

Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS

Understand the DB2 traces

Monitor and tune the DB2 subsystem

Monitor and tune the DB2 transactions



Paolo Bruni
Felipe Bortoletto
Adrian Burke
Cathy Drummond
Yasuhiro Ohmori



International Technical Support Organization

**Subsystem and Transaction Monitoring and Tuning
with DB2 11 for z/OS**

February 2014

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

First Edition (February 2014)

This edition applies to Version 11, Release 1 of DB2 for z/OS (program number 5615-DB2) and Version 11, Release 1 of DB2 Utilities Suite for z/OS, V11.1 (program number 5655-W87).

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this IBM Redbooks publication for more current information.

Contents

Examples	ix
Figures	xiii
Tables	xvii
Notices	xix
Trademarks	xx
Preface	xxi
Authors	xxi
Now you can become a published author, too!	xxii
Comments welcome	xxiii
Stay connected to IBM Redbooks	xxiii
Part 1. Introduction to DB2 monitoring.	1
Chapter 1. Introduction to monitoring and tuning	3
1.1 Overview of tuning: The four steps	5
1.1.1 Identifying	5
1.1.2 Diagnosing	5
1.1.3 Solving	6
1.1.4 Preventing	6
1.2 DB2 Performance Solution Pack	6
Chapter 2. DB2 traces	13
2.1 Trace types	14
2.2 Trace destinations	17
2.3 Managing DB2 traces	18
2.3.1 Start trace	18
2.3.2 Display trace	19
2.3.3 Modify trace	20
2.3.4 Stop trace	20
2.3.5 Loading IFCID field descriptions into a table	20
2.3.6 Reporting performance data	23
2.4 Compressing your trace records	23
2.4.1 Software compression	23
2.4.2 Hardware compression	24
Chapter 3. System z related information.	29
3.1 Workload Manager	30
3.2 The DB2 subsystem	32
3.2.1 z/OS factors helping DB2	33
3.2.2 DB2 subsystem consequences	34
3.3 Thread level	35
3.3.1 In DB2	35
3.3.2 Regarding WLM resource group capping	37
3.3.3 Transaction level goal adjustment	38
3.4 General WLM best practices	41
3.5 Looking at RMF data	44

3.5.1 Overview records	49
Part 2. Subsystem monitoring	51
Chapter 4. System address space CPU time	53
4.1 What contributes to address space CPU time	54
4.2 How to look at the numbers	57
Chapter 5. EDM pools	61
5.1 EDM pools overview	62
5.2 Plan and package storage and skeleton pool	65
5.3 DBD pool	67
5.4 DB2 statement caching	67
5.4.1 Behavior of executing dynamic SQL without statement caching	68
5.4.2 Global dynamic statement caching	69
5.4.3 Use of local dynamic statement cache	70
5.4.4 Dynamic SQL literal replacement	71
5.4.5 EXPLAIN MODE special register	73
5.4.6 Capturing data from statement cache	74
5.4.7 Monitoring statement cache	75
Chapter 6. Data set open and close	77
6.1 Open and close data sets	78
6.1.1 Pseudo close mechanism	78
6.1.2 CLOSE specification for the table spaces	78
6.2 Controlling number of open data sets	79
6.2.1 Monitoring open and close activity	80
6.2.2 Managing and collecting additional information for open and close data sets ..	81
Chapter 7. Log activity	83
7.1 Log activity	84
7.2 Log write activity	84
7.3 Log read activity	86
7.4 Log I/O tuning possibilities	88
Chapter 8. IRLM, locking, and latching	89
8.1 DB2 and the internal resource lock manager (IRLM)	90
8.1.1 IRLM startup procedure options	90
8.1.2 z/OS commands on IRLM	91
8.1.3 Accounting and tracing IRLM	93
8.2 DSNZPARMs related to locking	93
8.3 Lock avoidance	94
8.3.1 IRLM latch contention	97
8.4 Data sharing locking	98
8.4.1 Global contentions	99
8.4.2 P-lock contention negotiations	101
8.4.3 Data sharing lock tuning	103
8.5 Internal DB2 latch contention	104
Chapter 9. The zIIP engine and DB2	109
9.1 What a zIIP is	110
9.1.1 What drives customers to purchase zIIPs	110
9.2 What work is zIIP eligible	110
9.2.1 zIIP eligibility from DB2 V8 until now	111
9.2.2 System agents	112

