444I DISPLAY FROM ID=IM1B 987
FPDB BUFFER POOL:
  AVAIL = 172 WRITING = 1 PGMUSE = 0 UNFIXED = 427
POOLNAME CSIZE PBUF SBUF MAX CURRENT LK HITS VALID
ITEMPOOL 04096 00200 00100 00500 00500 Y 052% 099%
DISTPOOL 01024 00100 00025 00300 00125 Y 061% 100%
WAREPOOL 00512 00020 00005 00030 00020 Y 099% 100%
*2006248/170713*
```

Example 6-2 shows the IMS Performance Analyzer Fast Path Resource Usage and Contention report.

Example 6-2 Fast Path Resource Usage and Contention report

IMS Performance Analyzer										Page 1															
Fast Path Resource Usage and Contention - IM1B																									
		From 05Sep2006 15.01.34.20								To 05Sep2006 15.02.52.91								Elapsed= 0 Hrs 1 Mins 18.717.029 Secs							
Transact Code	Routing Code	Count	---DEDB Calls---				---ADS I/O ---				--VSO Activity--				-Common Buffer-				Contentions		LGNR Stat	Totl Tran			
			Reads	Updates	Reads	Updates	Reads	Updates	Reads	Updates	Reads	Updates	Reads	Updates	Usage	Wts	Stl	Tot Tot CI/	Total #CI	Sync			Rate		
			Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max			UOW	OBA	Sec	Comb	Logd	Fail	/Sec
TPCCN	*SF=L	29	20	31	18	29	10	15	0	0	5	9	0	0	22	32	0	0	0	0	1	0	0	29	0
	*Unknown	1	28	28	27	27	14	14	14	14	9	9	1	1	29	29	0	0	0	0	0	0	0	0	0
	TPCCN	3011	22	32	21	31	11	20	11	19	5	12	1	1	23	36	0	0	0	0	121	0	0	0	38
TPCCN	*Total*	3041	22	32	21	31	11	20	11	19	5	12	1	1	23	36	0	0	0	0	123	0	0	29	39
TPCCO	*MPP	233	10	15	0	0	1	3	0	0	0	0	0	0	1	3	0	0	0	0	1	0	0	0	3
TPCCP	TPCCP	2825	2	2	3	3	1	2	1	1	0	1	2	2	4	4	0	0	0	0	2	0	0	0	36
TPCCS	*MPP	247	418	477	0	0	229	304	0	0	0	1	0	0	10	10	0	999	0	0	162	0	0	0	3
System Totals		6346	28	477	11	31	15	304	6	19	3	12	1	2	13	36	0	999	0	0	288	0	0	29	81

6.4.7 Dynamic pools parameters

IMS defines the dynamic pools (the default limit per pool is 2 GB -1) as follows:

- ▶ AOIP: Automated operator (Type II) pool
- ▶ CESS: Communication external subsystem pool
- ▶ CIOP: Communication I/O pool
- ▶ EMHB: Expedited Message Handler buffer pool
- ▶ FPWP: Fast Path work pool
- ▶ HIOP: High I/O pool
- ▶ LUMP: LU 6.2 Manager private buffer pool
- ▶ LUMC: LU 6.2 Manager common buffer pool
- ▶ QBUF: Message queue buffer pool (in a shared queues environment)

The dynamic pool manager (DFSPOOL) manages storage by:

- ▶ Dynamically changing the size of each pool according to the system demands
- ▶ Each pool consisting of multiple blocks of fixed-length buffers
- ▶ Using fixed-length buffers to satisfy variable length requests
- ▶ Using bitmaps to manage storage

Each fixed pool consists of zero or more noncontiguous blocks of storage anchored off a pool header. Each block is divided into a number of fixed-length buffers, the size and the number of buffers that each block contains can vary from block to block within a pool. By obtaining new blocks and releasing unused blocks, a pool can be expanded and contracted as needed during the execution of IMS.

The 2 GB -1 default limit can be overridden by the DFSSPMxx member of IMS.PROCLIB (parameters AOIP, CIOP, EMHB, FPWP, HIOP, LUMC, and LUMP), but not the upper expansion limit of CESS pool, through:

```
FPL=poolname,(size,pbuf,sbuf,init),(size,pbuf,sbuf,init),(...)
```

Where the fields have the following explanations:

- ▶ size is the buffer size in bytes.
- ▶ pbuf is the primary allocation in buffers per block.
- ▶ sbuf is the secondary allocation in buffers per block.
- ▶ init can be Y or N and specifies whether to get the primary allocation during system initialization.

For example, to replace buffer definitions for HIOP, you could use:

```
FPL=HIOP,(200,20,30,Y),(300,20,20,N),(400,10,20,N)
FPL=HIOP,(500,10,30,Y),(600,10,10,N),(700,10,20,N)
FPL=HIOP,(800,5,10,Y),(900,5,5,N)
```

Note that this pool definition extends over three lines. No continuation character is used in column 72.

Table 6-1 shows where each of the DFSPOOL managed pools resides.

Table 6-1 DFSPOOL pool locations

Acronym	Location	Usage
AOIP	IMS CTL Extended Private Area	AOI exit messages
FPWP		Fast Path work buffers
HIOP		TP buffers
LUMP		LU 6.2 TP buffers
CIOP	IMS CTL Private Area	TP buffers
CESS	ECSA	External subsystem buffers
EMHB		EMH buffers
LUMC		LU 6.2 miscellaneous work areas

The pool headers for each of the pools reside in ECSA storage. A table is created during initialization that is used during buffer allocation. The table contains 256 two-byte entries. This table also resides in ECSA.

The load module containing the default pool definitions is also loaded into ECSA. Note that 16 KBs of ECSA storage are used for the storage manager trace tables. The dynamic storage management does not allow the CIOP or EMHB pools to be page fixed.

We recommend that you follow the steps on this checklist:

1. Maximize buffers in the primary blocks.
2. The best performance can be achieved from the storage manager by ensuring that all buffers are obtained out of the non-compressible blocks. Therefore, we recommend that you use the initial allocation option (INIT=Y) for persistent storage. Take samples at periods of peak traffic and establish the size according to the statistics.
3. Minimize the number of blocks per buffer set.
4. Avoid the large system effects of scanning long chains of blocks; overallocation can cost cycles and storage. We recommend that you define big blocks.
5. Spread the buffer set; distribution saves virtual storage.
6. A wide distribution of buffer sizes improves the chances of better fits of messages to buffers and thereby avoids wasting virtual storage.
7. Do not overly constrain pools.
8. It is essential that the upper limit for a pool is not too low. If the storage manager makes a request exceeding the limit, results are unpredictable. For the EMHB pool, a message is returned to the terminal; for other pools, an abend of IMS might ensue.
9. Avoid oversized traffic.
10. Use of the oversized buffers is very costly; individual GETMAINS, inefficient matching of large message length to the oversize buffer, and compression overheads can follow.
11. We recommend that you modify the definitions trying to keep the use of oversized buffers at 0 value. This use is shown by the **/DIS POOL** command (the overflow value is not zero).

In summary, we have the following recommendations:

- ▶ Do not exaggerate the fine-tuning; requirements are mostly met by self-adjusting.
- ▶ Sample the data to create more efficient buffer set groups.

- Avoid large system effects and pool fragmentation.

6.4.8 EMHL parameter

The EMHL control region execution parameter specifies the default size for Expedited Message Handler buffers.

6.4.9 EXVR parameter

This parameter can be used to page fix the Queue Manager buffer pools (EXVR=1). However, you should instead page fix those buffer pools either by:

- Utilizing the z/OS storage isolation function
- Using the DFSFIXxx member in PROCLIB, specifying POOLS=QBUF

6.4.10 Hash tables parameters: LHTS, NHTS, and UHTS

IMS uses certain control block pools for CNTs (LTERMs), VTCBs (VTAM nodes), and SPQBs (users). The control blocks in these pools are not in any order; instead, when IMS needs to locate one of these control blocks, it uses a hashing technique (not a sequential search of the pool). Each pool has a hash table, and each hash table contains a user-specified number of slots or anchor points. When a control block is added to the pool, its name is hashed so that a number representing one of the hash table slots is generated. If that slot is not in use, the address of the new control block is put in it. If the anchor point is already in use, a synonym is created. Synonyms are chained together off the anchor point in ascending alphabetical order.

So for performance reasons, the control block synonym chains, on average, should not contain more than seven blocks, to avoid a high cost sequential search of that synonym chain whenever a control block is needed.

To calculate the number of required slots in each hash table (that is, the value of the hash table parameters):

1. Take the maximum number of resources that can ever exist (conversations, LTERMs, VTAM terminals, and users) and divide that number by seven.
2. Round up to the next higher 2 K block.

For example, if you have a network with up to 18 000 ETO terminals expected to be concurrently logged on to IMS, then you need 18 000 VTCBs.

$18000 / 7 = 2571$
next higher 2k block = 4096

In this case, specify NHTS=4096. Set the same value for LHTS (for LTERMs) and also UHTS (for users).

For all four parameters, the valid values are from 0 to 32767. The default value is 256, which is appropriate for up to 1 792 resources (terminals). If you have significantly more than 1 792 terminals, set these parameters accordingly.

6.4.11 Logging parameters

The WADS parameter of the DFSPBxx member of IMS.PROCLIB specifies whether single or dual WADS logging is going to be used.

The use of dual WADS should only be considered for production systems, because dual writes are done to both copies in parallel. So, although no performance penalty is incurred if both copies are on DASD of equivalent performance, dual writes use extra CPU cycles so it is not free. It is highly unlikely that both IMS and WADS DASDs would fail at the same time.

All the WADS must be allocated on the same device type with the same allocation. Allocate WADS on a dedicated device, preferably with good caching function (such as ESS devices).

The LGNR parameter of the DFSPBxx member of IMS.PROCLIB specifies the number of DEDB buffer alterations that is to be held before an entire control interval is logged. Certain events, such as full-function transaction sync point and MSC message logging, demand that log records are written out immediately. These forced log writes are done to the WADS, pending the WADS being written to the OLDS when the log buffer fills up. Thus, the WADS contains the committed log records not yet written to the OLDS, and would be read by the online system in the event of an emergency restart (for closing the OLDS).

The log buffer is considered to be a set of 2 KB pieces. At each forced write, IMS examines the current log buffer, and any 2 KB piece that contains new log data is written to the WADS. IMS selects a WADS track that does not contain previously committed pieces of this buffer and writes the data (key 0) to the first 2 080-byte record on the WADS that comes under the write head (or into the cache, when appropriate).

When the current OLDS buffer eventually fills up, it is scheduled to be written. Once the write completes, the WADS tracks containing that block can be reused. With this technique, IMS actually needs a track for each 2 KB piece of a buffer, plus one. For example, a 26 KB buffer (OLDS BLKSIZE) needs 14 tracks for the WADS. This set of tracks is referred to as a *track group*.

Because BSAM sometimes chains multiple buffer writes together, the WADS must have at least two track groups. This is the minimum requirement but ensure that the size of the WADS data set is defined large enough to have a track group for every log buffer. This is also the maximum number of tracks that are ever going to be used, so there is no reason to overly specify the WADS size.

6.4.12 LSO parameter

This parameter specifies the area of storage in which IMS allocates some control blocks, buffers, and DL/I code. Using the LSO=S (meaning separate address space) option executes in cross-memory mode, frees CSA storage, allows more space for some buffers as they are relocated in ECSA, generally uses significantly less CPU, and experiences less virtual storage constraint than the option LSO=Y.

When LSO=Y (the default), DL/I calls must switch z/OS TCBs (from the dependent region to the control region), so this is a more expensive option. Although, it might also be a good option for a system, which is almost exclusively a Fast Path or IMS test system, if the PSB and DMB pools and associated work pools are small, and the system definition has a large number of transactions and databases defined. In these cases, the amount of CSA storage required for LSO=S might actually exceed the amount of CSA storage required for LSO=Y.

6.4.13 Message format buffer pool parameters

The FBP and FRE system execution parameters can be used to modify the definition of the message format buffer pool (MFBP).

6.4.14 OTHR parameter

This parameter specifies the number of concurrent output threads for the entire Fast Path system. The maximum value is 255 or MAXPST, whichever is lower. OTHR=*n* causes *n* service request blocks (SRBs) and extended SRBs (ESRBs) to be allocated during system initialization. Set the PST parameter to a value that is at least as great as the maximum number of dependent regions that are ever going to concurrently update a Fast Path database.

6.4.15 Parameters for scheduling pools

There are several parameters in the DFSPSBxx member of the IMS.PROCLIB that can be used to specify the size of the scheduling pools. These parameters are:

- ▶ PSB pools: PSB, CSAPSB, and DLIPSB parameters
- ▶ PSB work pool: PSBW parameter
- ▶ EPCB pool: EPCB parameter
- ▶ DMB pool: DMB parameter
- ▶ DMB work pool: DBWP parameter

6.4.16 PI parameters

PIINCR and PIMAX are the parameters in the DFSPBxxx member of IMS.PROCLIB that can have an influence the IMS internal lock manager performance and IMS performance.

6.4.17 PRLD parameter

This parameter specifies the two-character suffix for the DFSMPLxx member in IMS.PROCLIB, which identifies the modules to be preloaded in IMS regions. Any z/OS, IMS or user-written module can be made resident in IMS regions by using the IMS module preload function, which can improve the response time of transactions that frequently refer to the preloaded modules although the better candidates to be preloaded are those modules which are most frequently used and which load is most costly.

6.4.18 PST and MAXPST parameters

The PST parameter is used to specify the number of dependent region partition specification table (PST) control blocks that IMS creates at initialization time. Although if more PSTs are needed because a new dependent region starts, IMS dynamically allocates a new PST for each new dependent region.

The MAXPST parameter (default 255, maximum 999) limits the number of dependent regions that can be concurrently started. Set the PST parameter to a value at least as great as the maximum number of dependent regions that are ever going to be concurrently started in your IMS system. When specifying the MAXPST value, you need to establish a balance between:

- ▶ The number of dependent regions that are going to be necessary during peak time.
- ▶ The consumption of resources, such as PSB pools, database pools, or VSAM strings. If you allow the operators to open any number of regions; these resources could be overcommitted.

MAXPST also limits the value of the OTHR parameter. Do not specify a MAXPST value lower than the number of output threads that are going to be needed.

6.4.19 QBUF parameter

This parameter specifies the number of message queue buffers (minimum 3 and maximum 9999) to be allocated to the queue.

In an unshared queues environment, ideally very few I/Os to the message queue pool data sets should occur. If your system experiences frequent I/Os to the message queue pool data sets, increase the QBUF value.

In a shared queues environment, the number you specify for the QBUF parameter is used as the initial number of message queue buffers allocated and is dynamically expandable.

6.4.20 RECA parameter

This parameter specifies the number of receive-any buffers (1 to 500) and overrides what is specified in the RECANY= parameter on the COMM macro. Monitor the number of RECANY buffers that your system uses during the peak time and specify more buffers than necessary.

6.4.21 RES parameter

The RES parameter specifies whether (default RES=Y) the PSBs and DMBs defined as RESIDENT in the system definition macros should be made resident during system initialization. Reading an intent list from the ACBLIB requires an I/O and using RES=Y avoids ACBLIB I/Os for the intent lists during program scheduling time.

If RES=Y is specified, then during IMS startup:

- ▶ All the intent lists are loaded.
- ▶ All the PSBs and DMBs defined as RESIDENT are loaded and made resident in storage.

Use the default value, RES=Y, in all production systems. In a test environment (or if you have a storage constraint), then you might specify RES=N.

6.4.22 SAV parameter

This parameter specifies the number of dynamic save area sets (maximum 999) and save area prefixes (SAPs) for use by communications ITASKs. When needed, IMS expands the number of SAPs up to 10 times the SAV parameter value. If more SAPs are needed, the communications ITASKs have to wait for SAPs to become available. This can result in communication delays.

6.4.23 SRCH parameter

This parameter specifies whether the IMS control region searches the JPA and the LPA before the STEPLIB or JOBLIB when loading a module. If the SRCH parameter is set to 1, then IMS searches the JPA and the LPA. If it is set to 0, then IMS does a standard search.

If multiple IMS systems or IMS DL/I batch jobs are executing concurrently in the same z/OS LPAR, some virtual storage can be saved by referencing certain modules in LPA, so that they are shared by all the jobs rather than each job having its own copy. This is feasible only if all the IMS systems are at the same release and level.

If IMS modules are moved into LPA and the control region has a JOBLIB or STEPLIB DD statement, set the SRCH parameter to 1. This can result in two benefits: a save in storage and a save in I/Os and CPU time, because the modules do not need to be loaded.

6.4.24 VAUT parameter

Using the default VAUT=0 prevents the use of the VTAM authorized path facility, which requires less CPU for processing SENDs and RECEIVEs, because VTAM performs less validity checking of the VTAM control blocks. Specify VAUT=1 to enable this function.

6.4.25 VSPEC parameter

This parameter specifies which member (DFSVMxx) in IMS.PROCLIB to use to define the database buffer pools for OSAM and VSAM databases.

6.5 Data gathering

Certain performance data should be gathered on a regular basis, to assist with later problem determination.

6.5.1 IMS Monitor

Run the following for the IMS transaction peak load period:

1. Issue IMS simple checkpoint (/CHE) and /DIS POOL ALL commands.
2. Run IMS Monitor, tracing all options, for an approximately 10 minute time period.
3. Immediately after stopping the IMS Monitor, take another IMS simple checkpoint (/CHE).
/CHE
/DIS POOL ALL
/TRA SET ON MON ALL
(trace interval, say 10 minutes)
/TRA SET OFF MON
/CHE
4. Issue another IMS simple checkpoint (/CHE).
5. Run the post processor program (DFSUTR20) and print the DL/I CALL SUMMARY REPORT by specifying DLI in the //ANALYSIS DD * statement.

The distribution report is not required at this time, because it produces a detailed analysis of the distribution of IWAIT times, elapsed times, and so on.

6.5.2 Recording the pool transaction

We recommend the following steps for recording the pool usage:

1. Record pool usage regularly (suggestion: use TCO to schedule the following command every hour):

```
/DIS POOL DMBP PSBP DBWP HIOP FPDB
```

2. Record transaction usage regularly (suggestion: use TCO to schedule the following commands every five minutes):

```
/DIS A  
/DIS Q TRAN
```

3. Record Checkpoint statistics (no serialization, minimal overhead/log data):

```
/CHE STATISTICS (suggestion: use TCO to schedule every 60 minutes)
```

4. RMF and SMF:

Use IFASMFDP to dump RMF record types 70 through 79 as a minimum, archive to GDG.

6.5.3 Gathering the system monitoring data

We recommend the following steps to gather system monitoring data:

1. RMF Monitors I, II, and III can be used for gathering the system monitoring data:
 - ▶ RMF Monitor I session provides data on system-wide resources, such as paging activity, channel activity, device activity, SRM or WLM workload activity, and enqueue contention. The data is used to determine how well the overall system performs. Run the RMF Monitor I reports for 12 to 15 minutes to gather information about:
 - Processor activity
 - Channel activity
 - Device activity - DASD
 - Page/Swap data set activity
 - Enqueue/Dequeue reports - detail
 - Workload activity
 - XCF activity
 - ▶ RMF Monitor II session collects information about each address space in the system, as well as paging, real storage, processor, and SRM or WLM activity. Monitor II is basically a snapshot type of session, because it generates reports from a single sample of system indicators. Run the RMF Monitor II reports for:
 - Address space state data (ASD)
 - Address space resource data (ARD)
 - Address space SRM Data Paging reports (ASRM)
 - Address space resource data by jobparm (ARDJ)
 - Address space State Data (ASDJ)
 - Address space SRM Data (ASRMJ)
 - ▶ Run the COUPLING FACILITY ACTIVITY RMF Monitor III SYSRPTS reports for 10 to 15 minutes.

6.5.4 IMS log data

The IMS log contains other data that can be used in performance evaluation. This data can be printed by using the log formatting print utility (DFSERA10), IMS Performance Analyzer (IMS PA), or a user-written program that selects and formats the records of interest. The other data and associated log records are:

- ▶ X'38' – Offset x'5C' - Lockmax held at high watermark
- ▶ X'4001' – Control region CPU time (field CHKTMCTL)
- ▶ X'4001' – DLISAS address space CPU time (field CHKTIMEU=CTL + DLI elapsed time)
- ▶ X'41' – Lockmax for batch
- ▶ X'45xx' – IMS pool statistics:
 - X'4501' – Dynamic database log statistics
 - X'4502' – Queue buffer statistics
 - X'4503' – Format buffer pool statistics
 - X'4504' – Database buffer pool statistics
 - X'4505' – Variable pool statistics
 - X'4506' – Application scheduling statistics
 - X'4507' – Logger statistics
 - X'4508' – VSAM subpool statistics
 - X'4509' – Program isolation statistics
 - X'450A' – Latch statistics
 - X'450B' – Selected dispatcher statistics

- X'450C' – Storage pool statistics
- X'450D' – RECANY statistics
- X'450E' – Fixed pool statistics
- X'450F' – Dispatcher statistics
- X'4521' – IRLM
- X'4522' – IRLM
- ▶ X'56FA' – Transaction statistics
- ▶ X'67FA' – LOCK trace
- ▶ X'67FA' – Entry type x'CA' - Program Isolation trace
- ▶ X'67FF' – Application program user abends. User abends U777 (deadlock) and U778 (ROLL call), for example, can be selected and reported by analyzing these records.

6.6 Page fixing

Highly used modules probably do not need to be fixed, because they are referenced frequently enough to remain in central storage. However, some IMS and user modules can be fixed to improve performance (to save I/O time and paging overhead) by specifying them in the IEAFIXxx member of SYS1.PARMLIB, so that z/OS places them into FLPA.

Do not specify an excessively large fix list (because an increase in paging and swapping can then be experienced by other system components), so just page fix the resource cleanup modules:

- ▶ DFSMRCL0 (IMS RTM Routine)
- ▶ ASUH2RMT (VSPC RTM Routine if present)
- ▶ ERBMFRES (RMF RTM Routine)

Those modules are loaded and referenced for every task termination in the system, and they can be found faster if they are in the z/OS fix list, which results in saving CPU cycles. Also, page fix the following pool-related storage pools:

- ▶ OSAM database buffers and prefixes:

Using the IOBF control statement in the DFSVSMxx member of IMS.PROCLIB. To page fix OSAM buffers and prefixes for a pool, you must set the fix1 and the fix2 parameters to 'Y':

```
IOBF=(length,number,Y,Y,caching option)
```

With z/Architecture, all OSAM database buffers are page fixed above the 2 GB line.

- ▶ VSAM I/O-related blocks:

Using the VSAMFIX control statement in the DFSVSMxx member of IMS.PROCLIB. To page fix VSAM I/O-related blocks, you must indicate '(IOB)':

```
VSAMFIX=(IOB)
```

- ▶ PSB pool in CSA (when implementing LSO=S option):

Using the POOLS control statement in the DFSFIXxx member of IMS.PROCLIB.

```
POOLS=DLMP
```

- ▶ PSB pool in DL/I SAS (when implementing LSO=S option):

Using the POOLS control statement in the DFSFIXxx member of IMS.PROCLIB.