9.2.3 Native stored procedures	113
9.2.4 zAAP on zIIP	113
9.3 How many zIIPs are enough	114
9.3.1 Enough zIIP for the business	115
9.3.2 Enough zIIP for the workload	117
9.4 Avoiding self-imposed bottlenecks	119
9.4.1 IEAOPTxx parameters	119
9.4.2 zIIP capacity in DB2 10 and DB2 11	121
9.4.3 Conservative zIIP utilization	121
9.5 Getting more zIIP eligibility	122
Chapter 10. Buffer pools and group buffer pools	125
10.1 Local buffer pools	126
10.1.1 Basic placement and settings	126
10.1.2 Thresholds	130
10.1.3 Tuning	131
10.1.4 DB2 11	142
10.2 Group buffer pools	143
10.2.1 Best practices	144
10.2.2 What to look out for	147
10.2.3 Rough sizings	148
10.2.4 Group buffer pools and DB2 11	151
Chapter 11. DDF activity	153
11.1 DDF and distributed environments	154
11.1.1 Database access thread	155
11.1.2 Modes of DBATs and inactive connections	156
11.1.3 High Performance DBATs	156
11.1.4 Sysplex workload balancing and connection queue redirect	159
11.2 Monitoring DDF activity	161
11.2.1 DDF activity report	161
11.2.2 Statistics Global DDF Activity	162
11.2.3 Statistics DRDA remote locations	164
11.2.4 Monitoring using system profiling	165
11.2.5 Client information and workload management	165
Chapter 12. Workfiles, RID, and sort pools	167
12.1 The workfile evolution	168
12.1.1 Tuning the workfiles	170
12.2 Sorting	171
12.2.1 Buffer pool settings	174
12.2.2 DB2 11	176
12.3 RID list processing	177
Chapter 13. Virtual and real storage	179
13.1 Storage concerns	180
13.1.1 Real storage concerns	180
13.1.2 Virtual storage concerns	180
13.2 Monitoring real storage usage	181
13.2.1 Maintaining control over the system use of real storage	184
13.2.2 64-bit storage	187
13.2.3 LFAREA	188
13.2.4 New in DB2 11	189
13.3 Monitoring virtual storage usage	191

Part 3. Transaction monitoring	199
Chapter 14. Accounting trace overview	201
14.1 Response time factors	202
14.2 Response time scope	203
14.3 DB2 accounting data	204
14.3.1 When DB2 accounting data is written	204
14.3.2 Accounting class 1 data	210
14.3.3 Accounting class 2 data	212
14.3.4 Accounting class 3 data	214
14.3.5 Accounting class 1,2,3	216
Chapter 15. Analyzing accounting data: CPU and suspension time	217
15.1 Top-down analysis	218
15.1.1 Analyzing thread activity time	221
15.2 Time outside of DB2 versus time in DB2	221
15.2.1 Time spent outside of DB2 is bigger than time in-DB2	222
15.2.2 Time spent inside of DB2 is bigger than time outside of DB2	222
15.3 In-DB2 CPU time versus elapsed time	224
15.3.1 CPU time	224
15.3.2 Suspension time	226
15.3.3 What is left: NOT ACCOUNT time	228
Chapter 16. I/O suspensions	231
16.1 Synchronous I/O suspensions: Large number	232
16.1.1 Synchronous database I/O suspensions	234
16.1.2 Tuning the number of synchronous I/O suspensions	235
16.1.3 Tuning I/O wait time	235
16.2 Log read suspensions	236
16.2.1 Archive Log Mode (Quiesce) suspensions	237
16.3 Synchronous log write I/O suspensions	238
16.4 Suspensions for reads by other agents	240
16.5 Suspensions for writes by other agents	241
Chapter 17. Locking, latching, and buffer pool accounting counters	243
17.1 Lock and latch	244
17.2 Locks	245
17.2.1 Locking information in accounting at plan level	246
17.2.2 Data sharing locking	247
17.2.3 Global lock suspensions	248
17.3 Latches	250
17.3.1 Latch suspensions	252
17.3.2 Page latch suspensions	252
17.4 Buffer pool information in accounting at plan level	254
17.5 Group buffer pool information	254
Chapter 18. Service task suspension	257
18.1 Synchronous execution unit switch suspensions	258
18.2 Large suspensions for synchronous EU switches	260
18.2.1 Service task wait: Commit related timers	260
18.2.2 Data set open/close	261
18.2.3 SYSLGRNG REC	261
18.2.4 Data set Extend/Define/Delete	262
18.2.5 OTHER SERVICE tasks	262