```
POOLS=DPSB
```

- ▶ Queue manager buffer pool:

Using the POOLS control statement in the DFSFIXxx member of IMS.PROCLIB.
POOLS=QBUF

6.7 WLM

The job dispatching priority dictates in what sequence address spaces are to be dispatched when they have work to do. Any job with a dispatching priority higher than IMS and its dependent regions can cause interference. z/OS tasks, such as the Master Scheduler, JES, VTAM, and GRS, normally run at higher dispatching priorities. RMF and GTF must run at a higher dispatching priority than IMS to obtain the service necessary to collect the correct data. Figure 6-1 shows a recommendation for the dispatching priorities between the different IMS address spaces.

Priority	Address Space
n	IRLM
n - 1	VTAM, GTF, RMF, GRS
n - 2	DBRC, CQS
n - 3	IMS CTL
n - 4	DLS
n - 5 (or less)	IMS MPP1
n - 6 " "	IMS MPP2
n - 7 " "	"
n - 8 " "	IMS MPPn
n - 9 " "	BMP batch

Figure 6-1 Suggested IMS dispatching priorities

Refer to Section 4.2, "CPU management" on page 50 for further details.

6.7.1 IRLM address space

The IMS Resource Lock Manager (IRLM) address space is required for N-way data sharing and is optional for single-system program isolation usage. We recommend using the highest possible dispatching priority of the IRLM address space.

Most use of the IRLM takes place under the dependent region TCBs, using cross-memory services. However, the occasional task that executes within the IRLM address space itself must have top priority within the IMS system (for example, deadlock detection). IRLM must also be higher than any other z/OS client address space, such as VTAM. If this higher dispatching priority is not specified, response times are lengthened.

6.7.2 DBRC address space

Fix the dispatching priority of the DBRC address space at least one level above those of the IMS DLS and control region. IMS uses DBRC to process the database opens, first calls to the

database, EOVS on a database, the database close, and all log data set changes. The higher dispatching priority ensures that DBRC processing occurs when necessary.

6.7.3 CQS address space

If you run WLM in Goal mode, you should have the same service class as IRLM and the IMS control region.

6.7.4 IMS control region

If the IMS schedule NOT-IWAIT time is greater than 4 ms and the total system paging rate is low (fewer than 12 pages per second and a page resolution time of less than 100 ms), higher priority tasks might be executing in the system and using the CPU cycles needed by IMS. Determine whether these higher priority tasks must run at their present dispatching priority. Lowering their DPRTY below that of IMS, if possible, gives the CPU resource back to IMS.

Log Archive (DFSUARCO) jobs should have a dispatching priority that is at least as high as the IMS control region or even higher. It does not hurt the system, because the archive utility consumes very little CPU (it is an I/O bound program), but it provides processor cycles to the utility whenever it needs them.

6.7.5 IMS DLS address space

Fix the dispatching priority of the DLS address space at one level below that of the IMS control region. The reason is that the DLS address space processes the open, close, and allocation of all full-function databases. Fast Path databases are owned by the control region.

Additionally, the scheduling of transactions is performed in the DLS address space if it loads the application PSB.

6.7.6 IMS dependent regions

Specifying a unique dispatching priority ensures that a dependent region is always in the same position on the z/OS dispatcher sequence queue relative to the IMS control regions and dependent regions.

Without unique dispatching priorities, all dependent regions can have the same dispatching priority assigned to them. Therefore, the message regions can be dispatched in different priority sequences each time they are started. If the dependent regions are placed in either the mean-time-to-wait group or the rotate priority group, they can be at different relative dispatching priorities at any given time during their execution. The rotate type of dispatching is usually adequate for dependent regions that process the same scheduling classes. If the scheduling of dependent regions is by class according to type of work, a unique dispatching priority is a better alternative.

Not only is it important that the IMS control region get CPU service when needed, but as a result of parallel DL/I, the same is true for the dependent regions. The DLS address space requires a dispatching priority just below that of the control region. If other tasks on the z/OS dispatcher sequence queue come between the IMS control address spaces and the dependent regions, the intervening tasks receive service before the dependent regions.

Therefore, raising the DPRTY of the dependent regions higher than that of the intervening jobs might improve transaction response time.

The message processing regions need a higher dispatching priority than the BMPs, but the BMPs must immediately follow them, because the BMPs contend for the same IMS software resources (buffer handler latches and program isolation enqueues) as the message regions. A BMP could be interrupted while holding an IMS software resource and, therefore, suspend either the control address spaces or the MPP's execution. The address spaces needing the resource wait until the BMP is redispached and the resource is released. Tasks intervening between MPPs and BMPs could cause dependent-region elapsed time to increase, even though the dependent regions are getting adequate CPU service.

The number of message-processing regions has a direct effect on transaction response time. If too few regions are available, the transactions can remain on the input queue for a long period of time. Conversely, if too many regions are processing the same application workload, there can be contention for the database records (PI enqueues).

Having too many message-processing regions can increase CPU consumption by reducing the number of transactions per scheduling. This in turn causes more schedules and program loads. Too many regions also demand more storage, both for the regions themselves and for pools, such as the IMS PSB pool and ENQ/DEQ pool. They also potentially increase contention among regions for IMS buffer pools, IMS data sets, enqueued resources (PI), internal IMS latches, and program load libraries.

In environments with fewer transaction types, higher levels of occupancy can be tolerated, because the application programs might be able to remain in the regions and process multiple transactions without causing other transactions to queue.

6.8 IMS variable pool considerations

Table 6-2 shows the names of the main buffer pools according to where they are referenced.

Table 6-2 Variable pools

POOL name	System definition name	Procedure name	Fix list name	/DISPLAY POOL name	Location
DMB	DMB	DMB	DLDP	DMBP	EPVT(DLISAS)
PSB	PSB	PSB	DLMP	PSBP	ECSA
CSAPSB	CSAPSB	CSAPSB	DLMP	PSBP	ECSA
DLIPSB	DLIPSB	DLIPSB	DPSB	PSBP	EPVT(DLISAS)
DB work pool	DBWP	DBWP	DBWP	DBWP	EPVT(DLISAS)
PSB work pool	PSBW	PSBW	PSBW	PSBW	ECSA
EPCB	EPCB	EPCB	EPCB	EPCB	ECSA

PSB pools (DPSB and PSBP)

If LSO=Y (default) is specified in the DFSPBxxx member of IMS.PROCLIB, then a single PSBP pool is allocated in ECSA storage.

Alternatively, if LSO=S is specified in the DFSPBxxx member of IMS.PROCLIB, the PSB pool is divided in two pools:

- PSBP containing TP PCBs and Fast Path PCBs (about 10% to 20% of the old PSB pool) and is allocated in ECSA storage.

- DPSB containing full function PCBs (about 80% to 90% of the old PSB pool) and is allocated in the DL/I SAS extended private area.

LSO=S frees CSA storage, allows more space for some buffers (because they are relocated in ECSA), and generally uses significantly less CPU than those using LSO=Y.

Table 6-3 summarizes where IMS allocates each area of storage depending on the value specified in the LSO parameter.

Table 6-3 Storage allocation

	LSO=Y		LSO=S	
	Fast Path	No Fast Path	Fast Path	No Fast Path
Fast Path buffers	ECSA storage		ECSA storage	
VSAM buffers	IMS control region private area		DL/I SAS extended private area	
OSAM buffers			DL/I SAS private storage both above and below 16 MB line	
Control blocks ^a				
DL/I code			ECSA storage	
Intent list				
Log buffers	ECSA storage	IMS control region private area	ECSA storage	
ENQ/DEQ pool			ECSA storage	DL/I SAS extended private area
PSB pool	Single PSB pool in ECSA storage (100%)		PSBP in ECSA (10%-20%) DPSB in DL/I SAS extended private area (80%-90%)	

a. With LSO=Y, VSAM data set control blocks that are created when data sets are opened are allocated in CSA storage.

DMB pool (DMBP)

If the IMS Monitor is activated after all databases are opened, any IWAITs for DMB indicate either that the DMB pool size is too small or that a /DBR was issued earlier for the database and it is now being referenced again.

Each time a DMB has to be loaded into the pool and the necessary free space is unavailable, one or more databases not currently being referenced must be closed and their DMBs deleted to make pool space for the new DMBs being referenced.

After the new DMB is loaded into the pool, the database must be opened. Opening and closing databases suspends either the IMS control region or the DL/I SAS, depending on the specification of the LSO= option, and increases DL/I call elapsed time.

In the case of a DMB pool, page faulting is preferable to database open and close, so make the DMB pool big enough so that the open and close process does not take place due to DMB pool shortages.

We recommend that you make the DMBs for all databases resident (which IMS then loads into a pool called DFSDMBRS). If all the DMBs are made resident, the inactive DMBs can be allowed to page out without significantly affecting the active DMBs, but paging them back into storage can suspend processing in the control region or the DL/I SAS. We also recommend that you page fix the resident DMBs by means of specifying the DFSDMBRS module name in the z/OS fix list.

If you make all DMBs resident in the IMS system definition, leave a DMB pool size of at least 12 KB. This size allows a DMB that is not specified as resident in the IMS system definition to be loaded. You should monitor this pool regularly to determine whether nonresident DMBs exist.

If online change is used for the system, the DMB pool must be large enough to contain all of the resident DMBs that are to be changed by online change. The resident DMBs that are changed by online change are closed, and the new DMB is loaded into the DMB pool when referenced again. It is not reloaded into the resident DMB storage DFSDMBRS.

Monitor the usage of the DMB pool by the following actions:

- ▶ Be sure that the full IMS system has been running long enough so that all the databases are open.
- ▶ Use the **/DIS POOL DMBP** command to display the maximum pool occupancy.
- ▶ Repeat the process regularly.

The Variable Pool Statistics section of IMS Performance Analyzer Internal Resource Usage report reports the actual amounts of the DMBP pool used.

In summary, we have the following recommendations:

- ▶ Make all of the DMBs resident.
- ▶ Allow enough space in the DMB pool for online changed DMBs.
- ▶ Do not underallocate the DMB pool.

Database work area pool (DBWP)

This pool is used as a work area for segments being deleted and by other calls. A reasonable size for this pool can be calculated using 2 KB for each PST to be allocated by the control region. After the IMS system is up and executing for a while, use the highest allocated size for this pool to determine the new size. Allocate an additional 4 KB above this size.

PSB work pool statistics (PSBW)

All work areas required for the PSBs are allocated from this pool. No I/O ever occurs for this pool. The size of the pool is listed as *Size* and the maximum amount of space used is listed as *High*. Any requests for space that cannot be satisfied from this pool are reported in the IMS Monitor reports.

Make the PSBW pool large enough to contain the largest unit (PSB or PSBW) in the system, times the number of active dependent regions, plus 20% for possible fragmentation. For example, if the largest PSB work area in the system is 30 000 bytes and 50 dependent regions are active, the PSBW pool minimum size is:

$$1.2 \times (30,000 \times 50) = 1,800,000 = 1.8 \text{ MB}$$

Another way to calculate the appropriate size of the PSBW pool is to scan the ACBGEN output, which shows the largest PSB work area. If the largest PSB work area found on the ACBGEN is for an active online PSB, this size can be used for the calculation of the size of the PSBW pool.

The Variable Pool Statistics section of IMS Performance Analyzer Internal Resource Usage report shows the actual amounts of the PSBW pool used.

EPCB pool statistics (EPCB)

When any PSB containing Fast Path PCBs is scheduled, extended PCBs are built in the EPCB pool.

The formula for calculating the size of the EPCB pool can be found in Chapter 3 in the section “BUFPOOLS Macro” in *IMS Version 9 Installation Volume 2: System Definition and Tailoring*, GC18-7823:

$$\begin{aligned} \text{EPCB} = & (\text{number of TP-PCBs} * 28) + \\ & (\text{number of MSDBs} * 76) + \\ & (\text{number of DEDB SENSEGs} * 124) + 132 \end{aligned}$$

The actual EPCB space requirements for each PSB are listed by the ACBGEN utility. If the EPCB pool is too small, dependent region scheduling failures (internal conflicts) occur. The Variable Pool Statistics section of IMS Performance Analyzer Internal Resource Usage report shows the actual amounts of the EPCB pool used.

6.8.1 Relationship with scheduling

The dependent region might run into scheduling failures if the pool size is small. Five pools are associated with scheduling:

- ▶ PSB pools (PSBP and DPSB)
- ▶ PSB work pool
- ▶ EPCB pool
- ▶ DMB pool
- ▶ DMB work pool

I/O activity to IMS.ACBLIB or pool-space failures during application program scheduling indicate the need to tune these pools.

A *pool-space failure* in any of these pools indicates that a request for space cannot be satisfied from unallocated space (for PSBs, PSB work areas, or DMBs needed by scheduled application programs). Thus, the pool that caused the pool space failure is too small to handle the concurrent dependent region activity and needs to be increased.

We recommend that you apply the following rule when determining the starting size of these pools: each pool should be large enough to contain the largest unit (PSB or PSBW) in the system, times the number of active dependent regions, plus 20% for possible fragmentation.

For example, if the largest PSB work area in the system is 30 000 bytes and 50 dependent regions are active, the PSBW pool minimum size is:

$$1.2 \times (30,000 \times 50) = 1,800,000 = 1.8 \text{ MB}$$

This still might not be enough because of fragmentation. Therefore, we recommend that you monitor these pools using the **/DIS POOL ALL** command or IMS Performance Analyzer to determine whether the pool size is adequate. The Variable Pool Statistics section of IMS Performance Analyzer Internal Resource Usage report can be used for monitoring the scheduling pools. If the monitoring of the pools shows that the maximum number of bytes used in the pool is well below the allocated storage size, we recommend that you reduce the size of the pool to approximately 8 KB above the maximum number of bytes used in the pool.

The exceptions are the PSB and DPSB pools, which attempt to use all available storage. We recommend that you execute the IMS Performance Analyzer Internal Resource Usage report with a log data set that contains two checkpoints taken during the peak time in order to determine whether some of the pools are nearing their maximum size.

IMS.ACBLIB I/O activity can occur to the PSB and DMB pools if they are not large enough to hold all of the PSBs, nonresident intent lists, and DMBs referenced by the system. IMS.ACBLIB I/Os are undesirable for DMBs, except at IMS startup or following **/DBR** and **/STA** database commands.

We recommend that you make the pools large enough to minimize the number of I/Os, yet at the same time small enough to avoid a substantial increase of the constraints on real storage. IMS Block Mover is a serial resource; therefore, other dependent regions might have to wait to schedule when the block mover is busy loading the application's control blocks or when dynamic allocation is holding a block mover latch.

An acceptable number of IMS.ACBLIB I/Os is less than one per scheduling. Each I/O increases the application program schedule time not only for this application but for others as well.

If the number of I/Os per transaction to IMS.ACBLIB is more than one, this indicates a possible inadequate allocation of the PSB or DMB pool (the PSBW, EPCB, or DMBW pools never experience I/O activity).

The potential exists for a minimum of three IMS.ACBLIB I/Os per scheduling:

- ▶ One I/O for the intent list (INT=)
- ▶ One I/O for the PSB (PSB=)
- ▶ One I/O for the DMB (DMB=)

The number of I/Os for each PSB or DMB depends on the size of each and the block size of IMS.ACBLIB. We recommend that you make the ACBLIB block size 32 KB.



Performance considerations for DBCTL

This chapter discusses some items specific to DBCTL with references to other sections in this book containing information about specific parameters.

This chapter describes the following topics:

- ▶ DFSPZPxx
- ▶ Scheduling
- ▶ IMS (DBCTL) startup parameters
- ▶ Parallel Sysplex considerations

7.1 DBCTL performance considerations

Information related to general database performance is contained in Figure 5 on page 53 and is common to both IMS DB/TM and DBCTL environments. However, there are specific considerations for DBCTL, which are covered here. The module is DFSPZPxx. It is assembled and linked by the user and then used when the CCTL connects to DBCTL.

7.2 DFSPZPxx

DFSPZPxx module, referred to as the DRA startup table, is assembled and linked by the user into a library accessible to the CCTL region, which is typically CICS. It contains specific parameters, which affect the resources in DBCTL. The following parameters can have a significant effect on performance:

- ▶ **FPBUF=**

FPBUF specifies the number of Fast Path buffers that are allocated to each DBCTL thread from this CCTL. The MINTHRD and MAXTHRD values therefore must be considered as well as the CNBA value. This value multiplied by the MINTHRD value is the number of buffers needed in the IMS DBBF pool and must be within the CNBA value. This value is similar to the NBA value specified for IMS dependent regions.

- ▶ **FPBOF=**

FPBOF is the number of overflow buffers which can be used by each thread. This is similar to the OBA value specified for IMS dependent regions.

- ▶ **CNBA=**

DBCTL users need to consider how many threads (PSTs) should be defined in DFSPZPxx (DRA table) for the CICS connections. This DRA table is defined in IMS but used by CICS AORs.

CICS/DBCTL users might end up reaching the MAXTASKs defined in CICS side, so how can you determine the minimum number of threads defined in IMS to support the expected workload? Use the following formula:

Minimum thread numbers = Elapsed execution time(millisecond) * Transaction per second / 1000

In this formula, Elapsed execution time(millisecond) can be obtained from either the IMS Monitor report or IMS Performance Analyzer. Because you never want to fully use all threads, allocate at least 25% more threads.

For example, if the elapsed execution time is about 60 milliseconds, and the workload for this IMS is 500 transactions per second, the formula gives the following result:

$$60 * 500 / 1000 / 75\% = 40$$

So, we need 40 threads for that IMS to support the workload of 500 transactions per second in this case. Use IMS Performance Analyzer or IMS Monitor report constantly to monitor if the region occupancy is under 75% or the workload is greater than your expectation.

- ▶ **MINTHRD=**

MINTHRD specifies the number of thread TCBs to be allocated when this CCTL connects to DBCTL. The PST= value in IMS should be large enough to accommodate this value times the number of CCTLs plus any BMPs.

- ▶ **MAXTHRD=**

MAXTHRD is the maximum number of thread TCBs which can be created by this CCTL. Remember that this value is for each CCTL and relates to the PST and MAXPST values in IMS.

7.3 Scheduling

CCTL (CICS) applications issue schedule requests within the application when there is a need to access IMS resources. Because this is done from within the application, there are no scheduling options, such as PWFI with IMS TM. All requests to schedule a PSB are handled as separate new requests. All of the same pool considerations for the variable pools apply to DBCTL and are discussed in 6.8, “IMS variable pool considerations” on page 144 and therefore are not duplicated here. For reference, the list of pools to be considered are:

- ▶ CSAPSB
- ▶ DLIPSB
- ▶ PSBW
- ▶ DMB
- ▶ DMBW, which is not actually allocated at schedule but is a variable pool
- ▶ EPCB

PSBs used by CCTL must be defined to DBCTL. These are defined by the use of the APPLCTN macro. Unless there is some specific reason to do otherwise, you should always specify SCHDTYP=PARALLEL to allow parallel scheduling of these PSBs.

7.4 IMS startup parameters for DBCTL

The following DFSPBxxx parameters apply to DBCTL and can have an impact on performance. In some cases, a brief explanation is given or a reference to the parameter description is shown. The parameters are:

- ▶ ARC=
The primary reason that we list the ARC parameter as performance-related is because you need to make sure that you archive as soon as possible to avoid any delays due to unavailable OLDS.
- ▶ BSIZ=
BSIZ must accommodate your largest DEDB CI size but should not be oversized, because it can waste a significant amount of virtual and fixed storage depending upon the DBBF and DBFX settings.
- ▶ CPLOG=
Set CPLOG large enough to avoid excessive IMS checkpoints. Every 10 to 20 minutes should be a reasonable number.
- ▶ CSAPSB=
See 6.8, “IMS variable pool considerations” on page 144.
- ▶ DBBF=
See 6.4, “IMS parameters” on page 130.
- ▶ DBFP=
See 6.4, “IMS parameters” on page 130.
- ▶ DBFX=

See 6.4, “IMS parameters” on page 130.

► DBWP=

See 6.8, “IMS variable pool considerations” on page 144.

► DLIPSB=

See 6.8, “IMS variable pool considerations” on page 144.

► DMB=

See 6.8, “IMS variable pool considerations” on page 144.

► EPCB=

See 6.8, “IMS variable pool considerations” on page 144.

► FIX=

The FIX parameter specifies the suffix for the DFSFIXxx and DFSDRFxx IMS.PROCLIB members, where you can specify page fixing and DREF storage options. The most important option in DFSFIXxx is typically the EPST option if Fast Path is used.

► LGNR=

See 6.4, “IMS parameters” on page 130.

► MAXPST=

MAXPST must accommodate the MAXTHRD value from all CCTLS that connect to this DBCTL plus any BMP regions.

► OTHR=

See 6.4, “IMS parameters” on page 130.

► PIINCR=

See 6.4, “IMS parameters” on page 130.

► PIMAX=

See 6.4, “IMS parameters” on page 130.

► PSBW=

See 6.8, “IMS variable pool considerations” on page 144.

► PST=

The PST parameter value should be large enough to handle the MINTHRD from all CCTLS plus any BMPs. It is usually best to make it even larger to accommodate changes in MINTHRD, concurrent BMPs, or the number of connected CCTLS.

► RES=

Specify RES=Y to make the intent list and any PSBs that were defined at system definition resident in storage at initialization.

► SPM=

SPM is used to override the IMS default fixed storage pool manager specifications. In a DBCTL environment, there should be no need to override the defaults.

► VSPEC=

The VSPEC parameter specifies the suffix of member DFSVSMxx, which contains many important performance-related specifications. See Figure 5 on page 53 for additional information about database buffer pools.

► WADS=

While this is mainly an availability parameter, note that IMS writes concurrently to each WADS but waits for both to complete before posting any waiting tasks, so it is important to make sure if using DUAL WADS that both are on the best performing DASD.

7.5 Parallel Sysplex

Parallel Sysplex considerations are discussed in Chapter 11, “IMS Parallel Sysplex considerations” on page 201. Everything, except for the part about shared queues, applies to DBCTL.

Archived

System considerations

This chapter discusses some of the tasks that are general, yet critical to the functioning of IMS. Although we can debate whether these functions are actually a component of IMS Transaction Manager (IMS/TM) or IMS Database Manager (IMS/DM), these are really system functions that support both IMS/TM and IMS/DM.

This chapter discusses the following:

- ▶ The IMS logger and its components
- ▶ IMS Database Recovery Control Facility (DBRC)
- ▶ Resource Access Security (RAS) using Resource Access Control Facility (RACF)
- ▶ Batch and utility performance

8.1 The IMS logger

Logging in IMS is performed by the IMS control region under the control of a task control block (TCB) called the logger TCB. The purpose of the IMS logger is to provide the following:

- ▶ Record the status of the system through periodic checkpoints
- ▶ Record the status of changed database segments
- ▶ Cater for recoverability and integrity of IMS data
- ▶ Provide an audit trail
- ▶ Provide performance statistics

Every task in IMS must log. This implies that each log record has to be uniquely identified and information contained in the log record must pertain to a specific unit-of-work. It is this unit-of-work, together with the recovery token unique to each log record, that makes the logger exceptionally powerful by allowing IMS to cater for the integrity and recoverability of all work under its control.

Internally, IMS consists of two logging functions: the logical logger and the physical logger. Naturally, both these log functions have their own TCBs. The responsibility of the logical logger is to ensure that all log requests are moved to the log buffers. The responsibility of the physical logger is to ensure that the log buffers are written to disk.

In order to log data, IMS makes use of two types of logs: the write ahead data set (WADS) and the online log data set (OLDS). The WADS is used to externalize log records that IMS considers critical. The WADS is used in a wraparound fashion, whilst the OLDS is used in a round-robin fashion. IMS ensures that data written to the WADS is transferred to the OLDS. The write to the OLDS is done either when a track worth of buffers is filled or a timer pops every second, whichever occurs first. As the OLDS fills up, IMS schedules an archive utility to archive the OLDS to the system log data set (SLDS) and the recovery log data set (RLDS). Once archived, the OLDS can be reused.

IMS employs *WAIT WRITES* and *CHECK WRITES* to externalize log records to the WADS. A *WAIT WRITE* is a synchronous request to write log records and a *CHECK WRITE* is an asynchronous event to write log records.

Now we look at some of the log records produced and how IMS formats log records. Figure 8-1 on page 157 provides a view on the elements that make up the log record. The elements of the log record are as follows:

- ▶ The “LL” field shows the size of the log record.
- ▶ The “ZZ” field which is generally binary zeroes.
- ▶ The record identifier indicates the record type (1 byte).

Codes above x'80' are considered by IMS to be user log records, although IMS can write the x'99' record in the Change Data Capture Exit.

- ▶ A subrecord identifier indicates the subrecord type (1byte).
- ▶ The record content as mapped out by the relevant DSECT.
- ▶ A store-clock (STCK) value indicates the time the record was cut.
- ▶ The log sequence number (LSN), which is unique to each record.

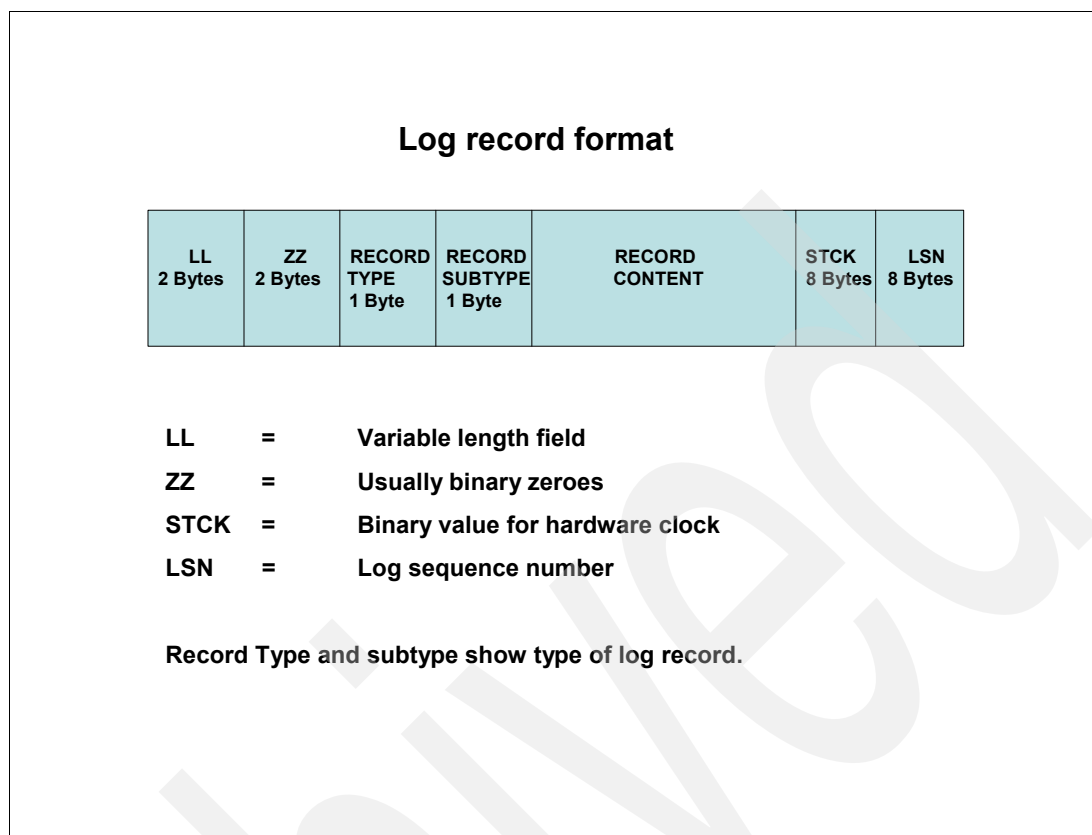


Figure 8-1 Log record structure

8.1.1 Logging considerations

Due to the enormous amount of logging that a large IMS system can perform, and the effect it can have on the performance of your system, the following considerations can help ensure optimum performance of your IMS system:

- ▶ When running on a CPU with 64-bit support (z/Architecture), the log buffers are page fixed in real storage above the 2 GB line. This feature allows you to specify the maximum possible amount of log buffers, which is 9999, without incurring a storage constraint. In order to take advantage of this enhancement, the OLDS block size is required to be a multiple of 4 096 (without exceeding half track value). We recommend a block size of 24 576, because it still retains half-track blocking. In the past, a block size of 26 624 was recommended and this can still be used, but you are not able to make use of storage above the line. Evaluate the usage of your log buffers and determine whether you need to allocate maximum buffers above the line. If you increase your log buffers, pay particular attention to the size of your WADS, because the size of the WADS must be increased.
- ▶ Ensure the correct number of OLDS buffers are defined in DFSVSMxx. The current maximum limit is 9 999 buffers. Example 8-1 shows the DFSVSMxx statements that need to be defined. It is also good practice to define more OLDS dynamically. This can only be achieved if you specify the OLDS in DFSVSMxx and set up the dynamic allocation members accordingly.

Example 8-1 Sample OLDS and WADS definition in DFSVSMxx

```

OLDSDEF OLDS=(00,01,02,03,04,05,06,07,08,09,10),BUFNO=30,MODE=DUAL
WADSDEF WADS=(0,1,2,3)

```

- ▶ Dual WADS need to be defined to ensure resilience is in effect. However, today's hardware and operating systems have built-in increased throughput and resilience. The decision to use dual WADS is generally based on your infrastructure and business requirements, which also applies to running dual OLDS.
- ▶ Allocation of OLDS and WADS must be made on separate controllers across your DASD infrastructure. Generally, the WADS must be placed on a controller with good caching functions. The WADS generally tends to show high I/O rates and therefore optimum and consistent response times must be achieved.

8.2 DBRC

DBRC forms an integral part of IMS. IMS relies on DBRC to record and manage recovery-related information about all aspects of IMS. DBRC keeps this information in a set of VSAM data sets that are collectively called the Recovery Control (RECON) data set. DBRC also provides information to IMS about how to proceed for certain IMS actions. DBRC specifically does the following:

- ▶ Helps you ensure IMS system and database integrity by recording and managing information associated with the logging process.
- ▶ Assists IMS in the restart process by notifying IMS which logs to use for restart.
- ▶ Assists IMS to allow or prevent access to databases in data-sharing environments by recording and managing database authorization information.
- ▶ Facilitates database and log recovery by:
 - Controlling the use and integrity of the information in the logs.
 - Recording and maintaining information about the databases and logs in the RECON data set.
 - Generating and verifying the Job Control Language (JCL) for various IMS utility programs.
- ▶ Supports Extended Recovery Facility (XRF) by identifying (in the RECON data set) if the subsystem is XRF capable.
- ▶ Supports Remote Site Recovery (RSR) by containing the RSR complex definition in the RECON data set and providing other services associated with controlling RSR.
- ▶ Supports IMSplexes by notifying all DBRCs in the same IMSplex when one of the DBRCs performs a RECON data set reconfiguration.
- ▶ Provides enhanced database function support for:
 - HALDB
 - Unrecoverable databases
 - Concurrent Image Copy
 - Image Copy 2
 - Fast Path VSO options
 - Online reorganization

Figure 8-2 on page 159 provides us with a view of the various record types that exist in DBRC.

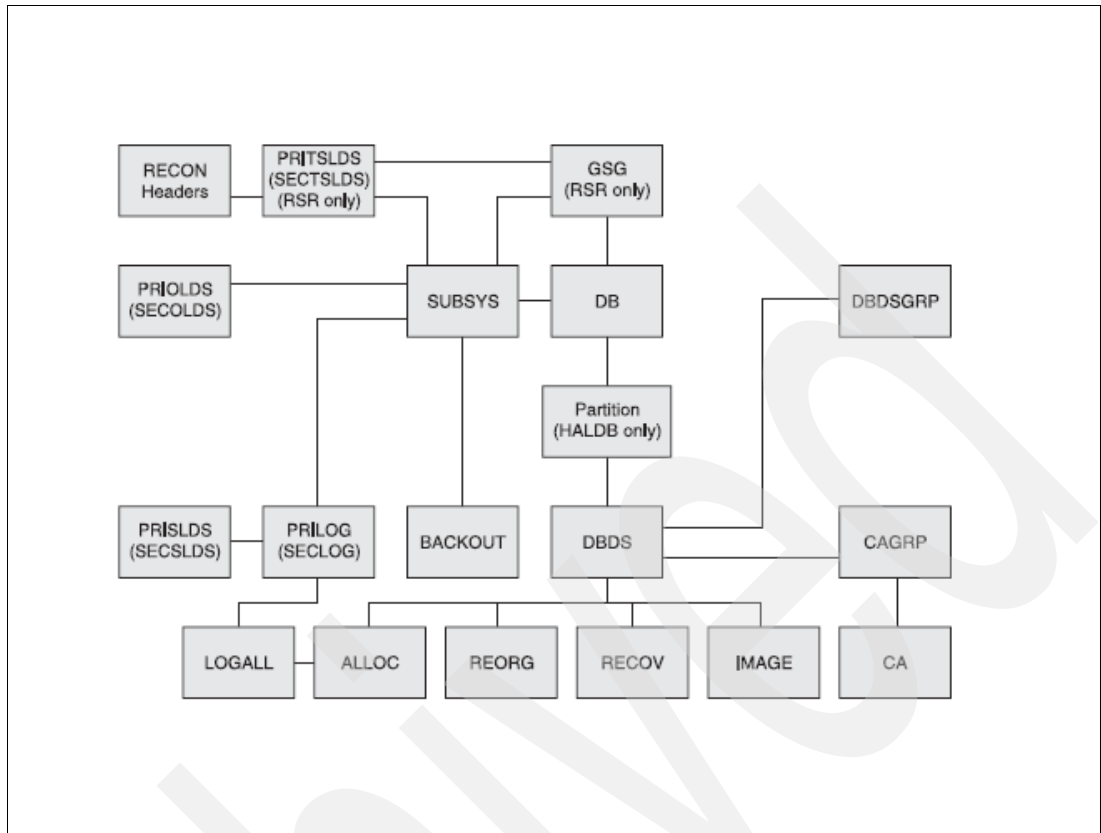


Figure 8-2 DBRC record types within RECON

8.2.1 DBRC performance considerations

There are a number of performance considerations with regard to DBRC and the management of the RECON data sets. The *IMS Version 9 Database Recovery Control Guide and Reference*, SC18-7818, provides detailed information about this topic. We highlight some of the major performance considerations with DBRC.

8.2.2 Defining the RECON data sets

All information pertaining to DBRC is kept on a VSAM key-sequenced data set (KSDS). DBRC uses two RECON data sets to increase availability and recoverability. They contain identical information. The data sets are identified by the DD names, RECON1 and RECON2. We *strongly* recommend that you define a third RECON data set (RECON3) to be a spare data set. This spare data set is empty until an error occurs on one of the two active RECON data sets, or you issue the **CHANGE.RECON REPLACE(xxx)** command. Then, DBRC copies the good RECON to the spare data set.

Naturally, there are performance issues pertaining to VSAM and the RECON definition. Specifically, they are:

- Use the same index control interval (CI) size and data CI size for all RECON data sets. Ensure that the specified data CI size exceeds the index CI size by at least 2 048 bytes, or else you degrade your RECON performance.

- ▶ Space allocation for IMS Version 9 must be twice the IMS Version 7 size.
 - Make the three RECONS different sizes (make the SPARE the largest), so that they do not all run out of space at the same time.
- ▶ Use secondary allocation in the event that your RECON data set runs out of space.
- ▶ We recommend the following keywords on the DEFINE CLUSTER command:
 - CONTROLINTERVALSIZE: Our recommendation is 8 192 K for data and 4 096 K for the index. Remember that the index CI size must be at least 2 048 bytes smaller than the data size and that all index and data components must have the same CI size.
 - FREESPACE: Our recommendation is that you specify at least 50% free space, and you can lower this amount of free space when you are happy with the usage of the RECON data sets.
 - Make the FREESPACE characteristics different on each RECON so that they do not take CI and CA splits at the same time.
 - KEYS must be specified as KEYS(32 0).
 - We recommend NONSPANNED, because DBRC always writes records smaller than the VSAM CI size. DBRC divides its own records into segments, each of which is always smaller than a single control interval and each of which is seen by VSAM as a complete physical record. VSAM record spanning is not used. Segmenting allows a logical RECON record to be as large as 16 MBs independent of the VSAM RECORDSIZE parameter.

However, this does not mean that the RECON records can grow indefinitely. A very fast growing PRILOG record can indicate incorrect image copy frequency, due to an unexpected increased volume of logging. So it might be useful to readjust your threshold values:

- LOGALERT(dsnum,volnum), which triggers the DSP0287W warning message
- SIZALERT(dsnum,volnum,percent), which triggers the DSP0387W warning message or the DSP0007I informational message

These two parameters give you the opportunity to react, such as deleting inactive logs, executing Image Copy, or ensuring that log compression is not inhibited, and to correct this before the IMS system abends. The two parameters give you time to determine what is causing the extremely large PRILOG record. You probably want to issue the messages much earlier than the default values do, so the value settings might need to be readjusted. You can find examples of more reasonable values in the IBM Redbook, *IMS Version 8 Implementation Guide A Technical Overview of the New Features*, SG24-6594.

- We recommend that RECORDSIZE is CI size minus 7 bytes. If the CI size was defined as 8 192 bytes, then RECORDSIZE (4086,8185) would apply.
- SHAREOPTIONS must be specified as SHAREOPTIONS (3,3).
- SPEED is recommended, because the initial load is faster.
- NOWRITECHECK is recommended, because IMS manages dual copies of the RECON, which eliminates the need for writing checks.
- ▶ Use dynamic allocation for all your RECON data sets across all your IMS systems. RECON data set names must not be coded in JCL. It is also easier to deallocate the data set if required.
- ▶ Ensure that the RECON data sets are monitored and reorganized when required. When performing a reorganization, always cycle through all RECON data sets to make sure that RECON1 and RECON2 are always indicated as COPY1 and COPY2. Remember to back up your RECON data sets before you perform any reorganization.

- To avoid deadlock situations, give special consideration to the placement of RECON data sets that are shared among multiple processors. During a physical open, DBRC reserves RECON1, RECON2, and RECON3. DBRC determines which are available and which are Copy1, Copy2, and spare. DBRC then closes and dequeues the spare (if it exists) and any unusable RECON data sets. So, during the use of DBRC, two RECON data sets are reserved most of the time. DBRC always reserves both RECON data sets in this order: RECON1 and RECON2. If RECON1 and RECON2 are specified consistently throughout jobs, DBRC does not encounter deadlock.

8.2.3 Resolving data set contention problems

The availability and performance of the RECON data sets are dependent on whether Global Resource Serialization (GRS) is running in a star or ring configuration. GRS provides methods to convert a RESERVE into a ENQ request. The ENQ, DEQ identifies a resource by its symbolic name containing major name, minor name, and scope. To ensure that resources are treated as you want them to be treated without changes to your applications, GRS serialization provides three resource name lists (RNLs):

- **SYSTEMS EXCLUSION RNL**
A list of resources that are requested with a scope of systems that you want global resource serialization to treat as local resources.
- **RESERVE CONVERSION RNL**
A list of resources that are requested on RESERVE macros for which you want global resource serialization to suppress the RESERVE.
- **SYSTEM INCLUSION RNL**
A list of resources that are requested with a scope of system that you want global resource serialization to treat as global resources.

We recommend in order to avoid contention problems:

- In a GRS star configuration, a RESERVE CONVERSION RNL should be implemented for DSPURI01 if all systems accessing the RECON data sets are within the sysplex (or GRSPlex). If the RECON data sets are accessed by systems that are outside of the sysplex, the RESERVE must not be converted. A SYSTEMS EXCLUSION RNL must be implemented instead. If you implement GRS RNL CONVERSION by adding the QNAME for the RECON data set, DSPURI01, to the conversion list, the hardware reserve is eliminated and replaced by a GRS enqueue that is communicated to all other sharing z/OS systems. Other data sets on the same DASD volume can be used while the RECON data set is reserved; this is the benefit of performing the RNL conversion. GRS RNL CONVERSION uses CPU and storage and affects system performance positively. The performance (in terms of the least CPU time used, the least storage used, and the least elapsed time) is best by using this option in a GRS STAR configuration.

In an IMS Fast Path 2-way data sharing environment comparison done in a controlled environment at the Silicon Valley Laboratory in which 6000 IMS Fast Path areas were opened in parallel using BMP, a significant reduction in total elapsed time was observed when the reserve is converted. In a GRS star configuration, placing DSPURI01 in the conversion list, as well as both SYSZVVDS and SYSVTOC (these two should always be in the same RNL), produced a roughly 3.5% reduction in total BMP elapsed time.
- In a GRS ring configuration, a SYSTEMS EXCLUSION RNL should be implemented for DSPURI01 so that the RECON is serialized by the hardware reserve. In this case, the following also applies:
 - The RECON data sets must be the only objects in their respective catalogs.

- The RECON data sets must be on the same device as the catalog and isolated from other data sets.

Note: Batch jobs also ENQ on a minor name so that they serialize on each other and avoid monopolizing the RECON to the detriment of online IMSs.

8.2.4 DBRC RECON maintenance

Monitoring the status of IMS RECON data sets should be one of the IMS database system administrator's routine tasks. IMS RECON data sets can be considered as the heart of the IMS system, especially when data sharing has been enabled. Even without data sharing, RECON data sets are used to store recovery-related information about subsystems, logs, and utility executions along with other pertinent information. The RECON data sets are critical resources for IMS. If all RECON data sets are lost, IMS cannot continue processing and it terminates abnormally.

RECON data sets are VSAM key-sequenced data sets (KSDS) that should be periodically reorganized. For direction and guidance about allocating RECON data sets, refer to the *IMS Version 9 Database Recovery Control Guide and Reference*, SC18-7818.

You can enter the TSO command, **LISTC ENT('IMS910.RECON1') ALL**, to determine the number of records, CI splits, and CA splits, in order to determine whether the RECON data sets might require reorganization. When one of the RECON data sets is in a DISCARDED state, take advantage of the situation and perform a reorganization on that one data set. A RECON data set can become discarded by IMS due to a problem or by issuing a DBRC **CHANGE.RECON REPLACE()** command.

RECON data sets can be reorganized by both an online process and a batch process. These processes should only be run during a slow period of activity in the IMS region because copying the RECON data sets from one data set to another might affect the performance of IMS.

If a RECON data set is discarded by IMS, you should, at a minimum, perform the delete and define step for the data set that has been discarded, in order to maintain two RECON data sets and one spare. The other RECON data sets could then be reorganized at a slower processing period.

Following is a process that can be used to reorganize the RECON data sets. To execute the process while online IMS systems are accessing the RECONS, the RECON data sets need to be dynamically allocated:

1. Enter the DBRC command to determine the status of the RECON data sets. If entering DBRC commands from the z/OS MCS or E-MCS console, be sure to enter a period at the end of the command or the IMS system issues the message "DFS972A *IMS AWAITING MORE INPUT*." The syntax of the command is as follows when entered as an IMS command:

```
/RML DBRC='RECON STATUS'.
```

You can also execute this command by submitting a job similar to the one in Example 8-2 on page 163. The command to be used, **LIST.RECON STATUS**, is provided in the SYSIN stream.

You might also want to delete unnecessary log information from the RECON by issuing the **DELETE.LOG INACTIVE** command before starting the reorganization. This can be done in the same job, as in Example 8-2 on page 163.

Example 8-2 Job for executing DBRC command

```
//JOUK05RL JOB CLASS=A,MSGCLASS=X,REGION=4M
//*****
//*      USE: RECON LIST
//*****
//*
//JS010   EXEC PGM=DSPURX00
//*
//STEPLIB DD DSN=IMS910G.SDFSRESL,
//          DISP=(SHR,KEEP,KEEP)
//*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//*
//SYSIN   DD *
LIST.RECON STATUS
DELETE.LOG INACTIVE
/*
```

2. Example 8-3 shows the most interesting part of the output from the command.

Example 8-3 LIST.RECON STATUS before starting the reorganization

-DDNAME-	-STATUS-	-DATA SET NAME-
RECON1	COPY1	IMS910G.RECON1
RECON2	COPY2	IMS910G.RECON2
RECON3	SPARE	IMS910G.RECON3

3. Issue **CHANGE.RECON REPLACE(RECON1)** to discard RECON1. RECON2 is copied to RECON3, and RECON1 is discarded. Output of the command is shown in Example 8-4.

Example 8-4 Output of the CHANGE.RECON REPLACE(RECON1) command

```
CHANGE.RECON REPLACE(RECON1)
DSP0380I RECON2 COPY TO RECON3 STARTED
DSP0388I SSID=IMSG FOUND
DSP0388I 0001 SSYS RECORD(S) IN THE RECON AT RECONFIGURATION
DSP0381I COPY COMPLETE, RC = 000
DSP0242I RECON1 DSN=IMS910G.RECON1
DSP0242I REPLACED BY
DSP0242I RECON3 DSN=IMS910G.RECON3
DSP0203I COMMAND COMPLETED WITH CONDITION CODE 00
DSP0220I COMMAND COMPLETION TIME 06.277 14:58:42.8
```

4. After the **CHANGE.RECON REPLACE(RECON1)** command, RECON2 is the new COPY1 and RECON3 is the new COPY2, and RECON1 is in DISCARDED status, such as Example 8-5.

Example 8-5 LIST.RECON STATUS after replacing RECON1

-DDNAME-	-STATUS-	-DATA SET NAME-
RECON2	COPY1	IMS910G.RECON2
RECON3	COPY2	IMS910G.RECON3
RECON1	DISCARDED	IMS910G.RECON1

5. Before you can delete the cluster, you must ensure there are no other users, such as batch jobs, that are using the data set. If you are using the automatic RECON loss notification feature, all DBRC instances, which have been registered with the Structure Call Interface (SCI), are automatically notified about the reconfiguration and the discarded

RECON data set. The SCI is used for the communication between all registered DBRC instances across your IMSplex. The unavailable RECON data set gets discarded instantly from all notified DBRC instances. As long as you are using dynamic allocation for your RECON data sets, all involved DBRC instances also deallocate the discarded RECON data set.

If you are not using automatic RECON loss notification feature, then online IMS subsystems, when they next access the RECONS, release the RECON that has been discarded. You can force IMS to access the RECON by issuing `/RML DBRC='RECON STATUS'` or the `/DIS OLDS` command.

Note: The automatic RECON loss notification feature can be used to simplify the process. The automatic RECON loss notification feature is optional and is automatically enabled if the SCI address space is up and running on the same z/OS image and the DBRC instance registers with SCI as a member of IMSplex.

6. Now you need to delete and define the VSAM cluster for the discarded RECON by using the appropriate attributes from “Defining the RECON data sets” on page 159. It is a good practice to store the IDCAMS statements for each RECON, because you might want to use the same attributes that were used when the RECON was previously allocated.

For the above RECON listing, delete and define RECON1, which is currently discarded. If you now display the RECON again, you see that RECON1 is now shown as a SPARE, as in Example 8-6.

Example 8-6 LIST.RECON STATUS after redefining the VSAM cluster for RECON1

-DDNAME-	-STATUS-	-DATA SET NAME-
RECON2	COPY1	IMS910G.RECON2
RECON3	COPY2	IMS910G.RECON3
RECON1	SPARE	IMS910G.RECON1

7. Issue the **CHANGE.RECON REPLACE(RECON2)** command to discard RECON2. RECON3 is copied to RECON1 and RECON2 is discarded.
8. Now you need to delete and define the VSAM cluster for the discarded RECON2. Before you can delete the cluster, you must ensure there are no other users using the data set. See step 5 on page 163 for details.
9. Delete and define the discarded RECON2 by using the appropriate parameters from “Defining the RECON data sets” on page 159. RECON2 becomes the new spare.
10. Issue the **CHANGE.RECON REPLACE(RECON3)** command to discard RECON2. RECON3 is copied to RECON1 and RECON2 is discarded.
11. Now you need to delete and define the VSAM cluster for the discarded RECON3. Before you can delete the cluster, you must ensure that there are no other users using the data set. See step 5 on page 163 for details.
12. Delete and define the discarded RECON3 by using the appropriate attributes from “Defining the RECON data sets” on page 159. RECON3 becomes the new spare. All of the VSAM clusters have now been redefined, and RECON1 is now the COPY1 and RECON2 is COPY2 as they were at the starting point.

You can automate this process by putting all the previous steps in the same job, but you need to schedule the job at a time when there are no other subsystems using the RECON. You must also ensure that the discarded RECON is released by all the sharing online systems. With only one online subsystem and no other users (batch jobs or utilities), you can perform the **/DIS OLDS** command by executing an automated operator BMP. With multiple systems sharing the RECON, you must propagate the command to the other systems. You can set up the automatic RECON loss notification feature for this purpose.

8.3 SMF and RMF considerations

For SMF, you should check the SMFPRMxx member of SYS1.PARMLIB. A parameter called DDCONS should be specified, and it should be set to DDCONS(NO). The default, if not specified, is YES. Setting this to NO does not affect any ability to gather data. Specifying YES or allowing DDCONS to default causes the system to consolidate DD information at every interval, which is also specified in this parmlib member. This can be a CPU intensive task for address spaces with a lot of open data sets, such as IMS and DB2, and can cause delays on a cyclical basis.

The RMF VSTOR parameter can also cause cyclical delays when the detail option is chosen for address spaces with large amounts of storage, such as IMS. The detail option can be turned on and off by command if you want this information, but you should not leave it on permanently or specify detail in the RMF start parameters.

8.4 Security considerations

IMS uses the SAF interface to check security. The product performing the security check can be RACF or another security product. IMS does not support Security Maintenance utility (SMU) after IMS Version 9. For this reason, security functions that required the use of SMU can now be performed using RACF, RAS, and security exits.

Table 8-1 provides the various resource classes that need to be managed by RACF. If the resource class does not exist in RACF, it needs to be defined by your RACF administrator. Resource classes are defined in IMS using the RCLASS= keyword in the SECURITY macro during IMS system generation. For a detailed description of the implementation of these various options, consult the *IMS Version 9: Administration Guide: System*, SC18-7807.

In conjunction with the resource classes, there are startup and system generation parameters that need to be considered. Most of these parameters can be changed in the DFSPBxxx startup member in IMS. Some parameters can also be changed using IMS commands. The *IMS Version 9: Administration Guide: System*, SC18-7807, provides a list of these parameters.

Table 8-1 Resource classes used by RACF

Description	Class Name	DB/DC, DCCTL, or DBCTL
APPC/IMS	APPCTP, APPCLU, and APPCPORT	DB/DC and DCCTL
Application group name	AIMS	DB/DC, DBCTL, and DCCTL
Command resource class	CIMS	DB/DC and DCCTL
Command group resource class	DIMS	DB/DC and DCCTL

Description	Class Name	DB/DC, DCCTL, or DBCTL
Database resource class	DIMS	DB/DC and DCCTL
Database group resource class	QIMS	DB/DC and DCCTL
Field resource class	FIMS	DB/DC and DCCTL
Field group resource class	HIMS	DB/DC and DCCTL
LTERM resource class	LIMS	DB/DC and DCCTL
LTERM group resource class	MIMS	DB/DC and DCCTL
Other resource class	OIMS	DB/DC and DCCTL
Other group resource class	WIMS	DB/DC and DCCTL
PSB resource class	IIMS	DB/DC, DBCTL, and DCCTL
PSB group resource class	JIMS	DB/DC, DBCTL, and DCCTL
Segment resource class	SIMS	DB/DC and DCCTL
Segment group resource class	UIMS	DB/DC and DCCTL
Transaction resource class	TIMS	DB/DC and DCCTL
Transaction group resource class	GIMS	DB/DC and DCCTL

Consideration must be given to the requirements for setting up the RAS interface. There are many RACF classes that need to be reviewed in conjunction with your current SMU requirements.

The following guidelines are presented with RACF in mind but conceptually apply to any security environment:

- ▶ Specify multiple RACF TCBs in IMS by using the RCFTCB= startup parameter. The default is one, but you should consider two or more depending on how much signon and signoff activity you have.
- ▶ Split the RACF database into multiple data sets to take advantage of the multiple TCBs in IMS.
- ▶ Cache ACEEs in VLF. This is accomplished by specifying the IRRACEE class to VLF and causes the ACEEs to be kept in VLF, once they are initially loaded, for subsequent access.
- ▶ Cache group trees similar to the above but by using the IRRGTS class in VLF.
- ▶ Use CF structures for the RACF database. This can reduce RACF I/O from milliseconds to microseconds.
- ▶ Specify APPCSE=CHECK or OTMASE=CHECK instead of FULL if possible. This reduces the number of ACEEs which need to be created.
- ▶ Investigate the use of the DFSBSEX0 exit routine. It might be possible to reduce the number of calls to security in certain cases, such as CHNG or AUTH calls, as well as other situations.
- ▶ Activating many RACF resource classes results in overhead, because the path length compared to SMU would be a lot more. This overhead must be measured and the business benefit understood before implementation.

8.5 Batch application performance

Batch application program performance should not be ignored. The EXEC statement has several performance-oriented options as discussed here.

8.5.1 Using DLI or DBB

The default batch application region type for most applications is DLI. This makes IMS build the application control blocks, for every execution, from the PSB and related DBDs, thereby consuming CPU and I/O resource every time. DLI is the *wrong choice* for production batch applications but is entirely suitable for application development and testing.

DBB makes IMS use the pre-built ACBs (see the ACBGEN utility). Using the pre-built ACBs saves CPU and I/O resources on every batch execution. Using the pre-built ACBs is also advantageous for avoiding errors that are injected by application developers and testers incorrectly updating the production PSB and DBD libraries when they meant to work on a testing set. Here are our recommendations for the batch parameters:

► PSB=

The default for PROCOPT= and even frequently chosen by casual developers, is ALL (PROCOPT=A). Although this choice often has little impact even when the program only reads the database, it does begin to matter in sharing environments, whether in a single online system or in Parallel Sysplex data sharing. The locks taken by PROCOPT=A are more restrictive than those for PROCOPT=Gx. The recommendation is to always specify the lowest possible PROCOPT for all programs so that they are better behaved as the installation moves toward data sharing.

► BUF=7

Sets the default OSAM buffer pool. This should always be overridden by specifying all the database buffer pools through DFSVSAMP DD.

► EXCPVR=0

Prevents page fixing of the OSAM buffer pool. This is the correct choice these days.

► PRLD=

Selects execution modules to be preloaded. Of limited value in batch environments.

► SPIE=0

This value allows the application SPIE to remain in effect during a database call. This is the correct value for a production application, because all bugs should have been eliminated, and therefore, '0' eliminates CPU involved in saving and resetting the SPIE around the call.

► SRCH=0

Tells IMS to use the standard library search when looking for modules. It is unlikely that batch applications reside in LPA or JPA, so this is the correct value.

► SWAP=

Makes address space swappable (Y) or nonswappable (N). The default is Y.

If you are using DBRC, you might want to choose option N because of the possibility of swapping out while holding the reserve on RECON.

If you are using IRLM for block-level data sharing, the SWAP parameter has no effect. If you specify IRLM=Y, a batch job is always nonswappable to prevent the job from obtaining locks for resources and then being swapped out.

- ▶ TEST=0

Another testing aid that should not be in effect for a production batch job. '0' is the correct choice.

- ▶ DFSVSAMP DD

This data set of control statements (could be a member of a PDS) should always be specified to set up the appropriate mix of database buffer subpools for the application.

Setup effort can be saved by using a few standardized pool setups for multiple applications. Buffer pool setup is described elsewhere in this book.

8.6 Utility performance

The standard IMS utilities are compatible with their setup in the earliest IMS releases (such as IMS/360 V1 in 1968). This means that, as they ran in a storage constrained world, they used the minimum number of data set buffers by default, usually two. Where more virtual storage is available, as is the case today, the number of data set buffers can be significantly increased to the elapsed time benefit of the utility executions.

All utilities should have the buffer number overwritten through the DD statement, allocating maybe hundreds of buffers. The various IBM and vendor go-faster products exploit the increased available storage automatically without requiring much in the way of definition.

Utilities, such as Change Accumulation, that invoke sorts internally benefit from increasing the amount of sort storage available. The defaults are far too small. Different utilities have different ways of specifying the storage to use.

Application considerations

This chapter discusses IMS application considerations and talks about designing applications in an IMS environment. It covers aspects related to performance and the design of applications.

This chapter contains the following:

- ▶ IMS application interface where we discuss the language interface and what you must do to invoke the interface
- ▶ Structure of a typical program using the language interface
- ▶ Application considerations for performance when looking at writing applications that operate in a IMS environment
- ▶ IMS and language environment

This chapter assumes that you have programming experience and that you have coded IMS programs.

9.1 IMS language interface

IMS supports application programs written in Assembler, COBOL, Enterprise COBOL for z/OS, PL/1 for z/OS, z/OS C/C++, Pascal, REXX, and Java™. Java programming considerations are discussed in Chapter 12, “IMS On Demand performance” on page 211.

In order to protect IMS and shield the developer from internal complexities, IMS makes use of an application programming interface called the *IMS language interface*. The language interface consists of both the language specific interface and the non-language specific interface. The inherent difference is that the former is language dependent and therefore only supports the language in question. The latter is language independent and therefore can be called from any of the supported languages in IMS.

For simplicity, we use COBOL as the preferred programming language and show examples of the language dependent interface only.

Figure 9-1 shows the structure of an application program as it relates to IMS and the language interface. A simple call to CBLTDLI with the required parameters ensures that IMS can obtain the data or message that is required by the application for processing. In its simplest form, we can either retrieve or send a message, or alternatively, retrieve and send data to a database in IMS.

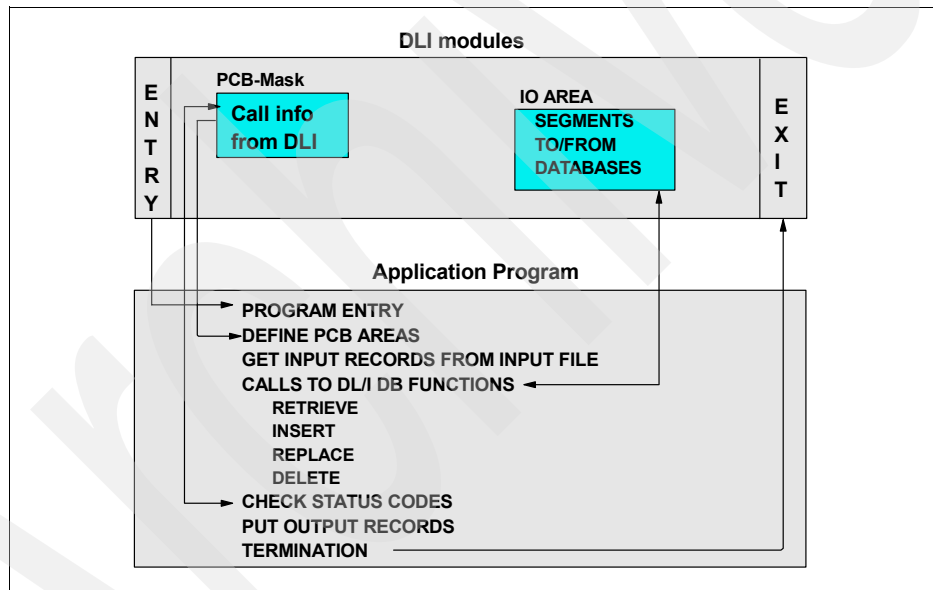


Figure 9-1 Structure of an application program

9.1.1 Structure of a typical program using the language interface

The structure of any IMS program must take into account a number of issues:

- ▶ The hierarchy of all the databases that the application uses, which is called the *DL/I hierarchy*. You need to understand the root and dependent segment relationships in detail.
- ▶ IMS separates the program from the physical characteristics of the database. This is possible by providing the program access to the program specification block (PSB). The program does not need to concern itself with the physical characteristics of the database description (DBD).

- ▶ The application program can only access the database using its program communication block (PCB) as defined in the PSB. The application therefore only has the view of the data as defined in the PCB.
- ▶ The call to the language interface is actually using the alias CBLTDLI. At linkage time, the linkage editor resolves the reference to DFSLI000.

Essentially, an IMS COBOL program includes the following processes:

- ▶ Program entry showing all the various sections of the program
- ▶ PCB definitions as per the Procedure Division mapping to the PSB
- ▶ I/O area definitions as required to retrieve and send messages and database segments
- ▶ DL/I calls as required by IMS
- ▶ SSA definitions as required by the database hierarchy
- ▶ Call to the language interface using CALL CBLTDLI
- ▶ Termination of program

9.2 Performance and programming considerations

Accessing data from a hierarchic database requires that we understand certain basic rules around Segment Search Arguments (SSAs). An SSA provides the application program with the ability to access data as specified by the hierarchy. Using the correct DL/I retrieval call makes the retrieval process a lot more efficient. Consider the following:

- ▶ If you need to process all segments sequentially in ascending sequence, within a database, a database record, or a specific segment type, use unqualified SSAs.
- ▶ If you need to process segments in a skip sequential order in ascending sequence, always fully qualify the root segment.
- ▶ If you need to process database records directly, as in a random order by online transactions, use a qualified SSA.
- ▶ We recommend that you use command codes to eliminate the overhead of doing multiple DL/I calls.
- ▶ We recommend using programming standards for efficiency and reusability. Issues pertaining to standards are as follows:
 - Use the COPY facility in COBOL to copy any predefined areas that are shareable across multiple programs. This prevents duplicating code. Function codes, segment descriptions, SSAs, and PCB masks are some of the common copybooks that you require.
 - Keep your I/O routines separate so that they can be easily changed if required. Checking status codes should be performed by a subroutine. This allows any new status codes that are introduced to be easily adapted.
 - Use a modular approach to design your application. Keep generic functions as subroutines that can be called by any program wanting to make use of this service.
 - With batch processing, try to perform all your business requirements through a single pass of your data. If you require data for input into another process, create flat files with the required data rather than reading the database again.

9.2.1 SSA considerations

SSAs can be either qualified or unqualified. A qualified SSA identifies a specific occurrence of a segment. With a unqualified SSA, you identify only the segment type. When incorporated with command codes, we end up reducing the number of DL/I calls required to retrieve a

segment or occurrences of a specific segment type. Assume we have a hierarchy as depicted in Figure 9-2.

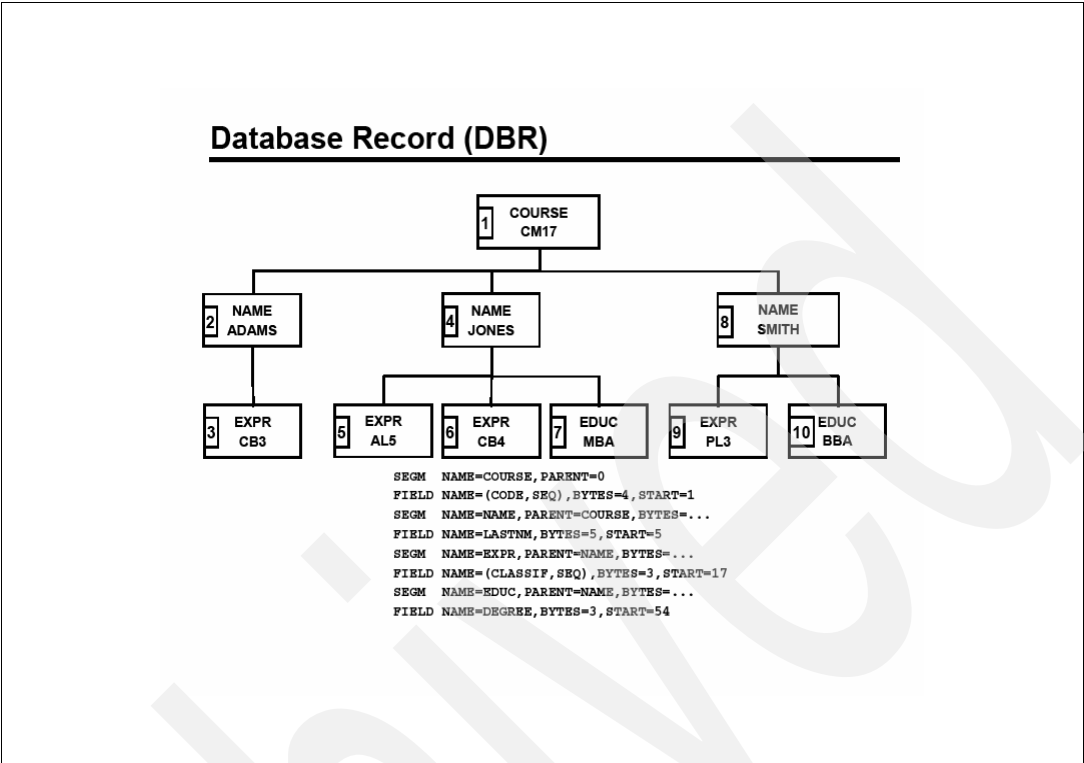


Figure 9-2 Sample hierarchy

We compare the differences between standard DL/I calls and those using command codes. For purposes of illustration, we only use the F command code. The input data consists of names. Assume we would like to produce a listing of all names that match the input data. The program should check each dependent EDUC segment for the given name and if an MBA degree is detected for the person, return to the first EDUC segment and list all his degrees. Table 9-1 gives us the calls that are required.

Table 9-1 Comparison of standard DL/I call as opposed to command code

Standard DL/I call logic	Using command codes
GUbb NAMEbbbb(LASTNMbbb=ADAMS)	GUbb NAMEbbbb(LASTNMbbb=ADAMS)
GNPb EDUCbbbb(DEGREEbbb=MBA)	GNPb EDUCbbbb(DEGREEbbb=MBA)
GUbb NAMEbbbb(LASTNMbbb=ADAMS)	GNPb EDUCbbbb*Fb
GNPb EDUCbbbb	GNPb EDUCbbbb

The first solution without command codes is to issue a GU call to the NAME segment qualified by the input data. A GNP call to the EDUC segment qualified on degree of MBA initiates a search of all dependent segments for a segment that meets that qualification. If none is found, the next input record can be read. If the person has an MBA, a GU call for the NAME qualified again by the input data backs up the database position to the root segment. A series of GNP calls specifying the EDUC segment retrieve all those dependent segments, which then are listed.

The second solution utilizes the “F” command code after a hit has occurred on the degree of MBA. For GN or GNP calls, the “F” command code allows backing up to the first occurrence of this segment type within a database record. Therefore, unlike solution one, a GU call is not

required to reestablish position at the root level for subsequent GN type calls, thus, saving an additional call and simplifying the programming.

Figure 9-3 shows the various command codes and which DL/I call type is suited for the command code. If a command code is used with a function for which it is not suitable, then it is ignored. There are no status codes for invalid combinations of command codes. IMS assumes certain defaults. If two command codes are in direct conflict, the last one within the set is effective. However, "F" or "L" override "U" or "V" in the same SSA.

Command Codes by Function						
Command Code	GU GHU	GN GHN	GNP GHNP	DLET	REPL	ISRT
C	Y	Y	Y	N	N	Y
D	Y	Y	Y	N	N	Y
F	Y	Y	Y	N	N	Y
L	Y	Y	Y	N	N	Y
N	N	N	N	N	Y	N
P	Y	Y	Y	N	N	N
U	Y	Y	Y	N	N	Y
V	Y	Y	Y	N	N	Y

◆ There is no status code for invalid combinations

◆ "F" or "L" will override "U" or "V" in the same SSA

Figure 9-3 Command codes by DL/I call type

9.2.2 Single as opposed to multiple positioning

When single positioning (the default) is specified for a PCB, DL/I maintains only one position in that database for that PCB. This position is used to attempt to satisfy all subsequent GN calls. If an application requires parallel processing of segment types, a better solution is to use multiple positioning. Multiple positioning is specified in the PSB as shown in Example 9-1.

Example 9-1 Single and multiple positioning

PCB...	POS=S
PCB...	POS=M

The use of multiple positioning allows you greater independence for dependent segments. It allows the programmer to use GN or GNP calls without regard to the relative order of segment types within the DBD. If the relative order of segment types was reversed and multiple positioning was specified, the change would have no impact to application programs using multiple positioning. However, multiple positioning requires the programmer to keep track of all positions maintained by DL/I; therefore, this approach requires additional coding. In any case, the program coding between single and multiple positioning is different. See Figure 9-4 on page 174.

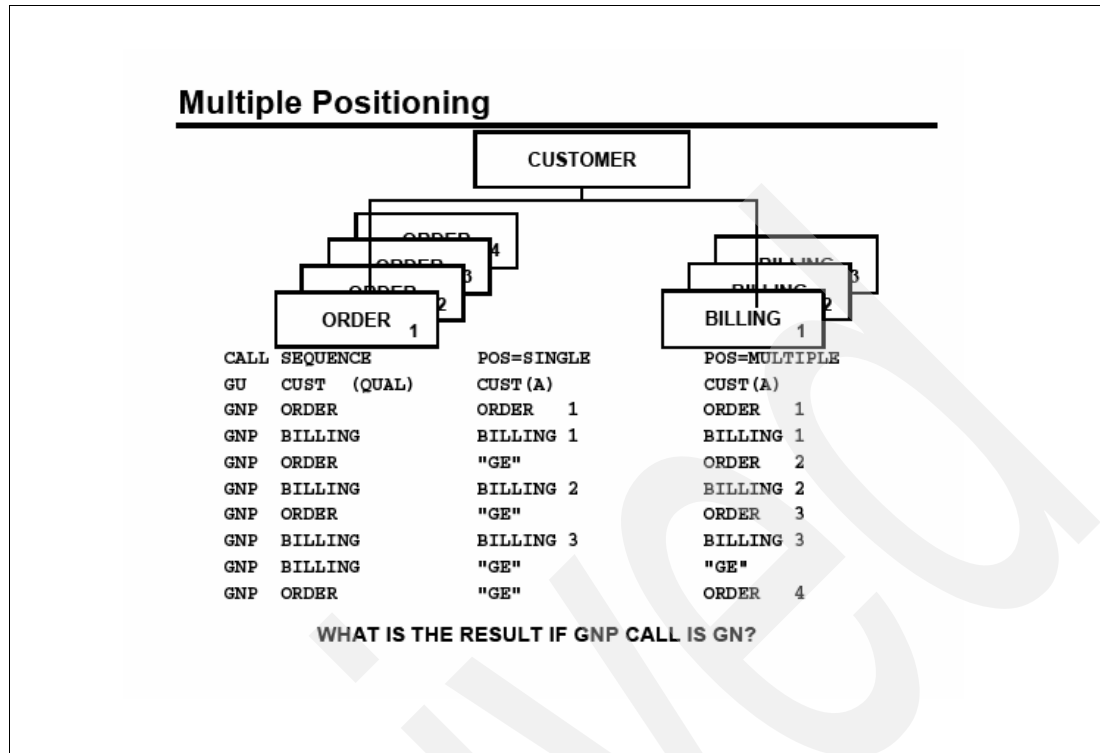


Figure 9-4 Differences between single and multiple positioning

Figure 9-4 shows the differences between a GNP call using single and multiple positioning. A "GE" status code is returned for single positioning when a GNP call is made to retrieve the second ORDER segment, because a GNP call can only move forward within the database. With multiple positioning, a position is maintained at each hierarchical leg within the database record until all segment occurrences are exhausted.

If we now replace the GNP call with a GN call, for single positioning the second call to the ORDER segment retrieves the first ORDER segment of the next root segment, the third call the next root segment, and so on. Each call moves forward in the database. With multiple positioning, the results of a GN call would be the same as the results of the GNP calls, except that the "GE" status on the next to last call retrieves the first ORDER segment of the next root segment.

There are also restrictions when using multiple positioning and a mixture of qualified and unqualified SSAs. We recommend that the programmer does not mix qualified and unqualified calls with multiple positioning, because the results could be unpredictable.

9.2.3 Variable length segments

You must make many decisions during the application design and the database design. Variable length segments must be considered when the size of the segment varies dependent on the data. Variable length segments save space in the database, and the programmer is responsible for ensuring that the size is defined correctly as part of the segment data to be inserted. The minimum and maximum sizes are specified in the DBD with the BYTES=(max,min) parameter. The inserted size can never exceed the max size. If it does, a "V1" status code is returned to the application.

The key field of a variable length segment type must be contained within the minimum size as defined during DBD generation. Other non-key search fields that can be used in SSAs might

appear anywhere within the segment. If a search is initiated on a non-key field and the field does not appear within a specific segment occurrence, a blank or zero is assumed for that data depending on how the data was originally described.

When used correctly, variable length segments can improve the efficiency of the database. However, they do involve overhead, especially if they are updated frequently. If you are using variable length segments, you should ensure that their length does not change frequently.

9.2.4 Secondary indexing

Secondary indexing (SI) was designed to provide two basic facilities:

- ▶ To provide processing of a database in sequences other than the prime root key sequence.
- ▶ To provide a direct search capability into a database record other than key field of the root.

Secondary indexes are automatically maintained by DL/I, when the program inserts, replaces, or deletes data within a database.

Some considerations when using secondary indexes:

- ▶ Secondary indexing is a database structuring technique and normally concerns the database administrator. It should be transparent to the majority of the application programmers. However, because the same logical structure can yield two different results in the PCB key feedback area, the programmer should be aware of these differences.
- ▶ The index target segment is the segment in the prime database pointed to by the SI segment. It is the segment that DL/I selects to satisfy the call made by the application.
- ▶ The XDFLD is a special field within the DBD that contains the name of the key field data of the SI. The XDFLD name can be used in SSAs for the target segment. When the SI is used, the XDFLD is considered the key field of the target segment, and thus its value is placed in the key feedback area on any access to that segment.
- ▶ The pointer segment is the segment in the secondary index database that allows the SI to be used as a database.

9.2.5 Program reusability considerations

When coding an IMS COBOL program, the manner in which you set up and initialize working storage plays an important role in the eventual performance of the application. A program is truly reusable if it is capable of executing multiple transactions through a single copy of the program. In an IMS world, online transactions execute in a dependent region. If a copy of the program executes for the first time in a dependent region, the programmer must ensure that the logic of the program is not compromised after subsequent iterations of the same transaction in the same dependent region. In other words, working storage areas should be initialized prior to retrieving a message.

9.2.6 Processing options and the PROCOPT statement

The PROCOPT parameter can be coded on the PCB and the SENSEG statement. If the PROCOPT parameter is not coded on the PCB statement, then it defaults to PROCOPT=A (all). When the PROCOPT is coded on the SENSEG statement, it overrides what is specified on the PCB statement, assuming they are compatible. IMS always locks a database record based on the PROCOPT specified on the PCB statement. In Example 9-2 on page 176, the PCB PROCOPT defaults to A.

PCB...

SENSEG...PROCOPT=GO

When a program accesses the SENSEG, the GO applies, because the SENSEG overrides the PCB when they are compatible (GO is a subset of A). However, IMS locks on the PCB statement. Therefore, if the program was read only, IMS would still use the PROCOPT=A default on the PCB to lock the entire database record. This affects the performance of a read only program. So, a PROCOPT must always be coded on the PCB statement.

9.2.7 Read only programs

For those programs that are read only on a database, two options can be specified. The first is read with integrity. This means that the read only program must retrieve data that is 100% up-to-date. If this is the case, then PROCOPT=G should be specified. The second option is read *without* integrity. This means that the read only program needs information that is not necessarily 100% up to date. If this is the case, then PROCOPT=GO should be specified. This tells IMS not to enqueue on the database record. As a result, the read only program might run concurrently with an update program and not cause database record lockouts.

Note: From a business perspective, supplying information based on aggregating data (such as number of employees) with PROCOPT=GO calls is usually acceptable, because minor errors are probably not significant. Supplying information based on a single record or a small number of possibly erroneous records by using PROCOPT=GO can compromise business integrity. Use *PROCOPT=GO with care*.

However, a PROCOPT=GO call might cause a program to ABEND with a U0853, because of IMS pointers that are not 100% up-to-date. Therefore, use PROCOPT=GON for read only jobs that do not require 100% integrity (see 9.2.8, “PROCOPT=GOT with DBRC SHARECTL” on page 176 for more information about GOT). When GOT is used and IMS encounters a bad pointer, IMS reads the segment a second time. If the pointer is still bad on the second read, then IMS returns a “GG” status code to the program. The program can then decide whether to issue the read again, to ABEND, or bypass that segment. Using GOx reduces the number of U085x ABENDs.

9.2.8 PROCOPT=GOT with DBRC SHARECTL

We strongly recommend that you use DBRC SHARECTL and register all databases in a production environment. However, be aware that IMS performance degradation can occur when using PROCOPT=GOT and DBRC SHARECTL.

This degradation applies to all users and might well apply to databases other than the one accessed with PROCOPT=GOT. This degradation applies to IMS DB/TM, DBCTL, CICS, and batch sharing environments.

If an invalid pointer is detected when using PROCOPT=GON or GOT, then processing is retried. With PROCOPT=GON, retry processing simply retries the call and returns a GG status code if the problem is still present.

The retry processing for PROCOPT=GOT consists of two actions:

- If a locking environment exists, a test lock on the database record containing the segment being retrieved is requested. (A locking environment exists for an IMS DB/TM, IMS DBCTL, or CICS system and also for IMS batch jobs if block-level data sharing is used.)

The test lock causes call processing to wait until conflicting holders of locks on the database record release their locks.

- The last block read for the call is reread.

This is an attempt to get the latest image of the block.

In many cases, this retry succeeds in correcting the problem that produced the original invalid pointer condition. However, it can also cause serious performance degradation in the IMS subsystem. The degradation might affect databases for which PROCOPT=GOT is not used.

As well as causing a reread of the last block, buffer invalidation takes place. This causes all the buffers for all databases that are registered at a share level greater than 0 (1, 2, or 3) to be invalidated. The lookaside process does not find any blocks in the pool for these databases until they are reread from DASD. OSAM sequential buffering buffers are invalidated, as well as other buffers.

If the use of PROCOPT=GOT causes unacceptable performance degradation, consider changing PROCOPT=GOT to PROCOPT=GON.

We recommend that you use PROCOPT=GOT only if contention with updaters, though possible, is highly unlikely.

9.2.9 Use of checkpointing in batch

Batch workload is extremely intensive and generally processes large amounts of data. When compared to an online transaction, a batch workload can be looked at as processing multiple messages in a single batch job. The problem is how to manage batch differently than online and at the same time allow both batch and online to coexist using the same limited resources. The second issue is how do we restart a batch job after it has run for a few hours. We certainly do not want to restore and rerun the batch job. IMS checkpointing and restart address these problems.

There are two types of checkpointing in IMS, namely:

- Basic checkpointing
 - With basic checkpointing, the restart and repositioning become the responsibility of the application.
- Symbolic checkpointing
 - With symbolic checkpointing, IMS restores the application work areas specified in the XRST call and performs the repositioning. The application then restarts processing from the last successful checkpoint.

There are performance considerations when looking at checkpointing in a batch process. We discuss a few critical issues around the design of batch processes:

- A checkpointing frequency plays an important role, especially if you have a mixture of online and batch workload that runs concurrently. We recommend that you make the checkpointing frequency an externally controlled variable either through a control data set or by using the IMS APARM capability on the EXEC statement. An optimum balance must be found and is usually a factor of how much work is performed by your batch process. Assume that the driver to the batch process is an input file with two million records. Each record requires complex processing across multiple databases. This unit-of-work per record needs to be evaluated in terms of its profile. When you understand the profile and how long it takes to process a single record, you can begin to work out the checkpoint frequency. We recommend that your checkpoint frequency reflects the type of work that is being performed. Too few checkpoints end up holding locks and might cause online

transactions to wait until the locks have been released. Too many checkpoints result in long batch run times, because the batch job spends most of its time checkpointing. A balance between batch and online must be found. There is no single good general rule for checkpointing, because it is a direct correlation to the complexity of the batch process.

- ▶ If your batch process updates Fast Path Data Entry Databases (DEDDB), you have to ensure that your application logic revolves around the use of Fast Path buffers, namely Normal Buffer Allocation (NBA) and Overflow Buffer Allocation (OBA). When updating records in a Fast Path database, your JCL normally overrides the NBA and OBA buffer allocation. Once you have used up all your NBA buffers, IMS returns a “FW” status code to your application. On receipt of the status code, you have to set an indicator in your application to checkpoint at the next opportunity. Do not continue processing, because you end up with a “FR” status code, resulting in your batch job abending. When deciding on fair NBA and OBA buffer allocations, as a general rule, try to keep your NBA buffers to a reasonable size that allows you to checkpoint often. Remember that the NBA and OBA allocations come out of your total Fast Path pool defined at IMS startup time. A high NBA value might cause other jobs or online transactions to abend if you are running a number of batch jobs and online transactions concurrently.

9.2.10 Multi-streaming your batch processes

If your environment performs all its major consolidation of financial transactions at night using a batch process, then you most likely have the problem of trying to complete your batch before your next online day. To complicate matters further, if you need your online system to be up 24x7x365, then you are faced with a bigger problem in that you must ensure that your batch processes do not conflict with your online processes.

A common approach is to ensure that you run your batch workload in parallel through multiple partitions or areas. If you use flat files as drivers for your batch process, then ensure that you sort and split the files in the sequence of the database. This means that if you are PHIDAM, then you would sort and split the files using the Partition Selection Exit or high keys as specified in the RECON in ascending sequence. If you are using DEDDBs, then you would sort and split the files in randomizer sequence. The sorts and splits can be easily achieved through specialized E15 and E35 exits using ICEMAN utility or alternatively, using ICETOOL.

Our recommendation is that if your batch processes are intensive and time is of the essence, use OSAM sequential buffering for PHDAM (with a sequential randomizer) or use PHIDAM, to sequentially process through the database, even if you are not updating each record. PHDAM (without a sequential randomizer) benefits from OSAM sequential buffering if the processing is consecutive (in relative byte address order) as with GN.

9.2.11 Why must online programs be serially reusable

When IMS receives a message for an message processing program (MPP) on the input queue to process, it goes through several steps to process the request. Consider each run of a program or transaction as a Message request to IMS. To keep this discussion in context, the program in question has not yet been processed today. The first thing that happens is IMS determines if there is a message region available for this program in which to execute. If there is only one message region available and there are two messages eligible to execute in that message region, IMS processes as follows (in order). IMS looks at the current priority of the two messages and picks the one with the highest priority. Next, IMS loads in the program load module. IMS loads in the application control block (ACB). The ACB is a marriage between the program specification block (PSB) and the database definition (DBD). This all takes time and impacts the performance of the transaction.

Once the program and ACB are loaded, IMS then passes the message to the program and passes it control. When the program is finished processing and sends a message back to the user, control is given back to IMS. IMS does not want to do any more work than necessary; it looks on the input queue to see if there are any different transactions with an equal or higher priority to process. If there are, IMS flushes the ACB and program from memory and proceeds with the new transaction. However, if there are no different transactions with an equal or higher priority to process, and there are messages on the input queue for this same transaction to process, IMS uses the same load of the program and ACB to process the next message on the input queue. This is why the program has to reinitialize all of working storage before it does a get unique (GU) to the input/output Program Communication Block (I/O PCB) of the PSB. IMS continues to do this until the processing limit count of the transaction is reached. Once the processing limit count is reached, IMS looks to see if it needs the message region for some other transaction. If it does, the message region gets flushed and the new transaction is processed. If not and there is more work for this transaction to process, IMS resets the processing limit count, cuts some accounting records, and continues to use the same program load and ACB. This is called a *quick reschedule*.

IMS continues to do this until it needs the message region or there are no messages for this transaction to work on. Now it gets more complex. If IMS does not need this message region to process other transactions and there are no more messages for the current transaction to process, IMS keeps everything in the message region. The program and ACB just sit there and wait for work to do. What IMS is hoping for is that another message for this program shows up on the input queue before IMS needs this message region for a different program. This is called *pseudo wait-for-input*.

There is also a parameter called the *parallel limit count*. This parameter determines how many messages must be on the input queue for a transaction before IMS schedules it in another message region. This allows multiple transactions of the same name to run concurrently. A value of zero sets no limit. Generally, the parallel limit count is set to 5. If the parallel limit count is set to 5, this means there must be five messages on the input queue before IMS schedules another message region. However, now that there are two message regions running the same program, there must be 10 messages on the input queue before IMS schedules another message region and so on. It is possible for a single transaction to occupy every message region available. There is a parameter called the *maximum region count* to prevent this.

9.3 Language environment

The language environment (LE) provides a single run-time environment for high level languages. It consists of the following:

- ▶ Basic routines that support starting and stopping programs, allocating storage, communicating with programs in different languages, and handling conditions.
- ▶ Callable services to handle generic services, such as math and date/time services.
- ▶ Language-specific portions of the library that ensure that the behavior is consistent across languages. Languages supported are:
 - C/C++
 - Cobol
 - PL/I
 - Fortran (but not in IMS)
 - Assembler

The functional overview of the LE environment is shown in Figure 9-5 on page 180.

Functional Overview of Compilers and LE

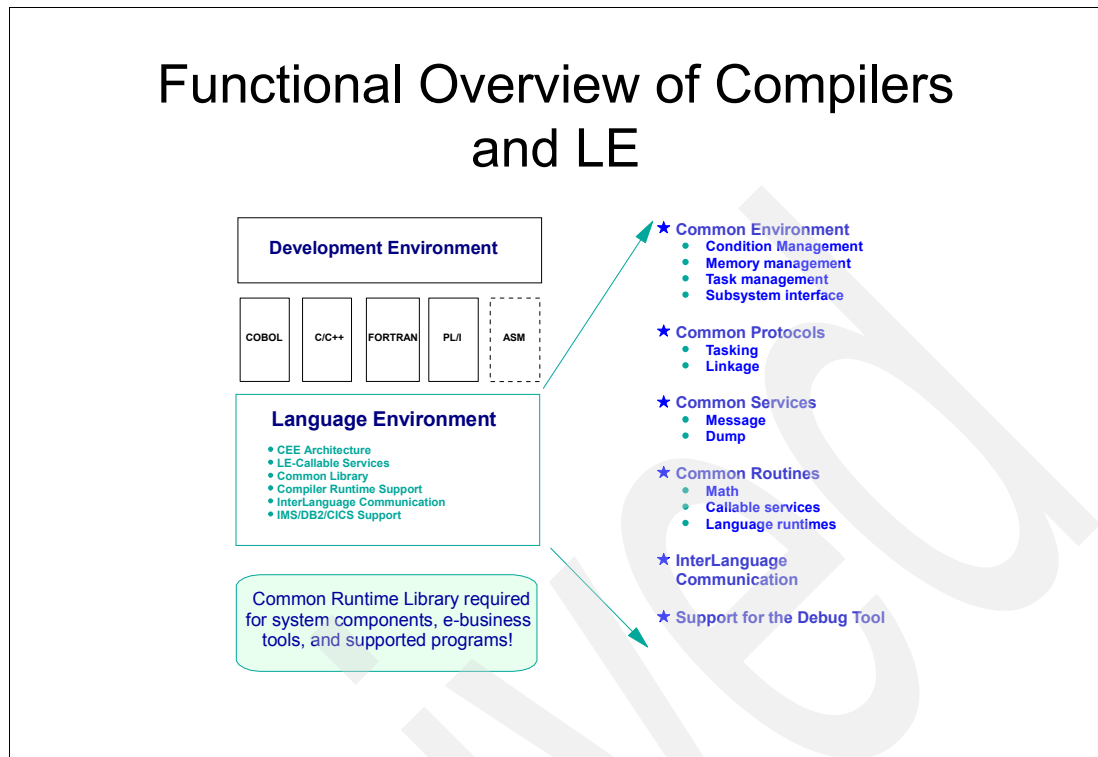


Figure 9-5 Language environment view

9.3.1 Application and performance considerations in a LE environment

Performance of an application in a LE environment is governed by many variables, some of which exist within LE and others are dependent on the application itself. Now we review several issues with regard to implementing LE in your environment:

- ▶ Review and understand exactly what is meant by migrating to LE. This involves understanding your applications and reviewing the requirements and differences between working storage and local storage and how uninitialized variables impact the logic in your program when converting to LE. The single important factor in your decision is the amount of work that would be required to change programs if your programs are not LE compliant. The end result is that the logic or program flow must not be impacted when moving to a LE environment.
- ▶ Investigate the usage of Library Retention Routine (LRR) in your environment, because it allows performance improvement for LE resources. Specify CEELRRIN in the DFSINTxx member in PROCLIB.
- ▶ Review run-time options related to HEAP and STACK storage. The initial stack segment should be large enough for all requests for stack storage. Stack storage is essentially used for routine linkage and acquiring dynamic storage areas for the application. Storage for data items declared in the COBOL LOCAL-STORAGE-SECTION is allocated using STACK storage. Heap storage on the other hand has a lifetime unrelated to the current execution of the current routine. It remains allocated until the enclave terminates. (An example is a WFI region being terminated). For best performance, the initial HEAP size must be large enough to satisfy all requests for HEAP storage. All working storage variables are obtained from HEAP storage; therefore, it is important to ensure that your working storage is initialized on each invocation. You need to review the following run-time options:
 - ANYHEAP, BELOWHEAP, HEAP, and THREADHEAP

- LIBSTACK, STACK, and THREADSTACK
- ALL31(ON)
- ▶ The defaults cause the z/OS operating system to issue multiple getmains for STACK and HEAP storage for the application and result in badly performing applications. Changes in the profile of a transaction, especially CPU, would be noticeable when looking at your tracking and trending reports of your workloads.
- ▶ With ALL31, you must ensure that all your applications are AMODE=31. If they are AMODE=24 programs that coexist with AMODE=31, then you have to use ALL31(OFF) and STACK(xx,xx,BELOW). We recommend that you try to ensure all programs are AMODE=31.
- ▶ Setting the run-time options would require that you customize the run-time options using either CEEDOPT (installation-wide defaults), CEEROPT (IMS region-wide default) or CEEUOPT (application specific options).

Extensive testing is of prime importance. Testing has to follow a strict methodology in order to identify any problems related to LE. A combination of both positive and negative testing must be done in order to detect possible errors in the logic of programs. Performance testing must also be performed to gauge the savings or overhead associated with LE.

Archived

Performance considerations with DB2

In DB2, there is a hierarchy. At the bottom of that hierarchy is a table. A DB2 table is like an IMS segment. Tables live in tablespaces (a physical data set), and there can be many tables in one tablespace. Tablespaces live in a DB2 database, and, hence, a hierarchy. There can be many DB2 databases defined to the DB2 region.

Several DB2 attachment facilities are provided for different environments:

- ▶ IMS External Subsystem Attach Facility (ESAF)
- ▶ IMS batch attach facility
- ▶ CICS attach facility
- ▶ TSO attach facility
- ▶ Call attach facility for any z/OS batch

The IMS ESAF interface provides access to external subsystems from any supported region type (MPP, BMP, IFP, JMP, or JBP) in any IMS system environment (DM/TM or DBCTL).

This chapter describes the IMS ESAF and various aspects of performance related to its use accessing DB2.

10.1 IMS External Subsystem Attach Facility

The IMS ESAF enables users to access DB2 from IMS DM/TM or DBCTL. Specifically for DB2, it provides support for:

- ▶ Connection between the IMS subsystem and one or more DB2 subsystems
- ▶ Communication between an IMS control region and DB2 for a command thread (for issuing DB2 commands in IMS)
- ▶ Communication between IMS dependent regions and DB2 for transaction threads (for enabling and processing SQL calls)
- ▶ Communication between IMS Java regions and DB2 for transaction threads (for enabling and processing SQL calls)
- ▶ The SQL application programming interface
- ▶ Coordinated sync point processing

IMS application programs using ESAF have concurrent access to the following resource managers:

- IMS message queue manager (online applications only)
- IMS Full Function database manager
- IMS Fast Path database manager (DBCTL BMPs or online applications only)
- DB2 database manager
- MQSeries® queue manager (local and remote queues)

IMS sync point processing uses a two-phase commit. IMS assumes the role of sync point coordinator¹ and communicates with all the relevant resource managers, including DB2. In case of a program or system failure, IMS coordinates the backout or recovery of both DB2 and IMS data (messages and databases). See the note under “LOCKSIZE parameter” on page 189 for another explanation.

IMS also supports read-only threads, which improve the DB2 read-only commit processing by eliminating the unnecessary Phase 2 call.

Figure 10-1 on page 185 shows the relationship between DB2 and IMS.

¹ For a CICS DBCTL transaction, CICS is the sync point coordinator. For a BMP in a DBCTL environment, IMS is the coordinator.

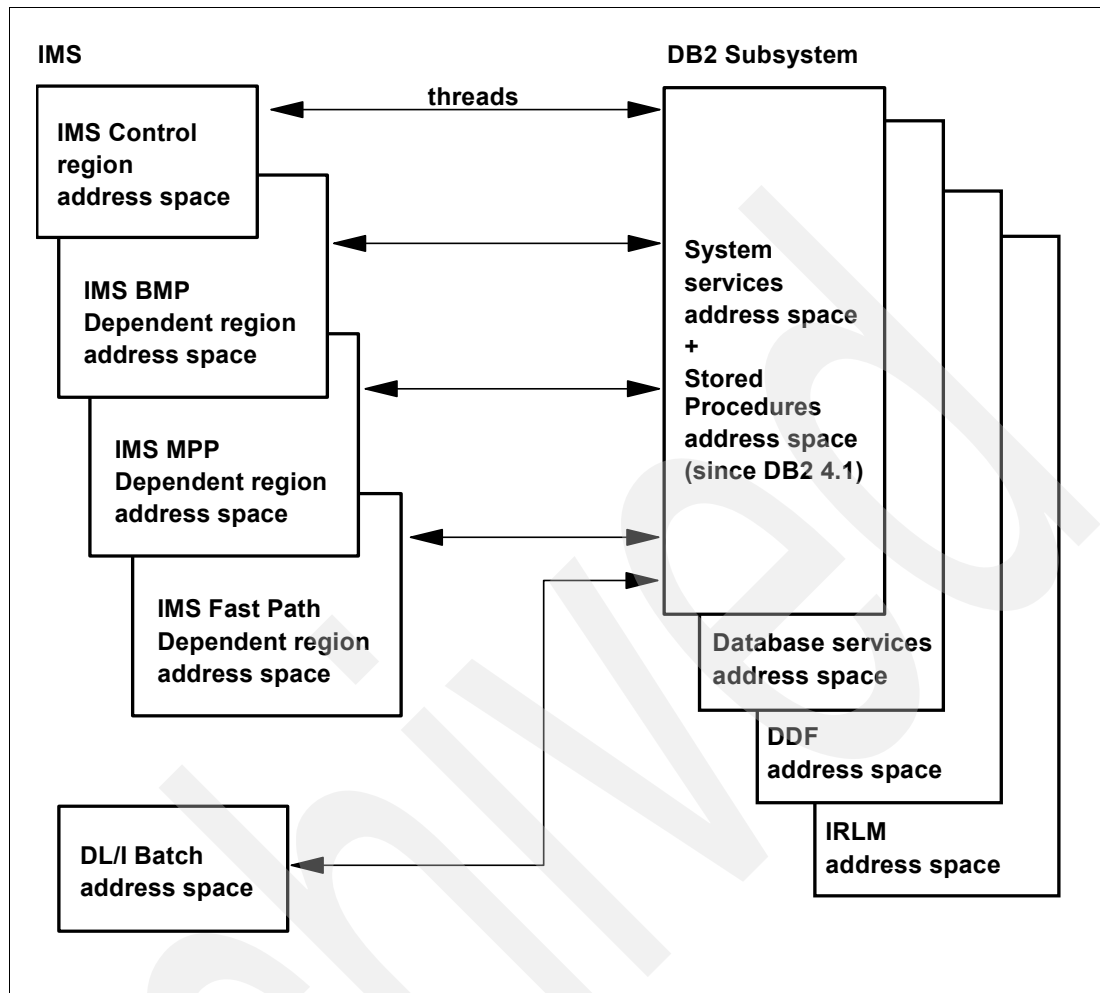


Figure 10-1 Relationship between DB2 and IMS

IFP, JMP, and MPP regions identify themselves to DB2 during the initialization phase of their startup. BMP and JBP regions wait until the first SQL call to identify themselves to DB2.

10.1.1 Subsystem member

The IMS subsystem member execution parameter defines which subsystem member (SSM) in IMS.PROCLIB is read by IMS when it starts up. The SSM defines which DB2 subsystems can communicate with this IMS. Each dependent region also has an SSM execution parameter, indicating which IMS.PROCLIB member defines the DB2 subsystems with which this region can communicate. This can be the same set or a subset of those defined for the control region.

The member used by the IMS control region is the default for the dependent regions, which do not have any SSM specification. If some regions have no need to access DB2, we recommend that you use an SSM that exists but is empty.

For example, in your IMS environment, you bring up your IMS system defined with the following parameters: IMSID=IMSF and SSM=DB28. In IMS.PROCLIB, there is an SSM member named IMSFDB28. This member is defined with the following values:

```
DB28,SYS1,DSNMN10,,R,+
```

These values correspond to the following positional parameters:

SSN,LIT,ESMT,RTT,RE0,CRC

Which have the following meanings:

- ▶ SSN: DB2 subsystem name
- ▶ LIT: Language interface token
- ▶ ESMT: External subsystem module table (DSNMIN10 is the required value)
- ▶ RTT: User-generated resource translation table (optional)
- ▶ RE0: Region error option
This option determines the action IMS takes when an application program issues a request for external subsystem services before connection to the external subsystem is complete, or if problems are encountered with the external subsystem:
 - R: The application program receives a return code indicating that the request for external subsystem services has failed.
 - Q: IMS abnormally ends (ABENDS) the application program with an ABEND code of U3051.
 - A: IMS abnormally terminates the application program with an ABEND code of U3047 and discards the transaction input if the application program has already issued the get unique (GU) call.
- ▶ CRC: Command recognition character, (+) in this case

Refer to “Defining Your External Subsystems to IMS” in *IMS Version 9 Installation Volume 2: System Definition and Tailoring*, GC18-7823, for further details. There is also useful information in the *DB2 Universal Database for z/OS Administration Guide*, SC18-7413.

Figure 10-2 on page 187 shows the relationship between the SSM and IMS regions (control and dependent).

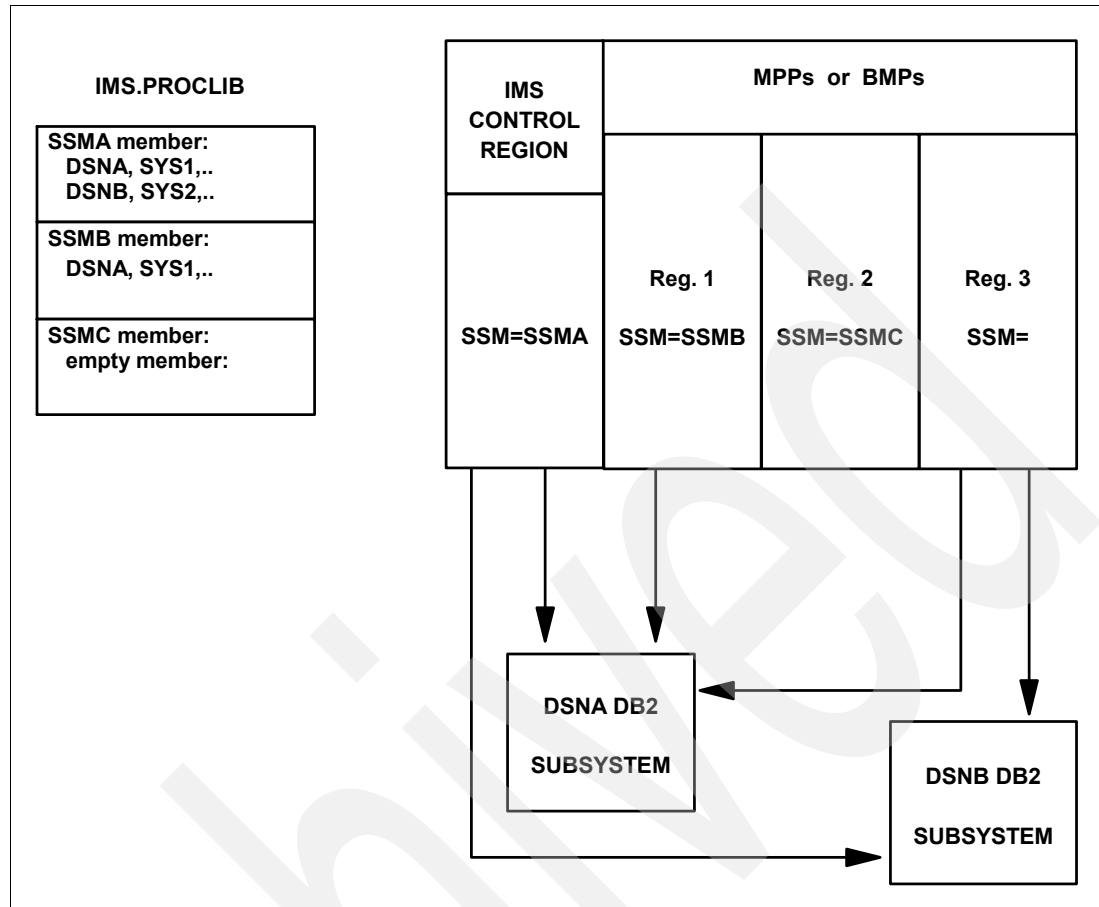


Figure 10-2 Relationship between SSMs and dependent regions

10.2 Tuning the External Subsystem Attach Facility

When using the ESAF, we recommend that you follow these guidelines:

- ▶ Minimize the number of thread allocations and terminations.
- ▶ Minimize the duration of locks.
- ▶ Avoid defining more external subsystems than needed in the SSM. Each definition adds storage requirements. Use an empty SSM for regions that do not require access to DB2, otherwise, the control region SSM is used.
- ▶ Consider isolating DL/I applications in IMS dependent regions separate from SQL applications. This facilitates:
 - Performance monitoring and problem isolation
 - Selection of scheduling algorithms and class assignments
- ▶ Analyze applications to choose between static SQL and dynamic SQL.
- ▶ Choose the best security level according to installation requirements.

10.2.1 Thread management

Thread creation and termination can use significant amounts of CPU in an IMS/DB2 transaction. To reduce the CPU cost of thread creation at program schedule time (and at signon to DB2), IMS uses a program call instead of an SVC call and makes use of cell pool (CP00L) requests to allocate dynamic storage.

The IMS Monitor, in the Region IWAIT report (Example 10-6 on page 196), shows the time of the thread creation and termination. It reports them as IWAITs, but they are not I/O IWAITs; although, the SQL normal call values do, of course, include I/Os.

The simpler the transaction, the proportionally higher the cost of thread creation and termination. Consequently, CPU time can be saved if the number of thread allocations and terminations is kept to a minimum. We recommend that you do the following in order to achieve this:

- ▶ Use a specific number of regions connected to the DB2 subsystem.
- ▶ Keep threads running as long as possible by exploiting:
 - Wait-for-input transactions
 - Pseudo-wait-for-input
 - Quick reschedule

Fast Path (IFP) regions, which are wait-for-input, create the thread only once.

We recommend that you design for thread reuse as much as possible to spread the cost of thread creation and termination over multiple transactions. Collapsing multiple MPPs into a single MPP, if practical, can help.

Note: The thread create and terminate processes are executed under the caller's TCB, and the CPU time is charged to the IMS dependent region.

Wait-for-Input

The IMS Wait-for-Input (WFI) function is the ideal solution for minimizing thread allocations, because it happens just once when the program is scheduled. However, a region occupancy of more than approximately 75% is required to fully justify its use.

Pseudo-Wait-for-Input

Pseudo-WFI (PWFI) applies to all message processing programs (MPPs), and IMS/DB2 applications can benefit not only from reduced program load but from reduced DB2 thread creation as well.

Quick reschedule

Although this is not an option from an IMS perspective, there are two requirements for it:

- ▶ The application program must issue a GU call against the I/O PCB after each message is processed, to drive synchronization point processing. It should terminate on a QC status code only.
- ▶ PROCLIM > 0 must be specified on the TRANSACT macro.

If the program is held, waiting on the I/O PCB GU, and another message arrives for that transaction, it can be immediately passed to the program, so that the thread to DB2 from the previous scheduling is still available.

10.2.2 DB2 lock management

Locking considerations are important for the subsystem's throughput and performance. Attention must be paid to when a lock is acquired, which level of lock is used, and when the lock is released.

Pages, tables, or tablespaces can be locked during an application process in the same manner as when using IRLM for IMS, and similar to when using PI. The page locks result from SQL calls and can be compared to the CI locks that result from IMS DL/I calls.

LOCKSIZE parameter

When creating a tablespace, you choose the granularity of locking by specifying the LOCKSIZE:

- ▶ LOCKSIZE(PAGE)

Page locking is the best strategy for concurrency in an online environment. In general, it is achieved by specifying LOCKSIZE(ANY). However, for WFI transactions, you should specify LOCKSIZE(PAGE) explicitly.

- ▶ LOCKSIZE(ANY)

This is the default granularity level and is the value that we recommend. It implies a page-level locking strategy. However, if too many page locks are taken for a table, lock escalation occurs so that a single tablespace lock is acquired instead.

For online processing, lock escalation is undesirable. If locking problems occur with LOCKSIZE(ANY) because of lock escalation, review the application design, and take sync points much more frequently.

Note: In a hybrid IMS and DB2 program, IMS is in control of the sync point processing through the checkpoint. Therefore, do not issue DB2 commit calls in a hybrid IMS/DB2 program, because they are ignored by DB2 if issued. You must use IMS checkpoint calls to perform synchronization processing for both DB2 and IMS to free up locks.

The exception to the recommendation is for WFI transactions, which should use tablespaces defined with LOCKSIZE(PAGE). If ANY is specified, and lock escalation takes place, then locking remains at the tablespace level until program termination (not transaction termination) which is undesirable; no de-escalation facility exists.

- ▶ LOCKSIZE(TABLESPACE) or LOCKSIZE(TABLE)

These are not recommended in an online environment.

- ▶ LOCKSIZE(ROW)

For normal transaction processing, no benefit is generally provided by locking rows rather than pages. Therefore, its use is not recommended unless it is the only way to suppress locking conflicts.

BIND ACQUIRE parameter

This parameter determines when tablespace intent locks are acquired:

- ▶ With ACQUIRE(ALLOCATE), all locks are taken when the PLAN is allocated. This corresponds to when the thread is created, which in turn occurs at the first SQL call after IMS program scheduling.
- ▶ With ACQUIRE(USE), each tablespace intent lock is acquired individually, as each table space is first referenced by the application.

If most of the tables included in a PLAN are actually referenced at each execution, then we recommend that you use ACQUIRE(ALLOCATE), which is best. A locking problem with any tablespace is recognized before any work is done. With ACQUIRE(ALLOCATE), the first transaction pays the full cost. ACQUIRE(ALLOCATE) is useful for IMS WFI transactions.

If some tables are not referenced in every execution, then we recommend that you use ALLOCATE(USE), because this is the most efficient technique.

Note: If you are using DB2 packages, you cannot choose. ACQUIRE(USE) is forced.

BIND RELEASE parameter

This parameter determines when tablespace locks are released. Page locks or row locks are released during message processing or at sync point time, exactly as for IMS IRLM locks. The RELEASE parameter can have a significant effect on performance:

- ▶ With RELEASE(DEALLOCATE), the tablespace locks are released when the PLAN is deallocated (in other words, when the IMS program terminates).
- ▶ With RELEASE(COMMIT), the tablespace locks are released at each sync point.

RELEASE(COMMIT) can have a severe performance impact as plan resources (tablespace locks, cursor sections, or DB2 DBDs) are released at sync point and have to be reacquired if the same IMS program continues processing the next message. Consequently, we recommend that you use RELEASE(DEALLOCATE), which is the default. However, this option causes contention on the application, because locks are held across messages.

BIND ISOLATION parameter

This parameter determines when shared page locks are released:

- ▶ At commit time, Repeatable Reads (RR) are requested with ISOLATION(RR). This is similar to accessing a DL/I segment with a Q command code in the SSA. It prevents any updates from gaining access to the page, but it increases the number of concurrent locks held by the transaction.
- ▶ When the position moves off the page, if Cursor Stability (CS) is requested with ISOLATION(CS). This is the same as standard record locking in IMS; when the PCB position moves to a new root, the previous root is unlocked (if not updated).
- ▶ The fewest locks are acquired when you use Uncommitted Read (UR), ISOLATION(UR). It is fast and causes little contention, but it reads uncommitted data just like a PROCOPT of GOx.

Unless the application has a specific need for referenced data to remain unchanged, our recommendation is to use ISOLATION(CS) or ISOLATION(UR).

For queries that do not update, we recommend that you always code FOR READ ONLY WITH UR on the query.

BIND CURRENT DATA parameter

DB2 has a lock-avoidance facility that is equivalent to PROCOPT=GOx in IMS. The parameter values are:

- ▶ CURRENT DATA(YES): Take locks
- ▶ CURRENT DATA(NO): Take no locks

This function provides DB2 with a dirty read capability. When using CURRENT DATA(NO) in the BIND, DB2 does not lock before a read.

In DB2, it is not always possible to specify the update intention in the EXEC SQL SELECT call. Therefore, the use of CURRENT DATA(NO) can be dangerous and can cause integrity exposures when two programs attempt to update the same data without locking it at read time.

For more details, refer to the *DB2 Universal Database for z/OS Administration Guide*, SC18-7413.

10.2.3 DB2 free space

DB2 uses free space in a tablespace just like IMS uses free space in a database. There can be x percentage free on a DB2 page (which is the same thing as an IMS OSAM block or a VSAM CI), and so many pages left empty after a load or reorganization. In order to insert efficiently, there must be free space in the tablespace. The same concept on free space for IMS applies to DB2. See 5.6, “Free space” on page 70 for a detailed discussion about the use of free space.

10.2.4 Static as opposed to dynamic SQL

In this section, we cover the performance differences between static and dynamic SQL.

Static SQL

With static SQL, SQL statements are embedded within a program and are prepared during the program preparation process before program execution. After the statements are prepared, they do not change, although the values of the host variables specified by the statements might change.

For best performance, we recommend that you use static SQL for conventional predefined transactions or programs. Static SQL avoids the EXEC SQL PREPARE overhead for access path selection.

Dynamic SQL

With dynamic SQL, the SQL source is contained in host language variables and not coded explicitly into the application program. When the program is executed, the SQL code can be varied or dynamically created before being executed.

In a transaction processing environment, the use of dynamic SQL usually has a negative impact on performance.

If dynamic SQL must be used, we recommend that you try to avoid multiple PREPAREs. If the statement is going to be used only once, it is better to code EXEC SQL EXECUTE IMMEDIATE. If the statement is going to be executed several times, possibly with different values, it is better to code EXEC SQL PREPARE once and code EXEC SQL EXECUTE with placement marker substitution.

10.2.5 Security controls

Security controls occur at various stages of processing, depending on the system definition and the security product you use (RACF, for example). The following security controls are frequently used:

- ▶ IMS authorization to access protected data sets
- ▶ IMS authorization to connect to DB2 subsystem
- ▶ User authorization to sign on to IMS
- ▶ User authorization to access an IMS transaction

- ▶ User authorization to sign on to DB2
- ▶ User authorization to submit IMS or DB2 commands

Security verification tasks affect performance. We recommend that you use the minimum level of security that meets all your security requirements.

10.3 Multi-row FETCH and INSERT

Multi-row fetch eliminates frequent trips between the application and the database engine. A single fetch statement can retrieve numerous rows of data from the result table of a query as a rowset (a *rowset* is the set of rows that is retrieved through a multi-row fetch). When the cursor is positioned on a rowset, the rows belonging to the rowset can be updated or deleted.

Explicit multi-row FETCH, INSERT, cursor UPDATE, and cursor DELETE can improve performance with a reduction of CPU consumption if a program processes a considerable number of rows. Depending on each particular case, a number between 10 rows and 100 rows per SQL can be an advantageous starting point for multi-row operations.

You need to define host variable array declaration of the cursor with rowset positioning and the text of the SQL to specify the rowset size.

You also have to modify your coding regarding how to perform cursor processing. You need to look at how many rows are returned on a FETCH using the SQERRD3 flag in the SQLCA area in your program.

Multi-row FETCH, INSERT, cursor UPDATE, and cursor DELETE are invoked by adding the general statement of “FOR x ROWS” to your SQL. Refer to the current version of the DB2 SQL Reference for the exact syntax and details about using “FOR x ROWS.”

10.4 Tools for monitoring

All the products that are normally used for separate IMS and DB2 subsystems can be implemented to monitor the system when IMS and DB2 are used together. The primary monitoring products are:

z/OS Tools

Resource Measurement Facility III (RMF) that monitors the physical resource utilization of the subsystem environment.

IMS Tools

Following are the IMS Tools available:

- ▶ IMS Performance Analyzer

IMS Performance Analyzer provides different reports with information about the ESAF.

- ▶ IMS Monitor

The IMS Monitor traces all IMS interactions with DB2, such as thread creation, signon, SQL calls, commit, and thread termination.

- ▶ IMS ESAF SNAP records and traces

When a deadlock situation occurs in DB2, an x'67FF' record is written to IMS log. The DFSERA10 and DFSERA30 modules can be used to report this deadlock information. See 10.4.3, “Deadlock report” on page 198 for an example.

- **IMS SSR command**

IMS SSR command allows the operator to route DB2 DISPLAY commands to the desired DB2 subsystem in order to display the DB2 status, database status, usage of buffer pools, and locks to be observed as they stand at one point in time. Refer to Example 10-1.

Example 10-1 SSR command

```
/SSR - DISPLAY THREAD (*)
/SSR - DISPLAY DATABASE(xxxx) SPACE(*) RESTRICT
/SSR - DISPLAY DATABASE(xxxx) SPACE(*) LOCKS
/SSR - DISPLAY BUFFERPOOL (*)
```

DB2 Tools

An exhaustive analysis of the DB2 performance tools and details about the different capabilities that each tool provides can be found in the IBM Redbook, *New Tools for DB2 for OS/390 and z/OS Presentation Guide*, SG24-6139.

- **DB2 Performance Monitor (DB2PM)**

DB2PM is a performance tool with all the capabilities that you might need when verifying and monitoring the critical performance elements of the DB2 environment.

It includes a variety of customized reports for in-depth performance analysis and exception reporting. Reports are produced from DB2 statistics, accounting, and performance trace data collected in the System Management Facility (SMF), GTF, or user files. An online monitor provides an immediate snapshot view of applications and DB2 activities through ISPF menus and panels. The Explain function is useful to analyze and optimize SQL statements.

- **SQL Performance Analyzer**

The purpose of this tool is to help database users, developers, and administrators assess the SQL performance problems. It provides realistic costs and resource usage for DB2 SQL statements without having to execute them and incorporates warnings, alerts, and recommendations for the queries.

- **DB2 Query Monitor Tool (QM)**

This is a thread monitoring tool, which reports all dynamic and static SQL queries issued as well as commands and utilities executed. QM complements DB2PM and includes three major facilities: dynamic monitoring, problem detection, and determination and analysis of historical data.

- **DB2 Trace Facility**

Using the DB2 trace, subsystem data and events related to performance or accounting can be recorded for further use by other print programs.

10.4.1 IMS Performance Analyzer

IMS Performance Analyzer produces two different reports related to the ESAF:

- **IMS Performance Analyzer ESAF (Log) report**

Provides a chronological listing of all the external subsystem connects and disconnects.

- **External Subsystem: Region Detail and External Subsystem: Program Detail sections of IMS Performance Analyzer ESAF (Monitor) report**

A detailed analysis of external subsystem activity in regions and by application programs can be obtained by means of this report. It is useful to determine what percentage of time

the transactions are spending inside DB2. Refer to Example 10-2 and Example 10-3 for examples of both these reports.

Example 10-2 IMS Performance Analyzer ESAF (Monitor) report (External Subsystem: Region Detail)

Report from 25Aug2006 09.34.16.61 IMS 9.1.0 IMS Performance Analyzer 4.1 Report to 25Aug2006 09.40.23.13

External Subsystem: Region Detail

From 25Aug2006 9.35.02.62 To 25Aug2006 9.40.04.70 Elapsed= 0 Hrs 5 Mins 02.080.071 Secs

Rgn No.	SSID	Function	Mod	----- Subsystem Calls -----				-- Transaction --			
				Count	Avg Elapse Sc.Mil.Mic	Std Dev	Max Elapse Sc.Mil.Mic	Count	Avg Elapse Sc.Mil.Mic	Calls /Tran	Pct Elaps
6	DB24	Normal Call	PRO	521	0.436	4.470	22.417	51	38.102	10.2	11.68%
		Signon	S00	48	6.547	0.173	10.946			0.9	16.17%
		Create Thread	CT0	46	6.755	6.176	286.629			0.9	15.99%
		Commit Ph1	P10	48	0.476	2.008	5.848			0.9	1.18%
		Commit Ph2	P20	3	2.539	0.076	2.795			0.1	0.39%
		Term Thread	D50	46	0.949	0.650	4.442			0.9	2.25%
		** Total **		712	1.301	8.415	286.629			14.0	47.66%
9	DB24	Normal Call	PRO	667	0.320	4.144	25.612	73	25.512	9.1	11.45%
		Signon	S00	71	7.079	0.304	21.825			1.0	26.99%
		Create Thread	CT0	68	1.192	3.950	39.602			0.9	4.35%
		Commit Ph1	P10	71	0.409	1.750	4.818			1.0	1.56%
		Commit Ph2	P20	4	2.662	0.128	3.116			0.1	0.57%
		Term Thread	D50	68	0.952	0.404	2.281			0.9	3.48%
		** Total **		949	0.950	2.652	39.602			13.0	48.40%
12	DB24	Normal Call	PRO	835	0.669	4.570	34.434	65	35.284	12.8	24.35%
		Signon	S00	64	7.006	0.328	21.814			1.0	19.55%
		Create Thread	CT0	64	2.502	4.423	77.734			1.0	6.98%
		Commit Ph1	P10	64	0.457	1.395	3.590			1.0	1.27%
		Commit Ph2	P20	4	2.541	0.113	2.824			0.1	0.44%
		Term Thread	D50	64	0.962	0.363	2.605			1.0	2.68%
		** Total **		1095	1.158	3.555	77.734			16.8	55.29%

Example 10-3 IMS Performance Analyzer ESAF (Monitor) report (External Subsystem: Program Detail)

Report from 25Aug2006 09.34.16.61 IMS 9.1.0 IMS Performance Analyzer 4.1 Report to 25Aug2006 09.40.23.13

External Subsystem: Program Detail

From 25Aug2006 9.35.02.62 To 25Aug2006 9.40.04.70 Elapsed= 0 Hrs 5 Mins 02.080.071 Secs

Rgn No.	PSBname	Trancode	SSID	Function	Mod	----- Subsystem Calls -----				-- Transaction --			
						Count	Avg Elapse Sc.Mil.Mic	Std Dev	Max Elapse Sc.Mil.Mic	Count	Avg Elapse Sc.Mil.Mic	Calls /Tran	Pct Elaps
6	PFZPP04	PFZUP04	DB24	Normal Call	PRO	1	0.544	0.000	0.544	1	12.265	1.0	4.44%
				Signon	S00	1	5.635	0.000	5.635			1.0	45.94%
				Create Thread	CT0	1	0.493	0.000	0.493			1.0	4.02%
				Commit Ph1	P10	1	0.194	0.000	0.194			1.0	1.58%
				Term Thread	D50	1	0.708	0.000	0.708			1.0	5.77%
				** Total **		5	1.515	1.364	5.635			5.0	61.75%
6	PFZPP05	PFZUP05	DB24	Normal Call	PRO	515	0.438	4.470	22.417	48	39.957	10.7	11.77%
				Signon	S00	46	6.574	0.174	10.946			1.0	15.77%
				Create Thread	CT0	44	7.040	6.056	286.629			0.9	16.15%
				Commit Ph1	P10	46	0.488	1.997	5.848			1.0	1.17%
				Commit Ph2	P20	3	2.539	0.076	2.795			0.1	0.40%
				Term Thread	D50	44	0.963	0.652	4.442			0.9	2.21%
				** Total **		698	1.304	8.474	286.629			14.5	47.46%
9	PFZPP05	PFZTHALT	DB24	Normal Call	PRO	353	0.316	4.409	25.612	33	19.446	10.7	17.37%
				Signon	S00	33	7.359	0.376	21.825			1.0	37.85%
				Create Thread	CT0	33	0.667	0.777	2.698			1.0	3.43%
				Commit Ph1	P10	33	0.277	0.278	0.415			1.0	1.42%
				Term Thread	D50	33	1.015	0.436	2.281			1.0	5.22%
				** Total **		485	0.864	2.610	25.612			14.7	65.28%
9	PFZPP05	PFZUP05	DB24	Normal Call	PRO	251	0.222	2.068	6.243	28	15.416	9.0	12.91%

			Signon	S00	27	6.453	0.096	8.402			1.0	40.36%	
			Create Thread	CT0	24	0.559	0.485	1.715			0.9	3.11%	
			Commit Ph1	P10	27	0.360	1.372	2.121			1.0	2.25%	
			Commit Ph2	P20	2	2.647	0.177	3.116			0.1	1.23%	
			Term Thread	D50	24	0.882	0.356	1.958			0.9	4.90%	
			** Total **		355	0.787	2.168	8.402			12.7	64.77%	
12	PFZPP01	PFZUP01	DB24	Normal Call	PRO	5	0.417	0.108	0.476	5	31.578	1.0	1.32%
			Signon	S00	5	8.047	0.309	12.859				1.0	25.48%
			Create Thread	CT0	5	0.515	0.095	0.612				1.0	1.63%
			Commit Ph1	P10	5	0.187	0.095	0.213				1.0	0.59%
			Term Thread	D50	5	0.743	0.130	0.897				1.0	2.35%
			** Total **		25	1.982	1.632	12.859				5.0	31.38%
12	PFZPP05	PFZTHALT	DB24	Normal Call	PRO	720	0.706	4.592	34.434	50	24.942	14.4	40.74%
			Signon	S00	50	6.931	0.350	21.814				1.0	27.79%
			Create Thread	CT0	50	0.592	0.517	2.167				1.0	2.37%
			Commit Ph1	P10	50	0.320	0.411	0.876				1.0	1.28%
			Term Thread	D50	50	0.985	0.361	2.605				1.0	3.95%

10.4.2 IMS Monitor

The IMS Monitor provides statistics about these significant DB2 events:

- ▶ Service calls, such as create thread, signon, commit Phase 1, commit Phase 2, and terminate thread
- ▶ Normal SQL calls
- ▶ Command calls using the IMS /SSR command

The information is reported by subsystem, region, or PSB. You get the number of occurrences of each type of call and their elapsed time. Each call is one IWAIT (the NOT-IWAIT time is always zero).

Examples are shown in:

- ▶ Example 10-4:
IMS Monitor Region Summary report showing DB2 Service and Command Calls
- ▶ Example 10-5 on page 196:
IMS Monitor Region Summary report showing DB2 Normal Calls
- ▶ Example 10-6 on page 196:
IMS Monitor Region IWAIT report showing DB2 IWAIT
- ▶ Example 10-7 on page 196:
IMS Monitor Program I/O report showing DB2 IWAIT
- ▶ Example 10-9 on page 197:
IMS Monitor Call Summary report showing DB2 Calls

Example 10-4 IMS Monitor Region Summary report (DB2 Service and Command Calls)

IMS MONITOR			*** REGION SUMMARY ***			TRACE START 2006 270, 09:23:51			TRACE STOP 2006 270, 09:31:00		
		ELAPSED TIME.....			NOT IWAIT TIME(ELAPSED-IWAIT)					
<u>OCCURRENCES</u>			<u>TOTAL</u>	<u>MEAN</u>	<u>MAXIMUM</u>	<u>TOTAL</u>	<u>MEAN</u>	<u>MAXIMUM</u>			
DB24 <u>SERVICE AND COMMAND CALLS</u>											
**REGION	6	217	592332	2729	65545						
**REGION	9	231	580401	2512	71352						

**REGION	12	98	319788	3263	88558
**REGION	14	224	542121	2420	32179
**REGION	15	226	614990	2721	65914
**REGION	18	223	603638	2706	61631
**REGION	20	223	504183	2260	19741
**REGION	22	246	773047	3142	84968
**REGION	23	226	528254	2337	30231
**TOTALS		1914	5058754	2643	

Example 10-5 IMS Monitor Region Summary report (DB2 Normal Calls)

IMS MONITOR *** REGION SUMMARY *** TRACE START 2006 270, 09:23:51 TRACE STOP 2006 270, 09:31:00

		ELAPSED TIME.....			NOT IWAIT TIME(ELAPSED-IWAIT)		
OCCURRENCES			TOTAL	MEAN	MAXIMUM	TOTAL	MEAN	MAXIMUM
DB24	NORMAL	CALLS						
**REGION	6	568	1031731	1816	126262			
**REGION	9	537	668054	1244	87589			
**REGION	12	148	338185	2285	46117			
**REGION	14	532	363598	683	35583			
**REGION	15	614	337343	549	34882			
**REGION	18	531	298572	562	30289			
**REGION	20	476	569527	1196	39248			
**REGION	22	703	731782	1040	43733			
**REGION	23	530	167065	315	25978			
**TOTALS		4639	4505857	971				

Example 10-6 IMS Monitor Region IWAIT report (DB2 IWAIT)

IMS MONITOR *** REGION IWAIT *** TRACE START 2006 270, 09:23:51 TRACE STOP 2006 270, 09:31:00

	IWAIT TIME.....				
**REGION	6 OCCURRENCES	TOTAL	MEAN	MAXIMUM	FUNCTION	MODULE
DB24	CALLS					
	50	57193	1143	8406	TERM THRD	D50
	57	26873	471	4156	COMMIT PH.1	P10
	568	1031731	1816	126262	NORMAL CALL	PRO
	50	97383	1947	65545	CREATE THRD	CT0
	57	403211	7073	13593	SIGNON	S00
	3	7672	2557	3430	COMMIT PH.2	P20
...TOTAL...	785	1624063	2068			

Example 10-7 IMS Monitor Program I/O report (DB2 IWAITs)

Example 10-8

IMS MONITOR *** PROGRAM I/O *** TRACE START 2006 270, 09:23:51 TRACE STOP 2006 270, 09:31:00

		IWAIT TIME.....				
PSBNAME	PCB NAME	IWAITS	TOTAL	MEAN	MAXIMUM	DDN/FUNC	MODULE
PFZPP05	I/O PCB	35	46740	1335	3606	DHZDSZP1	DBH
		1	1001	1001	1001	GMZDTRP3	DBH
	PCB TOTAL	36	47741	1326			

GMZDTRP3	1	13094	13094	13094	GMZDTRP3	DBH
<u>PCB TOTAL</u>	1	13094	13094			
DHZDSZP1	1	10423	10423	10423	DHZDSZP1	DBH
<u>PCB TOTAL</u>	1	10423	10423			
DB24	413	435756	1055	21756	TERM THRD	D50
	448	170370	380	4372	COMMIT PH.1	P10
	4408	4094602	928	126262	NORMAL CALL	PRO
	413	312777	757	65545	CREATE THRD	CTO
	448	3474268	7755	71352	SIGNON	S00
	16	37916	2369	3430	COMMIT PH.2	P20
<u>SUBSYS TOTAL</u>	6146	8525689	1387			

Example 10-9 IMS Monitor Call Summary report (DB2 Calls)

IMS MONITOR			*** CALL SUMMARY ***			TRACE START 2006 270, 09:23:51			TRACE STOP 2006 270, 09:31:00			
PSB NAME	PCB NAME	CALL FUNC	LEV NO_SEGMENT	STAT CODE	CALLS	IWAITS	IWAITS/CALL	..ELAPSED TIME... MEAN	..ELAPSED TIME... MAXIMUM	.NOT IWAIT TIME.. MEAN	.NOT IWAIT TIME.. MAXIMUM	
PFZPP05	I/O PCB	ASRT ()			419	0	0.00	1118	21874	1118	21874	
		GU ()		QC	419	31	0.07	5293	340720	5193	340720	
		ISRT ()			457	0	0.00	152	21879	152	21879	
		CHNG ()			28	0	0.00	1545	14345	1545	14345	
		PURG ()			28	0	0.00	70	1092	70	1092	
		(GU) ()			419	0	0.00	14	480	14	480	
		GU ()			38	5	0.13	4516	9354	4354	9354	
		<u>I/O PCB SUBTOTAL</u>				1808	36	0.01	1647		1621	
		GMZDSZP1	REPL (02)GMZSSZRN			19	0	0.00	777	13036	777	13036
			GU (02)GMZSSZRN			28	0	0.00	93	833	93	833
REPL (01)GMZSSZ00				19	0	0.00	46	181	46	181		
GU (01)GMZSSZ00				31	0	0.00	86	467	86	467		
GN (01)GMZSSZ00	GE			53	0	0.00	20	41	20	41		
REPL (02)GMZSSZT2				32	0	0.00	32	56	32	56		
GN (02)GMZSSZT2				90	0	0.00	28	142	28	142		
GU (02)GMZSSZT2				90	0	0.00	112	3365	112	3365		
ISRT (02)GMZSSZT2				12	0	0.00	84	208	84	208		
DLET (02)GMZSSZT2				12	0	0.00	1094	11628	1094	11628		
ISRT (02)GMZSSZRN				1	0	0.00	89	89	89	89		
GU (01)GMZSSZ00	GE			1	0	0.00	31	31	31	31		
ISRT (01)GMZSSZ00				1	0	0.00	135	135	135	135		
GU (00)			GE	1	1	1.00	10534	10534	111	111		
<u>DL/I PCB SUBTOTAL</u>				390	1	0.00	155		128			
DHZDKSP1	GU (01)DHZSKS00			218	0	0.00	66	1931	66	1931		
	GNP (02)DHZSKSTR			839	0	0.00	44	12208	44	12208		
<u>DL/I PCB SUBTOTAL</u>				1057	0	0.00	48		48			
DHZDTRP3	ISRT (01)DHZSTR00				1	1	1.00	13544	13544	450	450	
<u>DL/I PCB SUBTOTAL</u>				1	1	1.00	13544		450			

DB24	TERM THRD	OK	413	413	1.00	1055	21756	0	0
	COMMIT PH.1	OK	448	448	1.00	380	4372	0	0
	NORMAL CALL	OK	4408	4408	1.00	928	126262	0	0
	CREATE THRD	OK	413	413	1.00	757	65545	0	0
	SIGNON	OK	448	448	1.00	7755	71352	0	0
	COMMIT PH.2	OK	16	16	1.00	2369	3430	0	0
	<u>SUBSYS SUBTOTAL</u>		6146	6146	1.00	1387		0	
	<u>PSB TOTAL</u>		9402	6184	0.65	1237		322	

10.4.3 Deadlock report

When DB2 detects a deadlock, the ESAF message handler directs the IMS ILOG macro to log a x'67FF' SNAP trace record to identify the problem, and the application ABENDs with a U0777. Also, a DFS3624I message is issued prior to the ABEND of the dependent region.

This information is reported on the Deadlock report of the File Select and Formatting Print utility (DFSERA10). The Record Format and Print Module (DFSERA30) is needed to format the DEADLOCK report. When DFSERA30 encounters a deadlock block, it prints the block and produces a report based on the data in the block.

See Example 10-10 for a typical DB2 Deadlock report. See Example 10-11 on page 199 for a typical IMS Deadlock report. Note that the hexadecimal data from the log was removed from this example.

Example 10-10 DB2 deadlock report

```

*-----*
* THIS ROUTINE WILL LOOK FOR LOG RECORD X'67FF' AND THE NAME OF *
* THE BLOCK THAT STARTS WITH DEADLOCK. IT WILL THEN BE RUN *
* THROUGH DFSERA30 TO PRINT A NICE REPORT. *
*-----*
CONTROL  CNTL  STOPAFT=EOF
OPTION   PRINT OFFSET=05,FLDTYP=X,VALUE=67FF,FLDLLEN=2,COND=M
OPTION   PRINT OFFSET=33,FLDTYP=C,VALUE=DEADLOCK,FLDLLEN=8,COND=E,      X
          EXITR=DFSERA30
END
DFSERA30  -  FORMATTED LOG PRINT

PSEUDO ABEND RECORD  ABEND NO = 0777 RECNO = 0583D971  TIME  16:05:00.1  DATE 2006.270

DEADLOCK

EXTERNAL SUBSYSTEM DB28      DETECTED A DEADLOCK DURING NORMAL CALL
REGION TYPE   : MPP
REGION NUMBER : 004B
JOB NAME      : IMSFMSGK
PSB NAME      : SASPR100
SMB NAME      : SASPR100
RECOVERY TOKEN: C9D4D7C640404040000E2A8800000000

DEADLOCK ANALYSIS REPORT - END OF REPORT

```

Example 10-11 IMS deadlock report

```
*-----*
* THIS ROUTINE WILL LOOK FOR LOG RECORD X'67FF' AND THE NAME OF *
* THE BLOCK THAT STARTS WITH DEADLOCK. IT WILL THEN BE RUN *
* THROUGH DFSERA30 TO PRINT A NICE REPORT. *
*-----*
CONTROL  CNTL  STOPAFT=EOF
OPTION   PRINT OFFSET=05,FLDTYP=X,VALUE=67FF,FLDLN=2,COND=M
OPTION   PRINT OFFSET=33,FLDTYP=C,VALUE=DEADLOCK,FLDLN=8,COND=E,      X
        EXITR=DFSERA30
END

DFSERA30  -  FORMATTED LOG PRINT

PSEUDO ABEND RECORD  ABEND NO = 0777  RECNO = 0583EE8C  TIME 16:05:02.2  DATE 2006.270

<... for illustration purposes part of the report was removed here ...>

DEADLOCK ANALYSIS REPORT - LOCK MANAGER IS PI
.....

RESOURCE DMB-NAME LOCK-LEN LOCK-NAME      - WAITER FOR THIS RESOURCE IS VICTIM
01 OF 02 GRVDBOP1      08      014A05D400210140

KEY FOR RESOURCE IS FROM DELETE WORK AREA
KEY=(WI74907      123765976)

      IMS-NAME TRAN/JOB PSB-NAME PCB--DBD PST#  RGN  CALL LOCK  LOCKFUNC STATE
WAITER IMSF      GRVPR200 GRVPR200 GRVDBOP1 00074 MPP  DLET GRIDX 30400378 03
BLCKER IMSF      GRVPR400 GRVPR400 ----- 00093 MPP  ---- ---- 03
.....

RESOURCE DMB-NAME LOCK-LEN LOCK-NAME
02 OF 02 GRVDBOP1      08      0149E6E400210140

LOCKING ON NEXT HIDAM ROOT FOR GN CALL, KEY DISPLAYED IS FOR PRIOR HIDAM ROOT
KEY=(WI74883      123765976)

      IMS-NAME TRAN/JOB PSB-NAME PCB--DBD PST#  RGN  CALL LOCK  LOCKFUNC STATE
WAITER IMSF      GRVPR400 GRVPR400 GRVDBOP1 00093 MPP  GET  GRIDX 30400378 03
BLCKER IMSF      GRVPR200 GRVPR200 ----- 00074 MPP  ---- ---- 03

DEADLOCK ANALYSIS REPORT - END OF REPORT
```

10.5 When to reorganize your DB2 tablespace or indexspace

There are several indicators from the DB2 catalog that indicate when to reorganize a tablespace or indexspace. RUNSTATS needs to be run on a regular basis after tables have

been populated and are in their “general” production state. We suggest that you run REBIND after all of the reorganizations are completed for the application.

10.5.1 Tablespace

One of the indicators that a tablespace needs reorganizing is when NEAROFFPOSF divided by CARDF from SYSIBM.SYSINDEXPART is greater than 10% on the primary index for a tablespace. Another indicator is when FAROFFPOSF divided by CARDF from SYSIBM.SYSINDEXPART is greater than 10% on the primary index for a tablespace.

The CLUSTERRATIO from SYSIBM.SYSINDEXES on the primary index for a tablespace has several guidelines. If CARDF from SYSIBM.SYSINDEXPART is greater than 1 but less than 30 000, then the cluster ratio should be 100%. More than 30 000 but fewer than 500 000 rows, a 99% cluster ratio or better is suggested. If there are more than 500 000 but fewer than 10 000 000 rows, then the endorsed cluster ratio is 98% or better. When you get in the range of more than 10 000 000 rows but fewer than 100 000 000 rows, then the cluster ratio should not fall below 97%. A 95% cluster ratio or better is acceptable for more than 100 000 000 rows.

There are two parameters on the reorganization utility that you can specify to reorganize a tablespace. One is taken from the DB2 catalog table SYSIBM.SYSINDEXPART. The OFFPOSLIMIT, which is the percentage of the NEAROFFPOSF plus the FAROFFPOSF divided by the CARDF, should be less than 10%. The other parameter on the reorganization utility is the INDREFLIMIT, which is the NEARINDREF plus the FARINDREF divided by the CARDF from SYSIBM.SYSTABLEPART, and its percentage should be less than 10%.

The dropped percentage from SYSIBM.SYSTABLEPART should be 10% or less; otherwise, it is time to reorganize.

10.5.2 Indexspace

Indexspaces should be reorganized when the LEAFDIST from SYSIBM.SYSINDEXPART is greater than 200. Also, when LEAFFAR times three plus LEAFNEAR from SYSIBM.SYSINDEXPART is divided by NLEAF from SYSIBM.SYSINDEXES and that value is greater than 20, it is time to reorganize the indexspace.

Indexspaces should also be reorganized when the PSEUDO_DEL_ENTRIES from SYSIBM.SYSINDEXPART divided by the CARDF column from SYSIBM.SYSINDEXPART percentage is greater than 10.

10.6 More information

For more information about DB2 performance, see *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465.

IMS Parallel Sysplex considerations

With Parallel Sysplex, there are several additional factors which you must take into consideration for IMS performance. Some of these are typically outside the control of the IMS systems programmer but must be understood anyway. We document these considerations in this chapter.

This chapter covers the following topics:

- ▶ CF hardware and microcode
- ▶ Structure sizing
- ▶ IRLM considerations
- ▶ System-managed duplexing
- ▶ OSAM and VSAM cache structures
- ▶ Application considerations
- ▶ Shared VSO
- ▶ Shared queues considerations

11.1 Hardware and microcode

The Coupling Facilities can have a significant impact on IMS performance. Depending on your particular hardware configuration, you might need to use certain functions to maintain high availability, which are less than ideal for performance. These facilities include system-managed duplexing and the use of staging data sets for the MVS system logger. These things are discussed more in this chapter.

11.1.1 Coupling Facility configuration

Coupling Facilities can be part of the same physical box as the z/OS LPAR or they can be stand-alone boxes. If they are stand-alone boxes, you might generally be able to achieve high availability without having to use system-managed duplexing or staging data sets. If the CF LPARS are on the same physical box as the z/OS images running IMS, then it might be necessary to use these functions to prevent a loss of integrity. A few other things to be aware of in a production environment are:

- ▶ Dynamic dispatching for the CF LPAR might cause poor or erratic CF response times.
- ▶ Multiple CF engines, even when utilization is relatively low, can help keep response times more stable.
- ▶ Multiple CF links not only help availability but also benefit performance.

11.1.2 Coupling Facility microcode

New levels of CF microcode generally introduce new capabilities. Some of these new functions can take up additional control storage so that there might be less storage available for user data in the same size structure. The good news is that except for preloaded VSO, the microcode changes rarely cause any failures and IMS continues to run. The bad news is that your IMS performance could suffer because of something you might not even know about. For this reason, you should make sure your systems people let you know any time there is a change to the CF microcode or the configuration. Make sure you know how to obtain and read the RMF CF Structure Activity reports and keep a repository of these reports for future reference in the event of problems.

11.2 Structure sizing

Individual structure sizing considerations are discussed later in this chapter, but for all IMS structures as well as other system and subsystem structures, you might reference the following IBM Web site:

<http://www.ibm.com/servers/eserver/zseries/cfsizer>

If you click IMS, you can select any or all of the IMS-related structures to estimate the size. This Web site normally calculates the size based on the most current CFLEVEL available, which is displayed in the results page. Structure sizes are always a multiple of 256K so if you specify a size, which is not a multiple of 256K, the system rounds it up to the next 256K increment.

11.3 IRLM considerations

A couple of IRLM startup parameters need to be considered for IRLM performance:

► PC=YES or NO

IMS continues to support the use of IRLM 2.1 at this point in time, which allows the specification of either value. IRLM 2.2 allows either value to be specified but always runs with PC=YES regardless of the setting. Many years ago specifying PC=NO could have some impact on IRLM CPU usage but with today's processors, there is very little difference in CPU and removing the lock table from ECSA helps free up valuable common storage.

► DEADLOK=

Using values less than 1 second for DEADLOK have little impact on CPU but might be very beneficial by resolving deadlocks quickly, because there are typically hundreds of transactions running every second. Waiting a second or more to resolve a deadlock can back up transaction processing significantly, which might result in other undesirable conditions.

► MAXUSRS=

MAXUSRS value determines the size of each entry in the lock table, which can be 2, 4, or 8 bytes. MAXUSRS represents the number of IRLMs that you expect to have in the data sharing group. Any number up to 7 causes each entry to be 2 bytes, 8 through 23 create 4-byte entries, and numbers greater than 23 create 8-byte entries. As you can see, this number directly affects the number of lock entries in a given size structure, so use the smallest number (or something in the ranges mentioned) that meets your needs.

► LTE=

LTE value lets you override the IRLM default calculation of 1/2 of the lock structure is for LTEs. The number specified is multiplied by 1 048 576. Be careful specifying this value, because it is possible to severely restrict the number of record list entries and possibly cause tasks to fail. If more LTEs are necessary to reduce false contention, then it is usually better to increase the size of the lock structure. See below for more information.

F irlmproc,SET,TIMEOUT=nnnn,dbms

Use this command to establish a time after which IMS is notified that some tasks have been waiting for a lock longer than the time specified and also to create a 79.15 SMF record along with a DXR162I message issued by the IRLM. If no command is issued, then the default is 300 seconds (five minutes), which is probably longer than it takes for the phone to start ringing. Note that you must issue this command for each IMS, and it must be after IMS has identified to the IRLM.

LOCKTIME= in the DFSVSMxx member of IMS PROCLIB

This value interacts with the TIMEOUT value mentioned above. LOCKTIME, if specified, overrides the default value of 300 seconds; however, there is an important difference between specifying here as opposed to using the IRLM command. If LOCKTIME is specified, then any applications waiting longer than the specified time are abended with a U3310 or given a BA status code. This might or might not be the result you want. You can change the time using the IRLM SET command, but you cannot change the action (abend or status code) without a restart of IMS.

TRACE=NO should be set from a pure performance perspective. If there are locking problems, then having the trace information is probably worth the cost.

WLM for IRLM

WLM should be set up to put IRLM in SYSSTC, SYSTEM, or at least a performance group with similar settings. Most of the IRLM locking code is run under the dependent region TCB, and the times when IRLM must be dispatched are for contention and deadlock resolution in

which case you want IRLM to be dispatched as quickly as possible, because any delay can have a global (sysplex-wide) effect.

11.4 IRLM lock structure

IMS uses the IRLM to manage locks in a block level data sharing environment. The IRLM uses a lock structure to manage lock requests from its IMS clients.

11.4.1 Lock structure size

The size of the lock structure is defined in the CFRM policy. By default, the IRLM makes half of the structure for LTES. Because of this, we recommend that the size specified is also a power of 2 and that any increases then are double or four times the original amount. A reasonable starting size for production is probably 64 MB. That means 32 MB for LTES, and if there are two-byte entries, then there are 16 million LTES.

11.4.2 False contention

Monitor false contention with RMF and adjust the size of the structure. We recommend the number for false contention is 1/10 of 1 percent. This number might or might not be achievable, because much is dependent on the workload. If higher than 0.1 percent, you can try increasing the size of the structure to reduce this; however, remember that you should always increase by doubling the size. There is normally a point of diminishing (or no) return; so if increasing the size has little or no impact, you have probably reached that point and do not need to increase the size any more.

Remember that any contention, real or false, causes the IRLM to do additional work in order to resolve that contention. This involves the IRLM itself being dispatched, and XCF signalling activity with the other lock managers, which has a negative impact on performance.

11.4.3 Automatic rebuild

If you initially specify MAXUSRS as 7 or fewer so that you have two-byte LTES, then the IRLM initiates a structure rebuild if a seventh IRLM (yes, the seventh) joins the group. This causes the number of LTES to be cut in half, because they are now 4-byte entries instead of 2 (or perhaps 8, instead of 4). Keep this in mind when setting the MAXUSRS and the size of the structure, so that you do not get surprised by a sudden increase in false contention due to a reduction in the number of LTES.

If this happens, you can, of course, alter the size of the structure and then rebuild to get back to the optimal number of LTES.

11.4.4 System-managed duplexing

From a performance perspective, you want to avoid this function, because it can add significantly to the CF structure response time. The IRLM can rebuild the lock structure if something happens, except in the case where both the lock structure and an IRLM fail concurrently (such as when they are both on the same physical hardware). In that case, you have to restart IRLM (and IMS if it also went down) before the lock structure can be rebuilt and data sharing can continue. This is an unlikely event but might have to be considered over performance.

11.5 VSAM cache structure

The VSAM cache structure is a directory only structure, so it should be large enough to have a directory entry for each buffer in every IMS. It actually only needs to have enough entries to handle all of the unique blocks in all of the IMS images (including batch data sharing), but it is impractical and a waste of effort to try to figure out this number. To determine the size, you need to determine the total number of VSAM buffers in all of the IMS images, which might join the data sharing group. This number is then used as input to the CFSIZER program. A quick manual calculation can be done by taking the number of buffers, multiplying by 300 and adding another 256K for CF overhead. Always estimate high. It is better to have a bit too much storage than not enough.

If the size is too small to support all of the buffers in the data sharing group, there are directory reclaims, which can affect performance. In this case, some buffers in IMS cannot be used and any existing data in them has to be discarded.

IMS supports the dynamic ALTER of the VSAM cache structure size, if you determine it is too small. However, beware of using the AUTOALTER facility, because it is possible for the system to decrease the size of the structure if it needs storage for something else.

11.6 OSAM cache structure

The OSAM structure might be a directory only structure or might additionally cache data. If using the structure only for buffer tracking and invalidation as with the VSAM structure, then the sizing considerations are basically the same. Just add up all the OSAM buffers in all the IMS images in the data sharing group, including sequential buffer sets. Feed that number into the CFSIZER or manually calculate as shown above.

For data caching, IMS allows you to specify what is to be cached, if anything, by subpool. Because you can isolate database data sets to specific subpools, you can effectively cache by database data set. The following considerations should help guide you to decide if caching is a good idea:

- ▶ For the CACHE ALL option, consider that every block read into IMS is also written to the structure. This costs CPU and to a lesser extent, elapsed time. This option is probably best for very small databases (or small locality of reference) that are heavily referenced and all the blocks can remain in the structure.
- ▶ Full function database blocks are always written to DASD at sync point, regardless of the cache option.
- ▶ For the CACHE CHANGED option, blocks are only written to the structure if updated. If many IMS images update the same set of blocks frequently, then this option might be appropriate, because it prevents each sharing IMS from rereading updated blocks from DASD.

11.7 DEDB considerations

DEDBs do not use a cache structure except for shared VSO. With Parallel Sysplex data sharing, there is additional overhead however. This overhead is in accessing the lock structure. Shared DEDBs get a global lock to update. Because the IRLM lock structure typically has very fast response time, this is not usually a concern; however, additional contention can occur depending on the access pattern to these databases. Database

contention should be closely monitored and corrective action taken to reduce the contention. Most often this accomplished by using a smaller CI size.

11.7.1 Shared VSO

Shared VSO utilizes a cache structure or many cache structures. Its use is specified on an area by area basis. Here are some considerations:

- ▶ Use the PRELOAD option for small, highly referenced areas. With this option, the entire area must fit into the CF structure. NOPREL accommodates larger areas and can be as effective as PRELOAD in some cases, but it depends on the data access pattern.
- ▶ The LKASID option is normally best. You can monitor the %hits and %valid numbers and if these numbers are extremely low, then you can consider turning off this option. Otherwise, leave it on.
- ▶ Private buffer pools are actually located in ECSA. Keep this in mind, because common storage is a limited resource. If the %hits is not too good, but %valid is almost equal, then this is an indication that additional buffers might help.
- ▶ SVSO updates are written asynchronously to sync point processing. The *castout process* as it is called is initiated at IMS checkpoint time. IMS reads the updated blocks from the CF and writes them to DASD. For this reason, it is important that SVSO areas with lots of updates physically reside on the best performing DASD.
- ▶ IMS duplexing is more efficient than system-managed duplexing. A general guideline is that IMS duplexing is twice the CPU overhead of a single structure with little or no response time increase. System-managed duplexing increases CPU overhead as well but also increases the response time to the structure by three to five times.

11.8 Application considerations

Review PSB PROCOPTs to insure only the necessary processing options are used. This should be done for PSBs used by the high volume transaction. If a particular high volume transaction only performs read operations, then changing the PROCOPT from A to G or even, if business integrity allows, GOT or GON might have a significant impact on reducing contention.

Review DEADLOCK situations. In some cases, the number of deadlocks can actually be reduced with data sharing but that is a very rare situation. If deadlocks are an impact without data sharing, then they most likely get worse when Parallel Sysplex data sharing is implemented.

Look for areas of contention. This might be a simple change of PROCOPT as mentioned above or it might be that an application change is necessary. This is especially the case where a single segment or set of segments are updated by many high volume transactions. As an interim solution, it is sometimes necessary to limit the number of regions which can process a given transaction, or possibly process that transaction type on a single IMS. While this is not a permanent solution for availability, it might be a temporary patch for performance.

11.9 Shared queues

There can be some significant advantages to shared queues, which are primarily automatic workload balancing, increased capacity, and availability.

From a performance perspective, shared queues:

- ▶ Increase CPU usage. How much depends to a large degree on the speed of the Coupling Facility and the links.
- ▶ Increase log data and I/O operations, because the CQS logging is in addition to IMS logging.
- ▶ Increase elapsed times. This is most apparent if the synchronous APPC/OTMA function is enabled, which necessitates the use of MVS™ RRS.
- ▶ Facilitate a decrease in log data for any given IMS image by spreading the transaction load.

11.9.1 IMS parameters

The following IMS parameters can directly impact full function shared queues performance:

- ▶ QBUF

IMS dynamically expands and contracts the number of QBUFs, but it is best to specify a reasonable value. If no value is specified, then the value on the BUFFERS keyword of the MSGQUEUE macro is used, or 255, if BUFFERS is not specified. While the default of 255 is probably not a bad starting place, you should specify a value for QBUF so there is no question as to what value is being used.

- ▶ LGMSGSZ

LGMSGSZ size should be large enough to accommodate most, if not all, of your largest messages, if possible. Any messages larger than this cause additional interaction with the Coupling Facility which impacts performance.

- ▶ QBUFSZ

QBUFSZ is the size of each queue buffer. It should be the same as LGMSGSZ or some multiple. From a performance perspective, there is no real advantage for making this size larger than LGMSGSZ.

- ▶ SHMSGSZ

Set SHMSGSZ value to something evenly divisible into QBUFSZ. The basic principle is to have this value large enough to hold most of the messages, which are smaller than those going into LGMSGSZ to make the best possible use of storage in the queue pool.

- ▶ OBJAVGSZ

The OBJAVGSZ value sets the ratio of list entries to data elements in the Coupling Facility, which can impact overflow processing. If ever in doubt, set this value to 512, which yields a 1:1 ratio. We cover this more in the structure size section.

The other parameters, such as QBUFMAX, QBUFHITH, and so on, are normally fine when you use the default.

11.9.2 Structure size

As with other structures, the best place to get sizing information is with the CFSIZER Web site. It is best to avoid overflow processing under normal conditions. The Web site asks for the SHMSGSZ and LGMSGSZ values as well as the high used DRRN for short and long messages. These values allow for normal activity, but it is probably best to consider making the size much larger, because sometimes delays happen for various reasons. Consider factoring in how long you might want to hold messages if for some reason you had some locking problem hanging up your regions or if some destination, perhaps a remote MSC link, is unavailable. While these might be considered abnormal cases where overflow processing

could help, it might be better to be able to handle some period of time before invoking overflow. Also, consider specifying a SIZE as well as INITSIZE for the queue structure. This allows the structure to be altered to a larger value when it is becoming full. When this happens, there are a number of messages which can be used for automatic notification that there might be a problem.

11.9.3 Structure duplexing

IMS does not provide software duplexing of the queue structures. The same considerations apply to the queue structures as they do to the lock structure. That means from a performance perspective, you want to avoid system-managed duplexing (SMD), but if you want to be able to rebuild a queue structure without having to also restart CQS in the case of a concurrent failure, you would probably have no choice. Consider how likely it is that you would have a double failure such as that before using SMD.

11.9.4 Overflow

Overflow processing occurs whenever the threshold specified in CQSSGxxx is met. CQS first tries to alter the primary structure to a larger size if the SIZE value in the CFRM policy has not yet been reached. If the primary structure cannot be increased, then the overflow structure, if specified, is allocated and some queues moved to decrease the primary structure usage by 20%. Some things to consider:

- ▶ OBJAVGSZ: Only data elements are monitored by CQS, so be sure to set this value low enough to achieve the proper LE/EL ratio. If IMS stops processing due to a structure full condition, that is generally not considered good performance.
- ▶ SIZE of overflow structure: Consider making this the same or larger than the primary structure. This would allow the movement of a queue name in the case where one particular queue name is causing the full condition.
- ▶ Structure activity is quiesced during overflow processing so be sure this is only invoked in rare circumstances.

11.9.5 Structure checkpoint

The speed of processors and links keeps improving, which tends to reduce the impact of taking structure checkpoints, but depending on the amount of data in the structure this could still take several seconds.

- ▶ Activity quiesced while in progress. Time depends on amount of data.
- ▶ Frequency affects time to rebuild queues but also interacts with the amount of MVS logger, which needs to be kept.
- ▶ CFRM couple data sets are accessed by all systems to quiesce and resume, so their performance is extremely important to the checkpoint process.

11.9.6 MVS logger

Performance considerations for the CQS log streams are:

- ▶ Estimate the volume by looking at the IMS 01 and 03 records. There are other records as well, but these make up the large majority of the volume.
- ▶ Avoid the use of staging data sets if possible. Use of a structure with duplexing in a data space is the recommended configuration for best performance.

- ▶ When duplexing to a data space, be aware that the amount of data, therefore, storage used, in the data space is impacted by the frequency of structure checkpoints and the size of the log structure. A low checkpoint frequency and large structure size can cause a high storage demand.
- ▶ Set the logger lowoffload value to 0.
- ▶ Use large blksize such as 64K for the log streams.

CQS uses the MVS logger, which in turn uses a CF structure and staging data sets. Depending on your hardware configuration, this might not always be possible, as we have discussed with other structures. If you must use staging data sets or if it is even a possibility in the case of a configuration change, make sure these data sets are on the best possible performing DASD, because they are even more impacted than the WADS.

The amount of data written to the MVS log can be estimated by looking at the IMS 01 and 03 log records. There are other records as well, but these make up the majority of the volume.

Set your logger lowoffload value to 0.

11.9.7 FF scheduling differences

In a shared queues environment, IMS currently has no easy way to know how many messages are on the shared queue. Because of this, the standard PARLIM rules cannot be applied and something called *false scheduling* can occur. The best way to reduce the cost of false scheduling is to try to avoid scheduling in the first place. Here are some tips:

- ▶ Use (P)WFI when possible to avoid scheduling.
- ▶ Put high volume transactions in specific classes.
- ▶ Specify MAXRGN, especially for those transactions that are not in unique classes.
- ▶ Use PARLIM > 1. This makes IMS less reactive and potentially reduces scheduling. Take the average processing time for the transaction into account to be sure this would not cause unnecessary delays however.

11.9.8 FP Parallel Sysplex processing options

The DBFHAGU0 exit routine can be used to set a Parallel Sysplex processing code, which can impact how CQSS process work in a shared queues environment. There are three possible options:

- ▶ Local first
This is the default, and most likely the best option. With this option, there is virtually no shared queues overhead in the case where the transaction can be processed on the inputting system.
- ▶ Local only
This option avoids any shared queues overhead but limits availability because there is no queue sharing at all.
- ▶ Global only
This option always places the message on the shared queue, so you get all the overhead all the time. This is not the option for best performance.

Archived

IMS On Demand performance

This chapter describes the IMS On Demand operating environment and the related best practices of integrating current business application processes across the enterprise with existing core applications.

This chapter discusses all IMS On Demand features and reviews their performance considerations.

This chapter contains the following:

- ▶ IMS connectivity solutions:
 - IMS Connect
- ▶ IMS Integration solutions:
 - IMS Simple Object Access Protocol (SOAP) gateway
 - IMS Java environment
 - IMS Java performance considerations

12.1 IMS connectivity solutions using IMS Connect

IMS Connect, now part of the IMS Version 9 base product, supports communications between multiple TCP/IP clients and multiple IMS systems. IMS Connect communicates to IMS using IMS Open Transaction Manager Access (OTMA). OTMA can be implemented only in a z/OS sysplex-capable environment, because it requires the services of z/OS Cross System Coupling Facility (XCF).

OTMA is a high-performance protocol designed to operate through XCF with speeds almost comparable to main memory. The support of large networks is handled through the use of OTMA clients.

A review of the architecture of IMS Connect is needed in order to define some of the performance objectives. Figure 12-1 describes the IMS Connect configuration and some of the parameters that are required to configure IMS Connect.

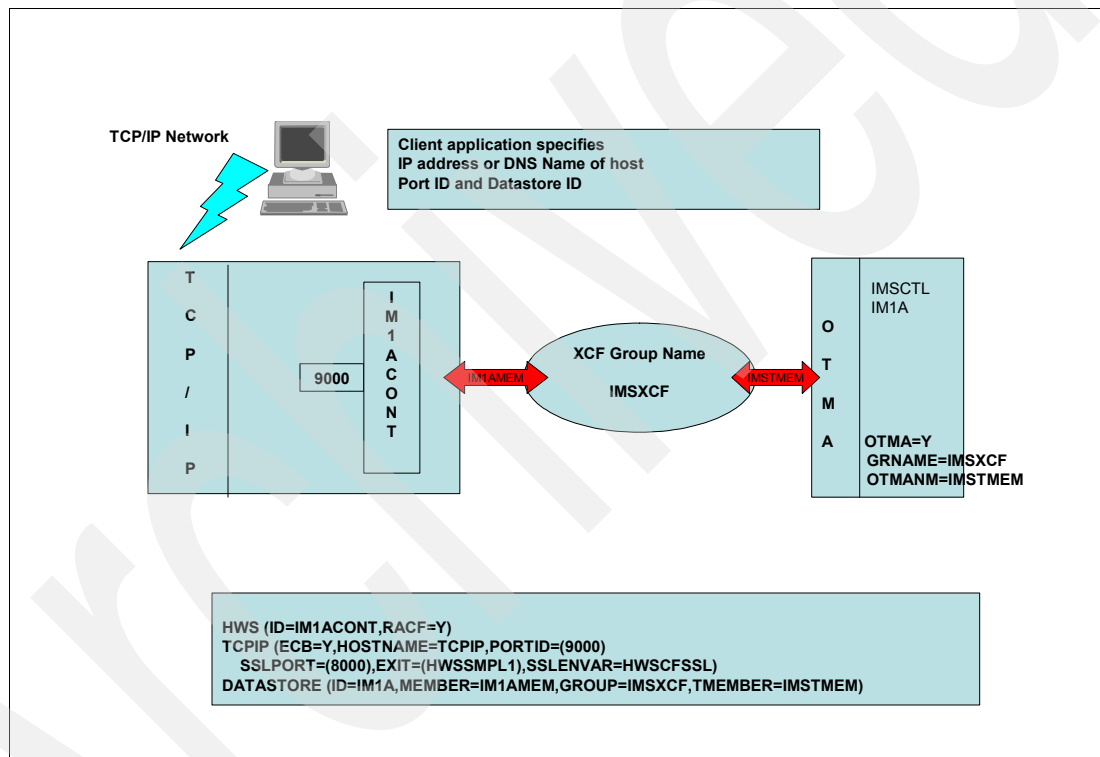


Figure 12-1 Overview of the IMS Connect configuration

In looking at the configuration of your TCP/IP network, a number of considerations must be understood in order to customize IMS Connect to site specific requirements. There are some restrictions that need to be understood:

- ▶ Each IMS Connect address space can have a maximum of 50 ports defined. Ports are defined using the PORTID definition on the TCP/IP statement. For SSLPORT, a maximum of one port can be used and it must not conflict with any ports that are previously defined. If more than one port is used, unpredictable results can occur.
- ▶ If more than one SSL port is required, z/OS 1.7 Communication Server supports multiple SSL ports. Multiple ports are then defined as part of the standard PORTID definition in IMS Connect configuration member. This means that SSL is not used within IMS Connect, but through TCP/IP instead. In this case, the IMS Connect configuration member does not have SSLPORT and SSELENVAR coded.

- ▶ Each port has a maximum of 65 535 connections of which one is always a listener in the TCP/IP address space.
- ▶ If you set MAXSOC, then you are limited to MAXSOC minus the number of listeners for ports you have activated.

12.1.1 Socket types

IMS Connect supports three types of client connection protocols, which are called *sockets*. The three socket types are:

- ▶ Persistent socket remains up for the duration that the client remains connected. It is only disconnected by either IMS Connect or the client.
- ▶ Transaction socket remains connected for a single connection or IMS conversation. This connection can be terminated only by IMS Connect.
- ▶ Non-persistent socket maintains a socket for a single input-and-output pair. IMS Connect terminates the connection after sending the output.

12.1.2 Asynchronous output processing

IMS Connect supports asynchronous output processing by allowing messages to be queued to an output tpipe destination. The client is required to connect using the clientid as the tpipe name. The clientid is part of the IRM header associated with starting up the socket connection.

Table 12-1 shows the various socket types and where they can be used. Note that the IMS Connector for Java supports only persistent socket types, including asynchronous output processing.

Table 12-1 *Socket types and their uses*

	Persistent	Transaction	Non-persistent
Send-then-commit	Available	Available	Available
Commit-then-send	Available	Available	Not available
Connector for Java	Available	Not available	Not available
Asynchronous output	Available	Available	Not available

Performance considerations for IMS Connect

In deciding how to set up and implement IMS Connect, an understanding of your network architecture is required to determine the following:

- ▶ Your firewall rules with regards to inactivity time-outs. An understanding of the firewall rules in determining what would happen if a firewall times out a TCP/IP connection due to inactivity.
- ▶ Your network resilience methodology. An understanding of the rules in place for dynamic management of connectivity to TCP/IP.
- ▶ What methodology is used to load balance TCP/IP traffic across your network.
- ▶ Bandwidth issues with regard to the size of messages flowing through the network.
- ▶ TCP/IP performance options that have been enabled. These can be obtained from your TCP/IP network specialist.

Some of the performance considerations are:

- ▶ Plan on designing the number of IMS Connect address spaces in relation to your resilience and connectivity strategy.
- ▶ Depending on your load balancing requirements, you might need to review whether you would like to use Sysplex Distributor as a means to ensure resilience.
- ▶ Deciding on Virtual IP addresses (VIPA) allows for high availability of the TCP/IP stack.
- ▶ Ensure that ECB=Y is specified on your IMS Connect address space. This allows TCP/IP to post an ECB into IMS Connect making performance better.
- ▶ Ensure that MAXSOC is high enough for maximum concurrent sessions. Member BPXPRMxx in SYS1.PARMLIB must have the following to match IMS Connect requests:
 - MAXSOCKETS must be set to the total number of active sockets per stack.
 - MAXFILEPROC must be set to the total number of socket descriptors per stack.
- ▶ Set IPV6=Y to allow for better performance even if the network is not at IPV6 level.
- ▶ If you are using the IMS Connector for Java, only persistent sockets are supported. Connection pooling plays an important role if you are going to use the connector for high volumes. Ensure that you are using managed connections to allow for best optimization of the connection factory. For asynchronous output processing using IMS Connector for Java, persistent sockets are used and the settings for the properties associated with the connector object need to be configured to your requirements.
- ▶ If you are building your own sockets, use persistent sockets for traditional transactional processing. This prevents the overhead of issuing TCP/IP connects and disconnects. For asynchronous output processing, transaction sockets are recommended, because they stay up for the duration of the timer value specified in the IRM header. IMS Connect terminates the connection and cleans up the control blocks to prevent duplicate clients when the connection is started.
- ▶ TCP/IP considerations with IMS Connect:
 - On the client TCP/IP environment, ensure:
 - TCPNODELAY=DISABLE. This allows optimization of transmission but depends on the client environment. Allows for multiple writes and waits for the buffer to be filled before sending.
 - SO_Linger=Y,VALUE=10 ensures no loss of data. The close of the socket is blocked until ACK is received or 10 seconds, whichever comes first.
 - In PROFILE.TCPIP configuration on the mainframe, ensure:
 - IMS Connect PORT set to NODELAYACKS. This allows ACKS to be sent immediately.
 - Specify SHAREPORT, which allows IMS Connect PORTS to be shared by multiple IMS Connect instances on the same stack.
 - TCPCONFIG INTERVAL or KEEPALIVEOPTIONS INTERVAL allows TCP/IP to maintain a connection that can be inactive for long periods of time.
 - SOMAXCONN must be defined large enough for maximum concurrent connections.
- ▶ IMS Connect makes use of XCF signalling to send messages to IMS. The XCF group must be isolated from the other groups and given its own set of buffers. The buffers must be large enough to accommodate your peak message flows.

More detailed information about IMS Connect and the various implementation options can be found in *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794.

12.1.3 Performance statistics on IMS Connect

The latest performance statistics are based on IMS Connect 9.1, which is now part of the IMS Version 9.1 base product. The performance was conducted comparing SNA as opposed to TCP/IP. The setup configuration for IMS Connect was as follows:

- ▶ RACF was enabled (RACF=Y).
- ▶ MAXSOC was set to 90000 (MAXSOC=90000).
- ▶ IPV6 was set to no (IPV6=N).
- ▶ ECB was set to Y (ECB=Y).
- ▶ 60000 socket connections were used.

For VTAM, the configuration was set up to allow for a maximum of 60000 SLUTYPE 2 terminals emulating 3270. Standard LU2 definitions were defined to VTAM.

Table 12-2 Results of the performance test between SNA and TCP/IP

Variables measured	SNA	TCP/IP
Number of connections	60000	60000 across 50 ports
CPU busy	89%	85.2%
External throughput rate (ETR)	1 829 transactions per second	1 996 transactions per second
Internal throughput rate (ITR)	2 055 transactions per second	2 342 transactions per second
Storage above 16 MB line	146 MB	119 MB

As you can see from Table 12-2, the results of the study indicate that the ITR for TCP/IP improved by 13.9% when compared to the ITR for SNA. The ETR is the actual arrival rate and shows the processor utilized at 89% and 85.2% respectively.

ITR is a calculated figure by working out what the throughput would be if the processor is running 100%. The throughput achieved is directly related to the manner in which TCP/IP handles connectivity when compared to VTAM. The shortened path length and improved protocols affect the ITR and naturally, affect the response times end-to-end.

Storage above the line has also reduced by 17.8%, again a factor of the efficiency of TCP/IP.

12.2 IMS SOAP Gateway

IMS SOAP Gateway allows you to leverage existing IMS applications as Web services without the need for a full blown J2EE™ server. It allows for the integration of your IMS investment into a service-oriented architecture (SOA). IMS SOAP Gateway supports open standards, including SOAP/HTTP 1.1 and WSDL 1.1.

Figure 12-2 on page 216 provides an overview of the IMS SOAP Gateway and the integration with IMS Connect.

Important: No performance data was available on IMS SOAP Gateway at the time of writing this book. We discuss some of the performance considerations when implementing the IMS SOAP Gateway.

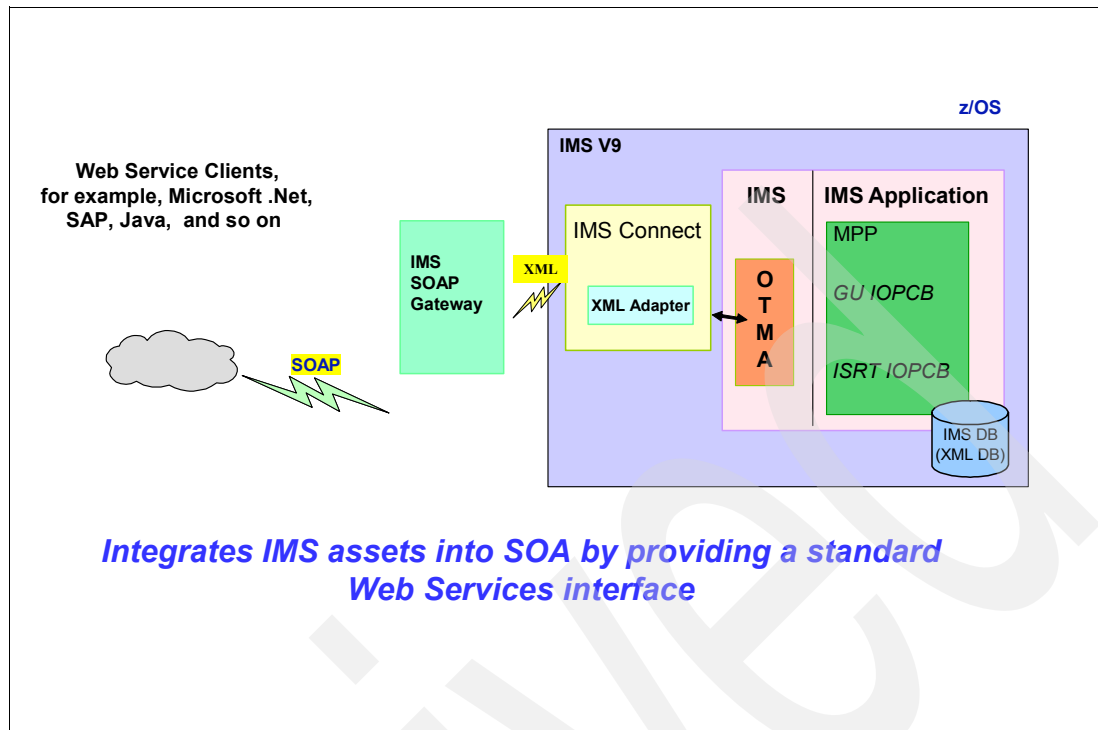


Figure 12-2 IMS SOAP Gateway overview

12.2.1 Performance considerations using the IMS SOAP gateway

In understanding the need to implement the IMS SOAP Gateway, the need to identify with a SOA type architecture is of prime importance. Typically, SOA provides the organization with a business process driven architecture, which is responsive, flexible, and consistent in its delivery of processes across the organization.

We now discuss several of the requirements for setting up IMS SOAP Gateway and a few limitations in IMS in order to address performance constraints:

- ▶ Messages can be transported to IMS as XML, or the XML adapter in IMS Connect can translate the message into the standard IMS format. When using XML as a means to transport messages into IMS, consideration must be given to the size of the XML message. Messages larger than the IMS online log data set (OLDS) blocksize would imply use of multiple buffers. This inefficiency results in performance degradation of your IMS system.
- ▶ When setting up IMS SOAP Gateway, the number of threads required to allow for concurrency must be reviewed. This is specified in the server.xml file in IMS SOAP Gateway.
- ▶ IMS SOAP gateway only supports persistent connections. To set up the connection, IMS SOAP gateway provides a deployment utility to set up a connection bundle. All that is required to set up the connection is the following:
 - Give the connection bundle a name.
 - Provide either the host name or the IP address.
 - Provide the port number.
 - Provide the datastore name.
 - Provide user ID, password, and RACF group name (all optional).

When setting up the SOAP gateway, the number of threads required to allow for concurrency must be reviewed. These are specified in the server.xml file in the SOAP Gateway. Example 12-1 shows how you define the default number of maximum threads of 150 on the non-SSL port of 8080.

Example 12-1 The contents of C:\Program Files\IBM\IMS SOAP Gateway\server\conf\server.xml

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
  <Connector port="8080"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" redirectPort="8443" acceptCount="100"
    debug="0" connectionTimeout="20000"
    disableUploadTimeout="true" />
  <!-- Note : To disable connection timeouts, set connectionTimeout value
    to 0 -->
```

12.3 IMS Java environment

The IMS Java environment can be viewed in three layers:

- ▶ The assembler layer is required, because we need to interact to the IMS assembler modules using the C interface. CEETDLI provides this interface.
- ▶ The IMS Java class library is essentially comprised of the following:
 - Java Native Interface (JNI), which enables communication between C and the Java layers.
 - The base package provides the mapping of the DL/I calls in Java.
 - The application package is for use in IMS dependent regions for transaction and message queue processing.
 - The database package provides the JDBC™ layer that make the SQL calls.
 - The XML package for storing and retrieving XML documents in a IMS hierarchy.
- ▶ The client code represents the Java application layer designed by the java developer.

Figure 12-3 on page 218 provides us with an overview. It shows the Java environment, deployment of the Java class library, and how IMS data is accessed from various sources, namely:

- ▶ Java message region (JMP) or Java batch region (JBP) with IMS transaction and message queue services
- ▶ CICS Transaction Server
- ▶ DB2 stored procedures
- ▶ WebSphere® Application Server

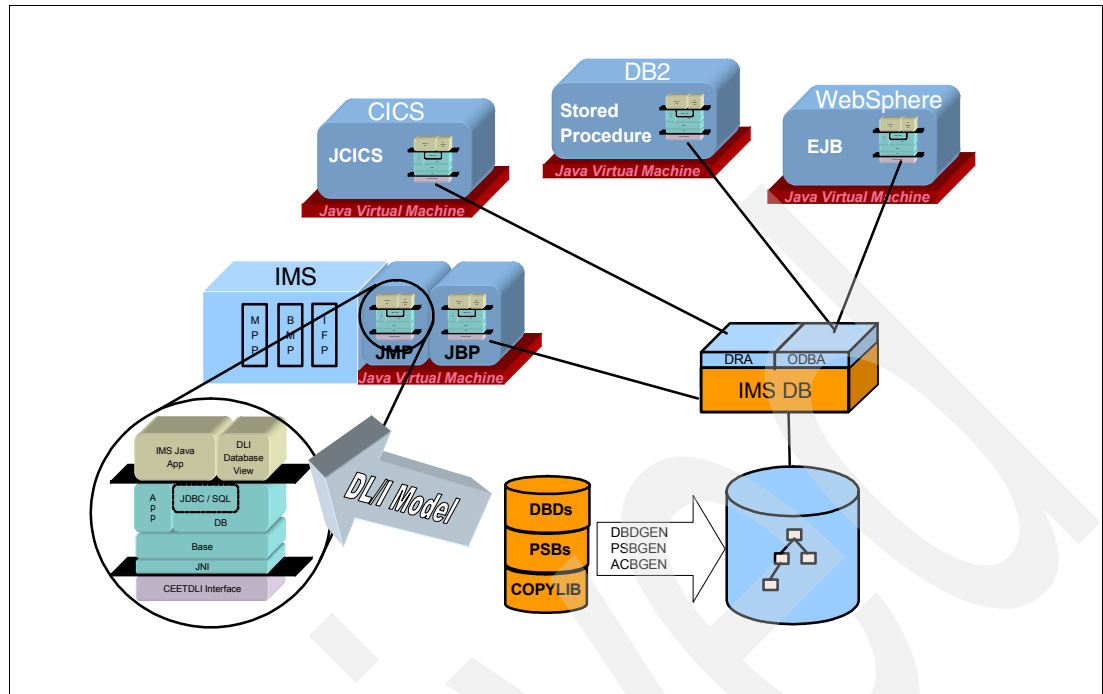


Figure 12-3 A view of the Java environment for IMS

12.3.1 IMS Java application performance considerations

Traditionally, COBOL and PL/I have been the preferred languages when writing mission critical application applications under IMS. Assembler from an application perspective is generally used in combination with COBOL applications to provide specialized interfaces in IMS and the z/OS operating system.

Java on the other hand has evolved into a widely used programming language. Today, it is a standard in teaching at universities and colleges. Java and performance need to be looked at in relation to traditional programming languages. Reasons why Java behaves differently than traditional programming languages are:

- ▶ Java Virtual Machine (JVM™) compared to a real processor. There is overhead when running a JVM. This overhead causes Java to run slower than an equivalent application written in a traditional language, such as COBOL, PL/I, or Assembler.
- ▶ Java is object-oriented, offering features such as inheritance and polymorphism. This requires extra computation time to decide which of the inherited methods in the hierarchy is the one to actually call.
- ▶ Memory management and garbage collection are done automatically and the programmer does not need to worry about memory allocations or freeing memory. Both the allocation and the garbage collector, but especially the garbage collector, are expensive and consume huge resources while executing. In traditional programming environments, the programmer is responsible for allocating and freeing memory.
- ▶ Java incorporates a number of security measures while traditional programming languages require the programmer to handle security.

Java advantages, such as platform independence, memory management, powerful exception checking, built-in multithreading, and security checks all add costs in terms of an interpreter, garbage collector, thread monitors, and run-time checks. Traditional languages do not have

these costs and are cheaper to run, but they might not provide the flexibility and platform independence that is required.

12.3.2 COBOL to Java translations

Translating COBOL to Java can be achieved in three ways:

- ▶ Use of a standard COBOL to Java translator product available in the market.
- ▶ Develop your own COBOL to Java translator.
- ▶ Manual translation.

Manual translation seems to be the most appropriate, because no product supports direct translation of IMS COBOL to Java. There are however considerations when translating from COBOL to Java:

- ▶ Mappings between COBOL, Java, and IMS data types. COBOL offers a wide selection of data types, including self-defined decimal data types. IMS Java uses `java.math.BigDecimal` class to mirror these complex data types.
- ▶ COBOL allows hierarchical data structures and these cannot be mirrored with Java base data types. An object container hierarchy needs to be implemented to mirror the COBOL structure accordingly. An example of this type of structure is an SSA list. In COBOL, it is implemented as a byte array with hierarchical access. In IMS Java, SSA lists are a type of container objects, containing the appropriate SSA.
- ▶ COBOL in an IMS implementation requires the appropriate use of a PCB. With IMS Java, this happens implicitly and is hidden from the programmer.

Table 12-3 provides an overview of important mappings between COBOL data types and Java data types. However, data type mapping remains a complex problem, because there are many special cases to be considered for PACKEDDECIMALs and ZONEDDECIMALs.

Table 12-3 Data types and conversion in different environments

COBOL type	Digits (Bytes)	IMS Java and SQL type	Java type
PIC X		CHAR	<code>java.lang.String</code>
PIC 9 BINARY	4 (2)	SMALLINT	<code>short</code>
PIC 9 BINARY	9 (4)	INTEGER	<code>int</code>
PIC 9 BINARY	18 (8)	BIGINT	<code>long</code>
COMP-1	(4)	FLOAT	<code>float</code>
COMP-2	(8)	DOUBLE	<code>double</code>
PIC9 COMP-3	(16)	PACKEDDECIMAL	<code>java.math.BigDecimal</code>
PIC9 DISPLAY		ZONEDDECIMAL	<code>java.math.BigDecimal</code>

12.3.3 Performance statistics comparing COBOL to Java

This section shows the performance characteristics of the IMS Java Message Processing region support when compared to an equivalent IMS application program using COBOL. An evaluation including COBOL was deemed valuable for purposes of application planning and design considerations.

The tests were conducted on the following hardware and software levels:

- ▶ IMS Version 9
- ▶ IMS Connect 2.2
- ▶ Java JVM (SDK 1.3.1)
- ▶ IBM zSeries z990
- ▶ z/OS 1.5
- ▶ IEEE floating point support

At the time of writing this book, new levels of hardware and software have evolved. These results do not take these new features into account. They are:

- ▶ Java JVM (SDK 1.5.0)
- ▶ z/OS 1.7
- ▶ IBM zSeries Z9
- ▶ zAAP processor

The test environment profile was as follows:

- ▶ Light transactions with 10 database calls per transaction.
- ▶ Heavy transactions with 400 database calls per transaction.
- ▶ IMS Connect with CM1 and synclevel=none.
- ▶ Cobol applications were rewritten in Java.
- ▶ Performance comparison was between a JMP and a MPP.

In reviewing these performance results, we need to understand the differences between Java and COBOL as highlighted in 12.3.1, “IMS Java application performance considerations” on page 218. We also need to consider the fact that this benchmark is based on a rewrite of the COBOL application to Java.

Table 12-4 summarizes the throughput and CPU time per transaction on both the light and heavy profile transactions in a non-sysplex environment.

Table 12-4 Comparison between COBOL and Java

Light and heavy transaction in COBOL	Light and heavy transaction in Java
2 428.79 transactions per second	1 617.59 transactions per second
180.34 CPU time per transaction	577.4 CPU time per transaction
125.12 transactions per second	83.82 transactions per second
3 204.92 CPU time per transaction	11 882.61 CPU time per transaction

Figure 12-4 on page 221 shows the differences when we run the light transactions showing both the throughput and CPU time per transaction consumed by the workload in a non-sysplex environment.

Light Transaction non-Shared Q, non-Sysplex

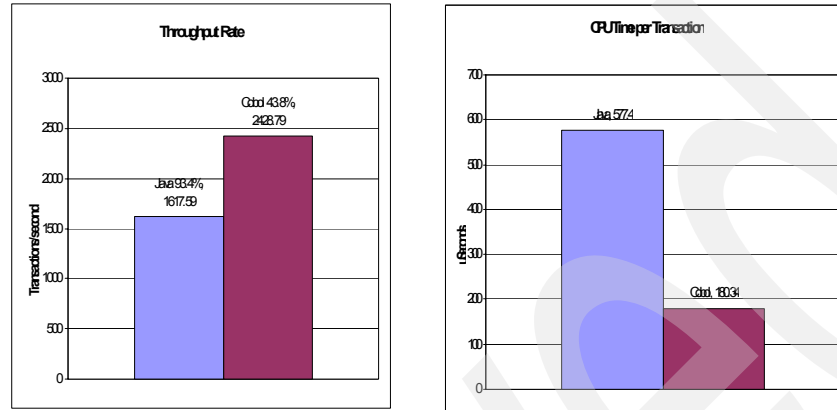


Figure 12-4 Performance benchmark showing light transaction throughput and CPU

Figure 12-5 shows the differences when we run the heavy transactions showing both the throughput and CPU time per transaction consumed by the workload.

Heavy Transaction non-Shared Q, non-Sysplex

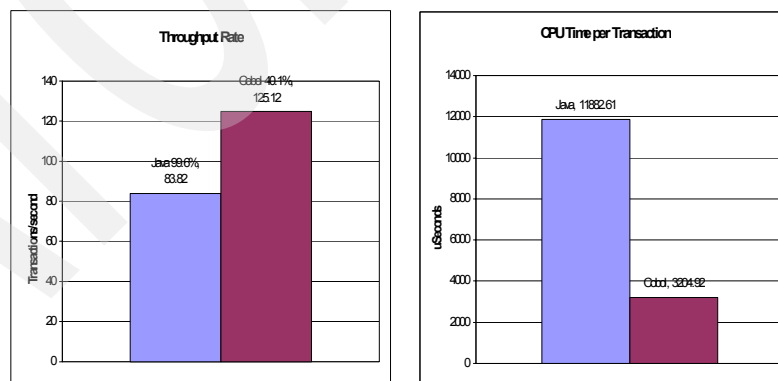


Figure 12-5 Performance benchmark showing heavy transaction throughput and CPU

The throughput in Java when compared to COBOL does show a decrease, but it is deemed reasonable due to the Java implementation. What is notable is the huge increase in CPU time

per transaction. A simple light workload shows a 2.2 times increase in CPU time when compared to a heavy workload, which shows a 2.7 times increase in CPU.

The original benchmark performed in 2003 using IBM zSeries Z900, IEEE floating point emulation, IMS Version 7, and JDK™ 1.3.1 without all performance fixes showed a 13.3 times increase in CPU. The Java environment is improving efficiency with improved JDKs. With newer JDKs and zAAP processors, the cost of running Java applications is significantly decreasing.



Guidelines and recommendations

This appendix contains guidelines in bulleted form from detailed information in this book.

A.1 First step

The first step in any performance and tuning effort is the evaluation process. We list below an evaluation process that you can use in any tuning effort:

1. Define reasonable, measurable objectives.
2. Examine the application design performance objectives and measure the application.
3. Have the objectives been met?
 - a. Yes, document the status quo.
 - b. No, go on to the next step.
4. Identify problem areas.
5. More information required?
 - a. Yes, select appropriate tool.
 - b. No, go on to the next step.
6. Take corrective actions.
7. Re-measure the application.
8. Go back to step 3 above.

The performance objectives can be either throughput-oriented or response time-oriented. But, they must be very precise. Do not say that response time must be less than a second. Say that response time must be five-tenths of a second or less for program XYZ as measured by the IMS DC Monitor run at 0830 hours, system time, on Wednesday of a work week. Update objectives as more information becomes available and as workload changes.

A.2 Choosing an IMS access method

To choose an IMS access method:

- ▶ What type of processing is done (Choices are shown in preferred order)?
 - Direct: Use DEDB, HDAM, HIDAM, or HISAM.
 - Sequential: Use DEDB (Seq Rand), HDAM (Seq Rand), HIDAM, or HISAM.
 - Both: Use DEDB (Seq Rand), HDAM (Seq RAND), or HIDAM.
- ▶ Is the data volatile? Yes, use DEDB, HDAM, or HIDAM.
- ▶ Do the database records vary in length? Yes, use DEDB, HDAM, or HIDAM.
- ▶ Are logical relationships needed? Yes, use HDAM or HIDAM.
- ▶ Are secondary indexes needed? Yes, use HDAM or HIDAM.
- ▶ Is there a need for a journaling capability? Yes, use DEDB.

Note: Wherever HDAM or HIDAM is shown, partitioning (HALDB) is preferred. Seq Rand means using a Randomizer that maintains the key sequence.

A.3 HISAM

HISAM considerations:

- ▶ One database.
- ▶ Must be VSAM.
- ▶ Two data sets.
- ▶ Primary is a KSDS.
- ▶ Secondary is a ESDS.
- ▶ Access to root is through the KSDS index component of the primary data set.
- ▶ Segments are stored physically adjacent within the VSAM record.

- ▶ One database record per VSAM record.
- ▶ Segment Prefix: Two bytes (segment code and delete byte).
- ▶ One or more database records per CI.
- ▶ CI size is determined in the define cluster, not in the DBD.
- ▶ FREESPACE is determined in the define cluster, not in the DBD.
- ▶ HISAM is difficult to use when database record lengths vary widely.
- ▶ There is no multiple data set group capability.
- ▶ Use for non-volatile database records (that is, low dependent segment insert or delete activity).
- ▶ Use fixed length segments for replace activity.
- ▶ Good for sequential processing:
 - Of database root segments.
 - Through the entire database record.
- ▶ Good for random root processing.

A.3.1 HISAM performance general guidelines

The suggested rules are:

- ▶ Only put on the pointers that are needed. The fewer the number of pointers, the less maintenance that IMS has to perform and, therefore, the faster the application.
 - VSAM record size and control interval size are key to tuning.
 - Keep 80% of the database records in the KSDS, and let the rest go into overflow.
 - Keep most accessed segments in KSDS.
 - Minimize I/O to ESDS.
- ▶ To insert dependents:

Define the VSAM CI to be larger than the average database record length at initial load. This leaves unused space in the VSAM CI, which can now be used at update time to insert dependent segments for a given root.
- ▶ To insert new roots:
 - Leave KSDS free space for inserts of a new database records after initial load.
 - If you are always inserting roots whose keys are higher than the highest key in the data set, do not leave free space in the CI or CA.

A.4 HDAM

HDAM considerations are:

- ▶ One database.
- ▶ One to 10 data sets (with data set groups).
- ▶ Database can be OSAM or VSAM.
- ▶ One or more database record per “VSAM record.”
- ▶ Segment Prefix: Segment code, delete byte, and pointers.
- ▶ CI size is determined in the define cluster, not in the DBD if VSAM.

- ▶ FREESPACE is determined in the Primary DBD, not the define cluster.
- ▶ Access to root is through a hashing routine. Typical routines (such as the recommended DFSHDC40) make consecutive processing non-sequential by key. Sequential randomizers can be built to cause consecutive processing in key order.
- ▶ Insert of segments uses free space in the most desirable block if possible.
- ▶ FREESPACE is determined in the define cluster, not in the DBD for secondary indexes along with CI size.
- ▶ Good for key read access to:
 - Database roots
 - Segments within a database record
- ▶ Synonyms are chained (PTF) in ascending key sequence if the keys are unique; otherwise, they are chained according to the insert rule.
- ▶ Use less than a 70% fill factor for HDAM databases that are going to have inserts to them. Use a 90% fill factor for HDAM databases that are read only.

A.4.1 HDAM performance general guidelines

The suggested rules are:

- ▶ Only put on the pointers that are needed. The fewer the number of pointers, the less maintenance that IMS has to perform and, therefore, the faster the application.
- ▶ Keep segments to be accessed in the same block as the entry segment.
- ▶ Do not code any FREESPACE (unless for a secondary data set group that does not contain the root).
- ▶ Do not code Twin-Backward pointer on the root.
- ▶ Code SCAN=0.
- ▶ Code SEARCHA=1.
- ▶ Unsequenced segments:
 - Use insert rules as follows:
 - (,FIRST)
 - (,LAST) with Physical Child Last pointer
 - Sequence segments only if required.
 - Use “9s complement” if adding higher keys.
- ▶ Root Addressable Area:
 - Packing factor:
 - General use 60 to 70%
 - Inquire only: Greater than 70%
 - Heavy insert activity: Less than 60%
 - Blocks and Root Anchor Points go together.
- ▶ Root Anchor Point.
- ▶ 1.2 roots per active RAP or less.
If the average database record size is greater than the block or CI size, then use one RAP per block or CI.

- ▶ HD databases do a Format Logical Cylinder, so always allocate on full cylinder boundaries.
- ▶ Different data set groups have separate bitmaps. Therefore, if there are long segments in the database, by using different data set groups, you would be able to insert the smaller segments if space was available.
- ▶ If a database is going to be HDAM, root only, no indexes, and very stable (that is, no inserts, no deletes, and no replaces), then go with SHISAM, it outperforms HDAM.
- ▶ Use the byte limit count only if you have a known performance problem.

A.5 HIDAM

HIDAM considerations are:

- ▶ Two databases (logically related).
- ▶ Two data sets at minimum (up to eleven with data set groups).
- ▶ Primary database can be OSAM or VSAM.
- ▶ Primary index database is a VSAM KSDS.
- ▶ One VSAM Record per CI.
- ▶ One or more database record per “VSAM record.”
- ▶ Access to root is through the index database.
- ▶ Access to dependent segments is through pointers.
- ▶ Segment Prefix: Segment code, delete byte, and pointers.
- ▶ Default pointers are Physical Twin Forward and Physical Child First.
- ▶ CI size is determined in the define cluster, not in the DBD if VSAM.
- ▶ FREESPACE is determined in the Primary DBD, not the define cluster.
- ▶ FREESPACE is determined in the define cluster, not in the DBD for primary and secondary indexes along with CI size.
- ▶ In HIDAM, on an open of an empty primary index, IMS inserts a high values record.
- ▶ Index database requires additional I/O.
- ▶ Good for sequential access to database roots (index).
- ▶ Good for random access to:
 - Database roots.
 - Segments within a database record.
- ▶ Use for volatile database records:
 - Insert activity: Look for FREESPACE, do not move other blocks.
 - Delete activity: Deleted space can be reused.
- ▶ Good for database records whose lengths vary widely.

A.5.1 HIDAM performance general guidelines

The suggested rules are:

- ▶ Only put on the pointers that are needed. The fewer the number of pointers, the less maintenance that IMS has to perform and, therefore, the faster the application.

- ▶ HD databases do a Format Logical Cylinder, so always allocate on full cylinder boundaries.
- ▶ Keep segments to be accessed in the same block as the entry segment.
- ▶ FREESPACE should be large enough to insert the largest segment type, including its prefix.
- ▶ Code large block sizes, such as 18 K or 26 K.
- ▶ Code SCAN=0.
- ▶ Code pointer of Twin-Backward or No-Twin on the root (reduces lock conflicts on inserts).
- ▶ If you are coding FREESPACE, code free blocks and use the second-most desirable block, code SEARCHA=2.
- ▶ Unsequenced Segments:
 - Use insert rules as follows:
 - (,FIRST)
 - (,LAST) with Physical Child Last pointer
 - Sequence segments only if required.
 - Use “9s complement” if adding higher keys.
- ▶ Different data set groups have separate bitmaps. Therefore, if there are long segments in the database, by using different data set groups, you are able to insert the smaller segments if space was available.

A.6 OSAM

OSAM considerations are:

- ▶ Maximum of 60 DASD extents.
- ▶ Maximum data set size:
 - 8 gigabytes for HIDAM and HDAM.
 - 4 gigabytes for everything else.
- ▶ Fixed length records, unblocked.
- ▶ Data set allocation:
 - Can be done at either load time through JCL or preallocated.
 - DCB parameters must not be specified.
 - Include secondary space if data set is going to expand.
- ▶ Only preallocate the number of volumes for data set extents that are used during initial load or reload process.
- ▶ Do not reuse multivolume data set extents without first scratching and reallocating the space.
- ▶ OSAM sequential buffers are dynamically allocated by IMS in the region's extended private address space, above the 16 megabyte line.

A.6.1 OSAM tuning general guidelines

The suggested tuning rules are:

- ▶ 60% or better hit ratio.

- ▶ Always page fix buffer prefixes.
- ▶ Page fix buffers if real storage permits.
- ▶ Isolate high activity databases in their own subpool.
- ▶ Match block size with a valid OSAM buffer size.
- ▶ For sequential-read BMPs and batch:
 - Use the largest block size possible.
 - Use sequential buffering.
 - Use cache controller.
- ▶ OSAM sequential buffering treats overflow and the RAA of a HDAM database differently. Sequential buffering looks at overflow to see if it is needed.

A.7 VSAM tuning general rules

The following are good rules to use:

- ▶ For sequential processing, use the largest block size possible.
- ▶ Cache IMS indexes.
- ▶ 90% or better hit ratio for index component of a KSDS.
- ▶ 60% or better hit ratio for data component of a KSDS or ESDS.
- ▶ Use VSAM Background Write.
- ▶ KSDS:
 - Cache the data set.
 - Support with HIPERSPACE.
 - Index Component:
 - Separate buffer subpool from data component.
 - Keep in main storage.
 - Allocate to an index subpool.
 - Page fix (FIXINDEX=YES).
 - Data Component:
 - Separate buffer subpool from index component.
 - Allocate to an data subpool.
 - Page fix (FIXDATA=YES) if real storage allows.
- ▶ Match CI size with a valid IMS/VSAM buffer size.
- ▶ Dedicate VSAM subpools to specific databases.
- ▶ Page fix control blocks (and buffers, if enough real storage).
- ▶ If NUMBER OF VSAM WRITES TO MAKE SPACE IN THE POOL from the DC Monitor VSAM Buffer Pool report is not zero, increase the number of buffers for the subpool.
- ▶ Track SCHBFR CALLS from the DC Monitor VSAM Buffer Pool report; are they increasing? Then it is time to reorganize.
- ▶ Verify BMP checkpoint frequency.
- ▶ VSAM takes a complete CI to satisfy FREESPACE, not part of one.

- ▶ IMS always uses one CI.
- ▶ The record size of a KSDS must be even.
- ▶ Put data and index data sets of a KSDS on different volumes.
- ▶ VSAM overhead for IMS indexes (if there are going to be more than one record in the CI, which is almost always) is 10 bytes.
- ▶ VSAM overhead for ESDS for HIDAM and HDAM is 7 bytes.
- ▶ A VSAM ESDS used with IMS does not use FREESPACE, do not code it on the VSAM delete define. Code the FREESPACE in the DBD.
- ▶ If you are allocating a VSAM data set using tracks, code the primary and secondary allocation the same.
- ▶ There should only be at most three levels in the VSAM index. If not, change the index CI size.

A.7.1 VSAM data set tuning general guidelines

VSAM data set tuning general rules are:

- ▶ Cache IMS indexes.
- ▶ VSAM takes a complete CI to satisfy free space, not part of one.
- ▶ IMS always uses one CI.
- ▶ The record size of a KSDS must be even.
- ▶ Put data and index data sets of a KSDS on different volumes.
- ▶ VSAM overhead for IMS indexes (if there are going to be more than one record in the CI, which is almost always) is 10 bytes.
- ▶ VSAM overhead for ESDS for HIDAM and HDAM is 7 bytes.
- ▶ A VSAM ESDS used with IMS does not use free space, do not code it on the VSAM delete define. Code the free space in the DBD.
- ▶ If you are allocating a VSAM data set using tracks, code the primary and secondary allocation the same.
- ▶ There should only be at most three levels in the VSAM index. If not, change the index CI size.

A.7.2 ESDS performance guidelines

ESDS performance guidelines are:

- ▶ Use IMS FRSPC=(fbff, fspf) for HIDAM ISRT.
- ▶ Larger CIs:
 - Improve sequential processing.
 - Reduce number of IWAITS.
 - Increase IWAIT time per IWAIT.
 - Decrease total IWAIT time.
- ▶ Smaller CIs:
 - Improve random processing.
 - No longer provides much benefit, because current disk devices read large blocks of data anyway.
 - Can increase number of IWAITS.
 - Reduce IWAIT time per IWAIT.
- ▶ Consider CI “fit” to DASD track.

A.7.3 KSDS performance guidelines

KSDS performance guidelines are:

- ▶ Place behind cache.
- ▶ Use DASD fast write.
- ▶ Minimize levels of index.
- ▶ Minimize CI split activity.
- ▶ Keep CA splits below four between reorganizations.
- ▶ Place index and data:
 - On separate volumes/paths.
 - In separate buffer subpools.
 - Entirely in memory (HYPERSPACE).
- ▶ Page fix index buffers.
- ▶ Page fix data buffers if memory allows.
- ▶ Consider CI “fit” to DASD track.
- ▶ Keep index components of a KSDS as available as possible.
- ▶ KSDS index buffer should be fix index, fix data, fix block.
- ▶ KSDS data buffer should be fix block.

A.8 Secondary index performance guidelines

Secondary index performance guidelines are:

- ▶ Use for alternate entry only.
- ▶ Use direct pointers.
- ▶ Avoid volatile source segments.
- ▶ Avoid volatile search and SUBSEQ fields.
- ▶ Use sparse indexing when possible.
- ▶ “Force” unique keys using SUBSEQ.
- ▶ Process a maximum of 10% of the target database records within a single synchronization interval.
- ▶ Use duplicate data and process just the index.
- ▶ Use sparse indexes if possible.
- ▶ If you have an index rebuilding tool:
 - Do not back up your indexes.
 - Register your indexes to DBRC as nonrecoverable.

A.9 Block or CI size performance guidelines

Block or CI size performance guidelines are:

- ▶ Larger CIs or blocks:
 - Improve sequential processing.
 - Reduce the number of IWAITS.
 - Increase IWAIT time per IWAIT.
 - Decrease total IWAIT time.
- ▶ Smaller CIs or blocks might:
 - Improve random processing.
 - Increase number of IWAITS.
 - Reduce IWAIT time per IWAIT.

A.10 FREESPACE performance guidelines

FREESPACE guidelines are:

- ▶ Purpose: Minimize CI/CA split activity:
 - No additions: No need for FREESPACE.
 - Few additions: No FREESPACE or some FREESPACE in the CI.
 - Evenly distributed additions: FREESPACE in the CA or FREESPACE in both CI and CA.
 - Unevenly distributed additions: Specify a small amount of FREESPACE.
- ▶ FREESPACE cost:
 - Additional DASD space which might remain unused.
 - Additional I/O to sequentially process the same number of records.
 - Additional number of index records (levels).

A.11 Segment edit/compression performance considerations

The considerations are:

- ▶ Improves DASD space utilization (more data in block)
- ▶ Improves buffer space utilization
- ▶ Might reduce I/O
- ▶ Increases CPU time unless you are using Hardware Data Compression

A.12 Programming performance considerations

The considerations are:

- ▶ Reduce the number of DLI calls:
 - Use path calls.
 - Eliminate redundant calls.
 - Use single segment input messages.
 - Send single segment output messages.
- ▶ Reduce the number of I/O IWAITS:
 - Use fully qualified calls.
 - Use XDFLD name in SSA when PCB has PROCSEQ.
 - Do not use PROCOPT=GOT, use PROCOPT=GON.

- ▶ Reduce lock contention.
Use the minimum PROCOPT needed for processing.

A.13 Logging performance considerations

Performance reflected in DLI-NOT-IWAIT time:

- ▶ Make database buffer subpools large (avoid LWA due to buffer flush).
- ▶ Use VSAM background write.
- ▶ Define databases as unrecoverable.
- ▶ Use 24 K or 28 K blocking. This allows the OLDS buffers to be placed in real storage above the 2 GB bar (by being multiples of 4 K).

A.14 Use HDAM physical sequence sort and reload

HDAM physical sequence sort and reload:

- ▶ Functions:
 - Sorts an unload database into load sequence.
 - Reduces cascading by loading some records last.
 - Provides tuning analysis reports:
 - DBD parameters and overrides.
 - Assigned roots per RAP.
 - Provides database record size and distribution information.
- ▶ Benefits:
 - Reduces elapsed time to reload.
 - Converts to HDAM.
 - Changes HDAM parameters.
 - Analyzes HDAM tuning parameters.

A.15 I/O error processing

When I/O errors occur for databases, the following things happen:

1. The database that has an I/O error *is NOT stopped*.
2. An Extended Error Queue Element (EEQE) is created for the block or CI. It identifies the block or Control Interval and the type of error.
3. A DFS0451A or DFS0451I message is issued indicating that a read or write error occurred during an I/O operation. This message is only issued once for each block or CI, which has an I/O error.
4. For read errors:
 - a. An “A0” status code is returned for the DL/I call.
 - b. The read is retried for each future request for the block or CI.
5. For write errors:
 - a. No status code is returned.
 - b. The block or CI is moved to a virtual buffer for future reference.
 - c. If the database is registered to DBRC. The DBDS record in the RECON is updated with the error information (EEQE), the “recovery needed” flag in the DBRC DBDS record is

turned on, and the “recovery needed” counter in the DBRC DB record is incremented. Further subsystem authorization is NOT denied.

- d. The virtual buffer is logged when it is created or changed and at each system checkpoint.
- e. Any read request for data contained in the virtual buffer is read from it. DASD is not accessed.
- f. Any write requests for blocks with previously allocated buffers result in a move of the database buffer to the virtual buffer. DASD is not accessed.

A.16 What is a Kilobyte

BIT	=	BIInary digiT a 0 or 1
Nibble	=	4 BITs
Byte	=	8 BITs
Kilobyte	=	8,192 BITs
Megabyte	=	8,388,608 BITs
Gigabyte	=	8,589,934,592 BITs
Terabyte	=	8,796,093,022,208 BITs
Petabyte	=	9,007,199,254,740,992 BITs
Exabyte	=	9,223,372,036,854,775,808 BITs
Zettabyte	=	9,444,732,965,739,290,427,392 BITs
Yottabyte	=	9,671,406,556,917,033,397,649,408 BITs
Xonabyte	=	9,903,520,314,283,042,199,192,993,792 BITs
Wekabyte	=	10,141,204,801,825,835,211,973,625,643,008 BITs
Vundabyte	=	10,384,593,717,069,655,257,060,992,658,440,192 BITs

Kilobyte	=	1,024 bytes or 2^{10}
Megabyte	=	1,048,576 bytes or 2^{20}
Gigabyte	=	1,073,741,824 bytes or 2^{30}
Terabyte	=	1,099,511,627,776 bytes or 2^{40}
Petabyte	=	1,125,899,906,842,624 bytes or 2^{50}
Exabyte	=	1,152,921,504,606,846,976 bytes or 2^{60}
Zettabyte	=	1,180,591,620,717,411,303,424 bytes or 2^{70}
Yottabyte	=	1,208,925,819,614,629,174,706,176 bytes or 2^{80}

Xonabyte = 1,237,940,039,285,380,274,899,124,224 bytes or 2^{90}
Wekabyte = 1,267,650,600,228,229,401,496,703,205,376 bytes or 2^{100}
Vundabyte = 1,298,074,214,633,706,907,132,624,082,305,024 bytes or 2^{110}

Archived

Archived



Coding examples

This appendix provides two examples for coding sparse indexes.

B.1 Sparse index examples

There are two ways to suppress entries into the secondary index. One way is on the source segment. With this technique, some field in the source segment is compared to a constant and if they match, an index entry is not created. The second method compares some field in the secondary index fields themselves to find a match. If they match, then an index is not created.

B.1.1 Source segment

With this technique, some field in the source segment is compared to a constant and if they match, an index entry is not created.

Example: B-1 Source segment technique

```

      1      2      3      4      5      6      7
1234567890123456789012345678901234567890123456789012345678901
|
TITLE 'sysAA1er INDEX MAINTENANCE ROUTINE'
*****
*              A P P L I C A T I O N   N A M E              *
*
*      THIS MODULE IS CALLED BY DL/I FROM THE sysDaaP1 DATABASE. *
*      IF THE ACTIVITY_CODE IS EITHER 'A' OR 'T' CREATE AN INDEX *
*      ENTRY IN INDEX sysDaaS1.                                *
*
*      REGISTER ASSIGNMENTS AT ENTRY TO SYSAA1ER:             *
*      R1  - PARTITION SPECIFICATION TABLE (PST) ADDRESS.    *
*      R2  - ADDRESS OF (PROPOSED OR EXISTING) INDEX SEGMENT.  *
*      R3  - ADDRESS OF INDEX MAINTENANCE ROUTINE PARMS SEGMENT.*
*      R4  - ADDRESS OF INDEX SOURCE SEGMENT.                  *
*      R13 - SAVE AREA ADDRESS.                                *
*      R14 - RETURN TO IMS ADDRESS.                             *
*      R15 - ENTRY POINT ADDRESS OF THE EXIT ROUTINE.         *
*
*      REGISTER ASSIGNMENTS AT EXIT FROM SYSAA1ER:            *
*      R0 THROUGH R13 ARE RESTORED.                             *
*      R14 - RETURN TO IMS ADDRESS.                             *
*      R15 - 0 TO NOT SUPPRESS THE INDEX ENTRY.                *
*      - 4 TO SUPPRESS THE INDEX ENTRY.                        *
*
*****
*LINKOPT: AMODE=31,RMODE=ANY,RENT,REUS
*
*****
sysAA1er START
*
*****
* REGISTER EQUATES
*****
*
```



```

R0      EQU    0
R1      EQU    1
R2      EQU    2
R3      EQU    3
R4      EQU    4
R5      EQU    5
R6      EQU    6
R7      EQU    7
R8      EQU    8
R9      EQU    9
R10     EQU    10
R11     EQU    11
R12     EQU    12
R13     EQU    13
R14     EQU    14
R15     EQU    15
*
*****
* SET UP THE ENVIRONMENT.                                     *
*****
*
INITIAL DS    0H
*
          STM   R14,R12,12(R13)  SAVE REGISTERS 14 THRU 12
          LR    R12,R15          LOAD R12 WITH ENTRY ADDRESS
          USING sysAA1er,R12     USE R12 AS BASE ADDR OF EXIT ROUTINE
          USING SECINDEX,R2      USE R2 AS BASE ADDR OF INDEX SEGMENT
          USING SECSOURX,R4      USE R4 AS BASE ADDR OF SOURCE SEGMENT
          LA    R15,4            LOAD 4 INTO R15 TO SUPPRESS INDEX
*
*****
* CHECK TO SEE IF WE WANT TO SUPPRESS THE INDEX ENTRY.       *
*****
*
CHKL1    DS    0H
*
          CLC   sysEaade,ATYP    IF ACTIVITY_CODE IS AN "A"
          BE    CREATESX         THEN WRITE INDEX
*                                     ELSE
*                                     CHECK SOME MORE
*                                     IF ACTIVITY_CODE IS A "T"
          CLC   sysEaade,TTYP    THEN WRITE INDEX
          BNE   WRAPUP           ELSE
*                                     DO NOT WRITE INDEX
*
*****
* CREATE INDEX ENTRY.                                         *
*****
*
CREATESX DS    0H
*
          SR    R15,R15          CREATE SPARSE INDEX (4 - 4 = 0)
*
*****
* DO NOT CREATE INDEX ENTRY.                                   *
*****

```

```

*
WRAPUP   DS    OH
*
          ST    R15,16(,R13)      STORE R15 IN SAVE REGISTER R15
          LM    R14,R12,12(R13)  RESTORE REGISTERS FOR RETURN
          SPM   R14              SET PROGRAM MASK
          BR    R14              BRANCH TO CALLING PROGRAM
          EJECT

*
*****
* PROGRAM INFORMATION AND THE LIKE.                                     *
*****
*
PGMINFO  DS    OH
*
          DC    CL8'sysAA1er'     PROGRAM NAME
          DC    CL9' &SYSDATE'    DATE ASSEMBLED
          DC    CL9' &SYSTIME'    TIME ASSEMBLED
*
*****
* DEFINE WHAT WE ARE LOOKING FOR.                                     *
*****
*
ATYP     DC    CL01'A'           THIS IS A "A" RECORD
TTYP     DC    CL01'T'           THIS IS A "T" RECORD
          EJECT
*
*****
* DEFINE WHAT THE SECONDARY INDEX SEGMENT LOOKS LIKE.               *
*****
*
SECINDEX DSECT                      SECONDARY INDEX SEGMENT
sysSaaS1 DS    0CL19              ACCOUNT ACTIVITY INDEX SEGMENT
sysEaa11 DS    CL008              FILL UP TO FIELD WE WANT
sysEaa1A DS    CL002              WORK_ACTIVITY_CODE
sysEaa12 DS    CL009              FILL UP TO THE END
*
*****
* DEFINE WHAT THE SECONDARY INDEX SOURCE SEGMENT LOOKS LIKE.        *
*****
*
SECSOURX DSECT                      SECONDARY INDEX SOURCE SEGMENT
sysSaa00 DS    0CL196             ACCOUNT ACTIVITY SEGMENT
sysEaaF1 DS    CL019              FILL UP TO FIELD WE WANT
sysEaaDE DS    CL001              ACTIVITY_CODE
sysEaaF2 DS    CL176              FILL UP TO THE END
*
*****
* EXIT                                                                *
*****
*
*
END

```

B.1.2 Index segment

This method compares some field in the secondary index fields themselves to find a match. If they match, then an index is not created.

Example: B-2 Index segment method

```

      1      2      3      4      5      6      7
1234567890123456789012345678901234567890123456789012345678901
|
TITLE 'SYSAA1ER INDEX MAINTENANCE ROUTINE'
*****
*              A P P L I C A T I O N   N A M E              *
*
*      THIS MODULE IS CALLED BY DL/I FROM THE SYSDAAP1 DATABASE. *
*      IF THE WORK_ACTIVITY_CODE IS EITHER 'A' OR 'T' CREATE AN INDEX *
*      ENTRY IN INDEX SYSDAAS1. *
*
*      REGISTER ASSIGNMENTS AT ENTRY TO SYSAA1ER: *
*      R1  - PARTITION SPECIFICATION TABLE (PST) ADDRESS. *
*      R2  - ADDRESS OF (PROPOSED OR EXISTING) INDEX SEGMENT. *
*      R3  - ADDRESS OF INDEX MAINTENANCE ROUTINE PARMS SEGMENT. *
*      R4  - ADDRESS OF INDEX SOURCE SEGMENT. *
*      R13 - SAVE AREA ADDRESS. *
*      R14 - RETURN TO IMS ADDRESS. *
*      R15 - ENTRY POINT ADDRESS OF THE EXIT ROUTINE. *
*
*      REGISTER ASSIGNMENTS AT EXIT FROM SYSAA1ER: *
*      R0 THROUGH R13 ARE RESTORED. *
*      R14 - RETURN TO IMS ADDRESS. *
*      R15 - 0 TO NOT SUPPRESS THE INDEX ENTRY. *
*           - 4 TO SUPPRESS THE INDEX ENTRY. *
*
*****
*
*LINKOPT: AMODE=31,RMODE=ANY,RENT,REUS
*
*****
*
SYSAA1ER START
*
*****
* REGISTER EQUATES *
*****
*
R0      EQU  0
R1      EQU  1
R2      EQU  2
R3      EQU  3
R4      EQU  4
R5      EQU  5
R6      EQU  6
R7      EQU  7
R8      EQU  8
```

```

R9      EQU    9
R10     EQU    10
R11     EQU    11
R12     EQU    12
R13     EQU    13
R14     EQU    14
R15     EQU    15
*
*****
* SET UP THE ENVIRONMENT.                                     *
*****
*
INITIAL DS    0H
*
          STM   R14,R12,12(R13)    SAVE REGISTERS 14 THRU 12
          LR    R12,R15            LOAD R12 WITH ENTRY ADDRESS
          USING SYSAAL1ER,R12      USE R12 AS BASE ADDR OF EXIT ROUTINE
          USING SECINDEX,R2        USE R2 AS BASE ADDR OF INDEX SEGMENT
          USING SECSOURX,R4        USE R4 AS BASE ADDR OF SOURCE SEGMENT
          LA    R15,4              LOAD 4 INTO R15 TO SUPPRESS INDEX
*
*****
* CHECK TO SEE IF WE WANT TO SUPPRESS THE INDEX ENTRY.      *
*****
*
CHKL1    DS    0H
*
          CLC   SYSEAA1A,ATYP      IF WORK_ACTIVITY_CODE IS AN "A"
          BE    CREATESX           THEN WRITE INDEX
*                                     ELSE
*                                     CHECK SOME MORE
*                                     IF WORK_ACTIVITY_CODE IS A "T"
          CLC   SYSEAA1A,TTYP      THEN WRITE INDEX
          BNE   WRAPUP             ELSE
*                                     DO NOT WRITE INDEX
*
*****
* CREATE INDEX ENTRY.                                       *
*****
*
CREATESX DS    0H
*
          SR    R15,R15            CREATE SPARSE INDEX (4 - 4 = 0)
*
*****
* DO NOT CREATE INDEX ENTRY.                                *
*****
*
WRAPUP   DS    0H
*
          ST    R15,16(,R13)        STORE R15 IN SAVE REGISTER R15
          LM    R14,R12,12(R13)    RESTORE REGISTERS FOR RETURN
          SPM   R14                SET PROGRAM MASK
          BR    R14                BRANCH TO CALLING PROGRAM
          EJECT
*

```

```

*****
* PROGRAM INFORMATION AND THE LIKE.                                     *
*****
*
PGMINFO  DS      OH
*
          DC      CL8'SYSAA1ER'      PROGRAM NAME
          DC      CL9' &SYSDATE'      DATE ASSEMBLED
          DC      CL9' &SYSTIME'      TIME ASSEMBLED
*
*****
* DEFINE WHAT WE ARE LOOKING FOR.                                     *
*****
*
ATYP      DC      CL01'A'            THIS IS A "A" RECORD
TTYP      DC      CL01'T'            THIS IS A "T" RECORD
          EJECT
*
*****
* DEFINE WHAT THE SECONDARY INDEX SEGMENT LOOKS LIKE.               *
*****
*
SECINDEX DSECT                                SECONDARY INDEX SEGMENT
SYSSAAS1 DS      0CL19                      ACCOUNT ACTIVITY INDEX SEGMENT
SYSEAA11 DS      CL008                      FILL UP TO FIELD WE WANT
SYSEAA1A DS      CL002                      WORK_ACTIVITY_CODE
SYSEAA12 DS      CL009                      FILL UP TO THE END
*
*****
* DEFINE WHAT THE SECONDARY INDEX SOURCE SEGMENT LOOKS LIKE.        *
*****
*
SECSOURX DSECT                                SECONDARY INDEX SOURCE SEGMENT
SYSSAA00 DS      0CL196                    ACCOUNT ACTIVITY SEGMENT
SYSEAAF1 DS      CL019                      FILL UP TO FIELD WE WANT
SYSEAADE DS      CL001                      ACTIVITY_CODE
SYSEAAF2 DS      CL176                      FILL UP TO THE END
*
*****
* EXIT                                                                *
*****
*
*
END

```

Archived

Abbreviations and acronyms

ACEE	Access Control Environment Element	JCA	J2EE connector architecture
AGN	Application Group Name	JCL	Job Control Language
AIB	Application Interface Block	JDBC	Java Database Connectivity
APF	Authorized Program Facility	JDK	Java Development Kit
APPC	Advanced Program to Program Communication	JRE™	Java Runtime Environment
BPE	Base Primitive Environment	JSP™	Java Server Pages
CCF	Common Connector Framework	JVM	Java Virtual Machine
CGI	Common Gateway Interface	LAN	local area network
CICS	Customer Information Control System	LPAR	logical partition
CSM	Complete Status Message	LTERM	logical terminal
CTG	CICS Transaction Gateway	LU	logical unit
DB2	DATABASE 2	LU2	logical unit 2
DBCTL	Database Control	MCI	message control information
DBD	Database Description	MFS	message format services
DRA	Database Resource Adapter	MOD	message output descriptor
DVIPA	dynamic virtual IP addressing	MPP	message processing program
EAB	Enterprise Access Builder	MSC	Multiple Systems Coupling
ECB	Event Control Block	MVS	Multiple Virtual System
EMH	Expedited Message Handler	ODBA	Open Database Access
FTP	File Transfer Protocol	OO	object-oriented
GUI	graphical user interface	OTMA	Open Transaction Manager Access
HTML	Hyper Text Markup Language	OTMA C/I	OTMA Callable Interface
HTTP	Hyper Text Transfer Protocol	PC	personal computer
IBM	International Business Machines Corporation	PCB	Program Communication Block
IMS	Information Management System	PPT	Program Properties Table
IMS TOC	IMS TCP/IP OTMA Connection	PSB	Program Specification Block
IPCS	Interactive Problem Control System	PST	Partition Specification Table
IPL	Initial Program Load	RACF	Resource Access Control Facility
IRM	IMS Request Message	RAR	resource archive
ISC	Intersystem Communication	RMM	Request MOD Message
ISPF	Interactive Systems Productivity Facility	RRS/MVS	Resource Recovery Services/MVS
ITOC	IMS TCP/IP OTMA Connection	RSM™	request status message
ITSO	International Technical Support Organization	SGML	Standard Generalized Markup Language
IVP	Installation Verification Program	SMB	scheduler message block
J2C	J2EE Connector Architecture	SMP/E	System Modification Program/Extended
J2EE	JAVA 2 Platform Enterprise Edition	SNA	System Network Architecture
		SOA	service-oriented architecture
		SOAP	simple object access protocol
		STSN	Set and Test Sequence Numbers

SVL	Silicon Valley Laboratories
TCB	task control block
TCP/IP	Transmission Control Protocol/Internet Protocol
Tpipe	transaction pipe
TPIPE	transaction pipe
UOR	unit of recovery
W3C	World Wide Web Consortium
WAN	wide area network
WAS	WebSphere Application Server
VIPA	Virtual IP Addressing
WLM	workload manager
WSDL	Web Service Definition Language
WWW	World Wide Web
XCF	cross system coupling facility
XML	Extensible Markup Language

Archived

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 251. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IMS Version 7 Performance Monitoring and Tuning Update*, SG24-6404
- ▶ *IMS e-business Connectors: A Guide to IMS Connectivity*, SG24-6514
- ▶ *Ensuring IMS Data Integrity Using IMS Tools*, SG24-6533
- ▶ *IMS Installation and Maintenance Processes*, SG24-6574
- ▶ *IMS Version 8 Implementation Guide - A Technical Introduction of the New Features*, SG24-6594
- ▶ *IMS DataPropagator Implementation Guide*, SG24-6838
- ▶ *Using IMS Data Management Tools for Fast Path Databases*, SG24-6866
- ▶ *IMS in the Parallel Sysplex, Volume I: Reviewing the IMSplex Technology*, SG24-6908
- ▶ *IMS in the Parallel Sysplex, Volume II: Planning the IMSplex*, SG24-6928
- ▶ *IMS in the Parallel Sysplex, Volume III: Operations and Implementation*, SG24-6929
- ▶ *The Complete IMS HALDB Guide, All You Need to Know to Manage HALDBs*, SG24-6945
- ▶ *Reorganizing Databases Using IMS Tools - A Detailed Look at the IBM IMS High Performance Tools*, SG24-6074
- ▶ *IMS Version 9 Implementation Guide - A Technical Overview*, SG24-6398
- ▶ *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794
- ▶ *System Programmers Guide to: Workload Manager*, SG24-6472
- ▶ *IBM eServer zSeries 990 (z990) Cryptography Implementation*, SG24-7070

Other publications

These publications are also relevant as further information sources:

- ▶ *IMS Version 9: Administration Guide: Database Manager*, SC18-7806
- ▶ *IMS Version 9: Administration Guide: System*, SC18-7807
- ▶ *IMS Version 9 Administration Guide: Transaction Manager*, SC18-7808
- ▶ *IMS Version 9 Application Programming: Database Manager*, SC18-7809
- ▶ *IMS Version 9 Application Programming: Design Guide*, SC18-7810
- ▶ *IMS Version 9 Application Programming: EXEC DL/I Commands for CICS & IMS*, SC18-7811

- ▶ *IMS Version 9 Application Programming: Transaction Manager*, SC18-7812
- ▶ *IMS Version 9 Base Primitive Environment Guide and Reference*, SC18-7813
- ▶ *IMS Version 9 Command Reference*, SC18-7814
- ▶ *IMS Version 9 Common Queue Server Guide and Reference*, SC18-7815
- ▶ *IMS Version 9 Common Service Layer Guide and Reference*, SC18-7816
- ▶ *IMS Version 9: Customization Guide*, SC18-7817
- ▶ *IMS Version 9 Database Recovery Control Guide and Reference*, SC18-7818
- ▶ *IMS Version 9 Diagnosis Guide and Reference*, LY37-3203
- ▶ *IMS Version 9 Failure Analysis Structure Tables (FAST) for Dump Analysis*, LY37-3204
- ▶ *IMS Version 9: IMS Java Guide and Reference*, SC18-7821
- ▶ *IMS Version 9 Installation Volume 1: Installation and Verification*, GC18-7822
- ▶ *IMS Version 9 Installation Volume 2: System Definition and Tailoring*, GC18-7823
- ▶ *IMS Version 9 Licensed Program Specifications*, GC18-7825
- ▶ *IMS Version 9 Master Index and Glossary*, SC18-7826
- ▶ *IMS Version 9: Messages and Codes Volume 1*, GC18-7827
- ▶ *IMS Version 9 Messages and Codes, Volume 2*, GC18-7828
- ▶ *IMS Version 9: Open Transaction Manager Access Guide and Reference*, SC18-7829
- ▶ *IMS Version 9 Operations Guide*, SC18-7830
- ▶ *IMS Version 9 Release Planning Guide*, GC17-7831
- ▶ *IMS Version 9 Summary of Commands*, SC18-7832
- ▶ *IMS Version 9 Utilities Reference: Database Manager and Transaction Manager*, SC18-7833
- ▶ *IMS Version 9 Utilities Reference: System*, SC18-7834
- ▶ *IMS Version 9: IMS Connect Guide and Reference*, SC18-9287
- ▶ *IMS Connect Extensions for z/OS User's Guide*, SC18-7255
- ▶ *IBM IMS Performance Analyzer for z/OS User's Guide*, SC18-9778
- ▶ *IBM IMS Performance Analyzer for z/OS Report Analysis*, SC27-0913
- ▶ *Program Directory for IBM IMS Connect Extensions for z/OS*, GI10-8504
- ▶ *Program Directory for IBM IMS Connect for z/OS*, GI10-8506
- ▶ *Program Directory for IBM Information Management System Transaction and Database Servers*, GI10-8594
- ▶ *A Guide to IMS Hardware Data Compression*, WP100416
- ▶ *Using IBM Tivoli OMEGAMON XE for IMS on z/OS*, GC32-9351
- ▶ *IBM Tivoli OMEGAMON II for IMS Transaction Reporting Facility*, GC32-9358
- ▶ *IMS on Demand Database Access and IMS Control Center presentation*
- ▶ *IMS and Language Environment presentation*
- ▶ *Using Log Records for IMS Diagnosis, Part 1 & 2*, B66 presentation from the IMS Technical Conference in San Jose, CA, October 10 - 13, 2005
- ▶ *IMS Database Performance and Tuning (Course Code CMW30)*
- ▶ *IMS TM Performance and Tuning (Course Code CMW21)*

- ▶ *IMS Database Application Programming* (Course Code CM17)
- ▶ *Performance Comparison between traditional Programming languages and Java on an IMS System based on a TPC Benchmark*, white paper
- ▶ *DB2 Universal Database for z/OS Administration Guide*, SC18-7413
- ▶ *New Tools for DB2 for OS/390 and z/OS Presentation Guide*, SG24-6139
- ▶ *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IMS Home Page:
<http://www.ibm.com/ims/>
- ▶ IBM Redbooks
<http://www.ibm.com/redbooks>

How to get IBM Redbooks

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

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

Symbols

/CHANGE CPLOG command 131
/CHE STATISTICS command 30
/DIS FPVIRTUAL command 96
/DIS OLDS 164
/DIS POOL ALL command 139, 147
/DIS POOL command 134, 139
/DIS POOL DMBP command 146
/DIS POOL FPDB command 99, 132
/RML RECON STATUS 164
/SSR command 195
/STA AREA command 99
/START AREA command 95
/TRA SET ON LINK n command 30
/TRA SET ON MON ALL command 139
/TRACE command 17, 21
/TRACE SET OFF MONITOR command 107, 119
/TRACE SET ON MONITOR ALL command 107, 119
/VUNLOAD command 100

Numerics

64-bit support 157

A

ACBLIB
 I/Os 5, 15, 126
 load 5
access methods 53–54, 58, 60–61, 77, 83, 115
 HIDAM 60
 HISAM 58
 OSAM 60
 VSAM 60
ACK 214
APPC 165, 207
application design xiii, 5, 53, 189
asynchronous output 213–214

B

BGWRT 117–118
BIND CURRENT DATA parameter 190
BIND ISOLATION parameter 190
BLDL 127
block mover 148
BPXPRMxx 214
buffer invalidation 177
buffer pools 16, 22, 43, 51, 82, 97–99, 131, 135, 139, 144, 152, 193, 206
 Database Work Area Pool (DBWP) 146
 DMB Pool (DMBP) 145

C

CA splits 162

capacity planning 4, 11
CFRM 204, 208
CFSIZER 205, 207
CHANGE.DB command 65, 104
CHANGE.DBDS 94
CHANGE.PART command 67, 70, 72–73
CHANGE.RECON REPLACE 163–164
CHANGE.RECON REPLACE(xxx) command 159, 162
CI splits 162
CIOP pool 6
class scheduling 15, 126, 128
CNBA 150
COBOL 127–128, 170–171, 175, 180, 218–221
 options 127, 180
COMM macro 138
contention 35, 51, 90–91, 93, 100, 104, 109, 128, 140, 144, 161, 177, 190, 203–206, 233
 buffer handler 109
conversational 28
conversations 135
Coupling Facility 91, 98–99, 207, 212
CPLOG 151
CPU resource 121, 143
CPU usage 36, 48–50, 203, 207
CSA 136, 141, 145
cursor 190, 192

D

data sharing 1, 36, 64, 90, 92, 96, 98, 100, 142, 162, 176, 203–206
DATABASE 75, 193, 238, 241
database buffers 67, 101, 141
database reorganization 115
database structures 54–56
datastore 42, 216
DB/DC 11–13, 17, 165–166
DB2 xiv, 6, 21, 24, 42–43, 48, 87, 165, 183–193, 195–200, 217
 BIND ACQUIRE parameter 189
 BIND RELEASE parameter 190
 Dynamic SQL 191
 IMS monitor 192, 195
 IMS Performance Analyzer 192
 Locking considerations 189
 SSM 185
 Static SQL 191
 Thread creation and termination 188
 Tuning the External Subsystem Attach Facility 187
DB2 Tools 193
 DB2 Performance Monitor (DB2PM) 193
 DB2 Query Monitor Tool 193
 DB2 Trace Facility 193
 SQL Performance Analyzer 193
DB2PM 193

DBBF 92, 101, 131, 150–151
 DBCTL xiv, 11, 17, 22, 24, 29–30, 32–33, 57, 101, 131, 149–153, 165–166, 176, 183–184
 DRA startup 150
 MINTHRD and MAXTHRD 150
 DBFULTA0 32, 34, 99, 102
 DBFX 101, 131, 151
 DBRC 26, 49, 57, 65, 67, 70, 72–73, 93–95, 104, 142, 155, 158–160, 162–164, 176, 231, 233–234
 commands 65, 72
 dispatching priority 142
 RECON data sets 159, 162
 DBRC SHARECTL 176
 deadlock 27, 94, 141–142, 192, 198, 203
 DB2 192
 DEADLOK 203
 DEDB 7, 33–34, 56–57, 90–97, 99–103, 131–132, 136, 147, 151, 178, 205
 general performance considerations 91
 OTHRD contention 91
 DELETE 101, 163, 192, 199
 DELETE.LOG INACTIVE 162
 dependent region 4, 13, 15, 25, 48–49, 101, 109, 126, 128, 136–137, 142–143, 147, 175, 185–186, 188, 198, 203
 DFSBSEX0 166
 DFSERA10 20, 27, 140, 192, 198
 DFSERA30 27, 192, 198–199
 DFSHDC40 72–73, 76, 226
 DFSILTA0 27
 DFSISTS0 27
 DFSPBxxx 144, 151, 165
 DFSPBxxx member 130–131, 137, 144
 DFSPSBxx member 137
 DFSPZPxx 150
 DFSUARC0 143
 DFSUTR20 21, 139
 DFSVSMxx member 24, 97, 106, 115, 120, 141, 203
 VSAMFIX 141
 dispatching priority 50, 52, 142–143
 dependent regions 142–143
 IRLM 142
 IRLM address space 142
 DL/I 3–7, 9–10, 15, 17, 22–25, 28, 33, 54, 60, 76, 90–91, 93, 118, 126–129, 136, 138–139, 141, 143, 145, 170–173, 175, 187, 189–190, 197, 217, 233, 238, 241
 DLS address space 143
 DRA 17
 DSN 35–36, 163
 dynamic allocation 148, 157, 160

E

ECB 2, 214–215
 E-MCS console 162
 EMH 6, 134
 exit 27, 63–65, 85–86, 90, 112, 124, 166, 209
 EXVR 135

F

Fast Path 1, 6–7, 9, 11, 13, 21, 26, 29, 33, 48, 53, 90–93, 97, 101, 103, 130–132, 134, 136–137, 143–146, 158, 178, 184, 188
 IMS Monitor 13, 21
 IMS PA Fast Path Transaction Exception Log report 102
 sync point processing 7, 132
 transaction flow 6
 Fast Path buffers 101, 131, 145, 178
 Fast Path transactions
 flow 6
 FIX 152
 FPBUF 150
 Full-Function transactions
 scheduling 5

G

GSAM 53, 55, 123–124
 PROCOPT 123
 GTF
 SMF 193

H

HALDB 29, 53, 61, 64–65, 67, 69–70, 72–73, 82, 87–89, 104–105, 124, 158
 HDAM 53, 55, 57, 61–63, 67, 69, 71–74, 76, 78, 81–84, 86–88, 90, 104–105, 124, 225–226, 228, 230, 233
 randomizers 62
 HIDAM 53, 56–57, 60–63, 66, 69, 71, 78, 80–84, 86–88, 104–105, 114, 121, 199, 227–228, 230
 High Performance Pointer Checker 74
 HIOP 4–6, 10, 14, 126, 133, 139
 HIOP pool 4
 Hiperspace 116, 118–119, 121
 HISAM 53, 55, 57–60, 66, 69, 77–78, 84–85, 87–88, 117, 225
 host name 216
 HSSP 57, 92, 103
 HTTP 215

I

IDCAMS 164
 IMS xiii–xiv, 1–17, 19–22, 24–30, 32, 34, 36, 41–43, 45–51, 54–56, 58, 60–62, 64, 66–84, 86–92, 94–101, 103–104, 106–107, 109, 111–112, 115–122, 124–153, 155–158, 160, 162, 164–166, 169–171, 173, 175–177, 181, 183–199, 201–209, 211–219, 222, 224–230, 238, 241
 IMS command 162
 IMS Connect 4, 19, 21, 26, 42, 211–216, 220
 MAXSOC 214
 PORTID 212
 IMS Connect Extensions 21, 42–43
 IMS Connector for Java 213–214
 IMS execution parameters 130
 IMS Java 184, 211, 217, 219
 IMS latches 144

- IMS Monitor 11–13, 17, 19–21, 139, 188, 192, 195
- IMS Monitor reports 12, 17
- IMS PA 21, 26, 92, 96, 99, 102, 104, 140, 146–147, 192–193
 - ESAF (Log) report 193
 - ESAF (Monitor) report 193
- IMS Performance Analyzer 42
 - see IMS PA 19–20, 26, 96, 102–103, 140, 194
- IMS pools 36
- IMS Problem Investigator 42
- IMS SOAP Gateway 215–216
- IMSID 185
- IMSMON 17
- IMSpIex 36, 158
- INIT.DB command 65, 104
- INIT.DBDS 94
- INIT.PART command 67, 70, 72
- intent conflict 15, 126
- intent lists 15, 126, 138, 147
- IOPCB 6
- IPV6 214–215
- IRLM 29–31, 34–36, 49, 100, 141–142, 189–190, 201–202, 204–205
 - address space 142
 - false contention 204
 - lock structure 203–204
 - locks 204
 - PC=NO 203
 - PC=YES 203
- IRM header 213–214
- ISC 4–6, 126
- ISPF 20, 26, 28–29, 193
- IWAIT
 - I/O 188, 195
 - NOT-IWAIT 5, 10, 109, 143, 233
 - region 195
- IWAIT times 139
- IWAITs 23–24, 92, 104, 145, 188, 196

J

- J2EE 215
- Java 170, 184, 211, 213–214, 217–221
- Java Virtual Machine 218
- JBP 183, 185, 217
- JCL 17, 20, 28, 111, 113, 130, 158, 160, 178, 228
- JDBC 217
- JMP 183, 185, 217, 220
- JVM 218, 220

K

- KBLA 20, 27–29, 31–32, 34
- key-sequenced data set (KSDS) 162
- KSDS 25, 58–59, 67, 69, 87, 115–117, 119, 123, 159, 162, 224–225, 227, 229–230

L

- LGMSGSZ 207
- LIST.RECON 162

- LKASID 97–99, 206
- LLA 127
- load balancing 214
- LOCKSIZE parameter 189
- log records 1, 3, 8, 13, 27–30, 34, 80, 131, 136, 140, 156, 209
- logical relationships 53, 57, 60, 81, 224
- LPA 129, 138
- LU2 215

M

- MAXFILEPROC 214
- MAXPST 152
- MAXRGN 5, 129–130, 209
- MAXSOC 213, 215
- MAXSOCKETS 214
- MAXTHRD 151
- MAXTHREAD 151
- MAXUSRS 203
- MCS 132, 162
- message queue
 - I/Os 138
 - QBUF 10, 138
- message queue data sets 5, 14, 126
- MFS 4–6, 14, 126
- migration xiv
- MINTHRD 150
- MINTHREAD 150
- mode 7, 50, 129, 136, 143
- MODE=MULT 16
- monitoring tools 12, 19–20
 - IMS PA 20, 26
- MPP 102, 112, 129–130, 132, 144, 183, 185, 188, 198–199, 220
- MQSeries 184
- MSC 11–13, 29–30, 136, 207
- MSDB 7, 57, 90, 93–95, 102–103
 - IMS system checkpoint 94
- MVS 129, 131, 135–138, 140–143, 202, 207–209
- MVS fix list 141, 145

N

- NBA 33–34, 98, 101, 103, 131, 150, 178
- number of message-processing regions 144

O

- OBA 92, 98, 101, 103, 131–132, 150, 178
- OBJAVGSZ 207–208
- OLDS 4, 28, 30, 32, 34, 104, 130–131, 136, 151, 156–158, 164–165, 216
- on demand 52, 211
- online change 146
- OSAM 31, 43, 51, 53, 55, 60, 63, 66–67, 69, 73, 83, 88, 104–109, 111, 114–115, 118, 124, 139, 141, 145, 177, 191, 201, 205, 225, 227–228
 - sequential buffering 51, 112, 115, 229
 - statistics 110
- OSAM buffer pool 106–108

OTMA 4, 207, 212
OTMASE 166

P

page faulting 145
page faults 107
page fixing 131
paging 13, 36, 50–51, 95, 107, 128, 140–141, 143, 145
 swapping 141
Parallel xiv, 36, 153, 201
parallel scheduling 151
PBOF 150
PCB 89, 92–94, 97, 112, 114, 171, 173, 175–176, 188, 190, 196–197, 199, 219, 232
performance xiii–xiv, 1–4, 6, 8–13, 17–21, 24–26, 29–30, 32, 36, 42–43, 45–46, 48–51, 53–54, 58, 62–63, 74–75, 86, 90–96, 98, 100–101, 107, 111, 118, 123–130, 134–137, 139–141, 150–152, 156–157, 159, 161–162, 169, 175–177, 180, 183, 189–193, 200–204, 206–209, 211–216, 218–220, 222, 224–227, 230–232
 database design 21
 DB2 183, 190
 DBCTL 150
 DBRC 159
 logging 157
 objectives 2, 46, 212, 224
persistent socket 213
PI 21, 31, 100, 137, 144, 189, 199
PIMAX 137, 152
PL/I 128, 179
PLPA 128
pool space failure 147
port 212–213, 216
preload 128
problem
 performance 11, 227
PROCLIB 94–95, 97, 101, 107, 112, 116, 128, 131, 133, 135–137, 139, 141, 144, 180, 185
PROCLIM 5, 129, 188
PROCOPT 92–93, 97, 100, 104, 124, 175–177, 190, 206, 232
 GOT 176–177, 232
 GOx 176, 190
Program isolation 140
program load 5, 7, 10, 125, 144, 188
project xiv
PSB pool 10, 141, 144–145
Pseudo-WFI (PWFI) 188
PST 13, 36, 102, 113, 137, 146, 150–152, 199, 238, 241
PWFI 130, 151, 188

Q

QBUF 4, 135, 138, 142, 207
QBUFHITH 207
QBUFMAX 207
QBUFSZ 207
QSAM 54–55, 106, 124
Quick reschedule 188

R

RACF 31, 155, 165–166, 191, 215–216
real storage 20, 27, 43, 110–111, 128, 140, 148, 157, 229
RECANY buffers 138
RECON 162, 164
RECON data sets 159–162
Redbooks Web site 251
 Contact us xvi
region occupancy 15, 118, 126, 188
RES 152
RESLIB 112
Resource 36, 50, 92, 101, 104, 115, 132, 142, 146–147, 155, 161, 165, 192
resource 42
response time
 I/O 14, 51, 126
RMF 49, 105, 139–141, 165, 192, 202, 204
RMF Monitor I 140
RMF Monitor I reports 140
RMF Monitor II 140
RMF Monitor II reports 140
ROLL call 141
RRS 207
RSR 158

S

scheduling 4–7, 10, 13, 15–16, 27, 33, 123–126, 128, 137–138, 140, 143–144, 147–148, 151, 187–189, 209
SDFSRESL 163
security 28, 42, 87, 165–166, 187, 191–192, 218
 DB2 subsystem 191
 SAF interface 165
 SMU 165
SECURITY macro 165
Shared Queues 201, 206
shared queues 138, 153, 206–207, 209
shared SDEPs 100
Shared Virtual Storage Option 91, 96
Shared VSO 96, 201, 206
SHMSGSZ 207
SINGLE 108–111
SMB 4–5, 126, 198
SMF 3, 139, 165, 203
SMU 165–166
SOAP 211, 215–216
sockets 21, 213–214
space failure 147
SPM 152
SQL 3, 184–185, 187–189, 191–193, 195, 217, 219
SSENVAR 212
SSLPORT 212
state data 140
Statistical Analysis Utility 27
storage 4–5, 20, 27, 31, 36, 43, 50–51, 56–57, 85, 90, 92, 94–96, 99, 101, 107, 110–111, 115–116, 118, 121, 123, 128–130, 133–134, 136, 138, 140–141, 144–145, 147, 151–152, 157, 161, 165, 179–181, 187–188, 202–203, 205–207, 209, 229

syncpoint 9, 93–94, 98, 206
SYS1.PARMLIB 141, 214
sysplex 1, 153, 161, 201, 204–206, 209, 212, 220
Sysplex Distributor 214
system definition 136, 138, 146, 191
system generation 130, 165

T

TCP/IP 21, 49, 212–215
TCP/IP statement 212
TIMEOUT 203
TRANSACT 129–130, 188
transaction
 profile 2–3, 47, 181, 220
transaction code 130
transaction profiles 48–49
TSO 162, 183
tuning xiii–xiv, 11, 20–21, 26, 30, 42, 101, 106, 109–110, 115, 121–123, 134, 224–225, 228–230, 233
two-phase commit 6, 184

U

U777 141
UOR 17
UPDATE 192
user data 89–90, 202
user exits 42

V

VAUT 139
VIPA 214
Virtual fetch 129
VLF 166
VSAM 22, 24–25, 31, 43, 51, 53–55, 57–59, 66, 68–70, 73, 83, 88, 95, 104–105, 112, 115, 117, 119–124, 137, 139–141, 145, 158–160, 162, 164, 191, 201, 205, 224–225, 227, 229–230, 233
 BGWRT 118
 statistics 24, 120–121
VSAM buffer pool 24, 106, 119
VSO 33–34, 57, 90–96, 99–100, 102, 132, 158, 201–202, 205–206
 I/O reduction 93
 IMS system checkpoint 93
 OThread process 94
 performance tips 91
VSPEC 152
VTAM 4, 49, 126, 135, 142, 215
VTCBs 135

W

WebSphere 217
WFI 7, 27, 130, 180, 188–190, 209
 transactions 189
WLM 4, 6–7, 15–16, 45–52, 126, 140, 142–143, 203
workload balancing 36, 206
Workload Manager 12, 45–46

X

XCF 140, 204, 212, 214
XCF group 214
XML 216–217
XRF 158

Z

z/OS 2, 12, 15, 19–21, 26, 36, 41–43, 45–46, 50–52, 67, 69, 74–75, 84, 90–92, 95, 99, 104, 115–116, 118, 126, 128, 161–162, 170, 181, 183, 192, 202, 212, 218, 220
z/OS 1.7 220

Archived



IMS Performance and Tuning Guide

(0.5" spine)
0.475" <-> 0.873"
250 <-> 459 pages



IMS Performance and Tuning Guide



Redbooks

Learn about IMS database, transaction manager, and system performance

Look at the available methods and tools for monitoring

Examine various aspects of Parallel Sysplex

This IBM Redbook provides IMS performance monitoring and tuning information. This book differs from previous IMS performance and tuning redbooks in that there is less emphasis on the internal workings of IMS and more information about why and how certain options can affect the performance of IMS.

Most of the information in the previous IBM Redbook IMS Version 7 Performance Monitoring and Tuning Update, SG24-6404, is still valid, and in most cases, continues to be valid in any future versions of IMS. This book is not an update or rewrite but instead attempts to be more of a guide than a reference. As such, the team gathered experiences and data from actual production environments as well as from IBM benchmarks and solicited input from experts in as many areas as possible.

You should be able to find valuable new information and perhaps validate things you might have questioned. Hardware and software characteristics are constantly changing, but hopefully the information that you find here provides a basis to help you react to change and to keep your IMS running efficiently.

In this IBM Redbook, we introduce methods and tools for monitoring and tuning IMS systems, and in addition to IMS TM and DB system-wide performance considerations, we dedicate separate chapters for application considerations, IMS and DB2 interoperability, the Parallel Sysplex environment, and On Demand considerations.

**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**

SG24-7324-00

ISBN 0738494615