18.3 How to find which service task is invoked	263
Chapter 19. Stored procedures, user defined functions, and triggers	265
19.1 Stored procedures	266
19.1.1 Nested activity for WITH RETURN processing	270
19.2 User defined functions	272
19.3 Triggers	272
19.4 Nested activity accounting	273
19.5 IFCIDs for nested activity	274
Chapter 20. DRDA, parallelism, and statement cache	277
20.1 Accounting for DDF work	278
20.2 Setting extensions to client information	280
20.3 Accounting for parallel tasks	282
20.4 Capturing performance data with the statement cache	284
20.5 Performance traces for SQL tuning	287
Part 4. Appendixes	289
Appendix A. Production modeling	291
A.1 Functional requirements	292
A.2 Simulating production	293
A.3 Data sharing members on dissimilar hardware	294
A.4 Generating input data for spreadsheets	295
Appendix B. IBM OMEGAMON XE for DB2 performance database	297
B.1 Introduction	298
B.1.1 Performance database structure	299
B.2 Creating the performance database	302
B.2.1 Creating a DB2 z/OS database	302
B.2.2 Customizing the PDB create table DDL	302
B.2.3 Creating the PDB accounting and statistics tables	305
B.3 Extracting, transforming, and loading accounting and statistics data	306
B.3.1 Extracting and transforming DB2 trace data into the FILE format	306
B.3.2 Extracting and transforming DB2 trace data into the SAVE format	307
B.3.3 Preparing a load job	308
B.3.4 Loading accounting and statistics tables	309
B.3.5 Maintaining PDB tables	309
B.4 Sample query for application profiling	310
B.5 Using the UDF for application profiling	312
B.6 Additional information	313
B.7 MEMU2 versus Performance Database fields	313
Related publications	319
IBM Redbooks publications	319
Other publications	319
Online resources	320
Help from IBM	320
Index	321

Examples

2-1	START TRACE for accounting	19
2-2	DISPLAY TRACE	19
2-3	MODIFY TRACE	20
2-4	STOP TRACE	20
2-5	Create IFCID table	20
2-6	LOAD IFCID table job	21
2-7	LOAD IFCID table job output	22
2-8	Definition in SMFRMxx	24
3-1	QMF service class	31
3-2	Service units shown in WLM activity report	31
3-3	Blocked workload from RMF CPU report	34
3-4	-DIS THREAD(*) SERVICE (WAIT)	36
3-5	-DIS THREAD(*) TYPE(SYSTEM) cont.	37
4-1	CPU times from a DB2 9 customer	55
4-2	CPU times from a DB2 10 customer	56
4-3	Divide the address space times by the amount of work coming through the system	56
4-4	Where to find commits and rollbacks	56
4-5	IFCID 369 aggregated statistics and accounting	57
4-6	SPROC performance benefit not being utilized	58
5-1	EDM pool block from the statistics report	64
5-2	Plan and package storage below 2 GB	65
5-3	Monitoring EDM skeleton pool	66
5-4	Monitoring DBD pool	67
5-5	Sample dynamic statement cache listing for dynamic SQL literal replacement	72
5-6	Statistics trace output sample with literal replacement	73
5-7	Collecting explain information while executing dynamic SQL using SPUFI	73
5-8	Sample output of statement cache using EXPLAIN MODE	74
5-9	Dynamic SQL statement block from statistics report	75
6-1	OPEN/CLOSE activity	81
6-2	Monitoring number of open data sets at buffer pool level	81
6-3	-STA TRACE(P) CLASS(19)	82
6-4	IFCID 370 and 371 report	82
7-1	Log write statistics	86
7-2	Log activity in statistics report	87
8-1	Modify irlmproc,STATUS,STOR	92
8-2	Dynamically changing deadlock or timeout frequency	92
8-3	IRLM storage accounting block from statistics report	93
8-4	Locking activity	96
8-5	DSNR035I detecting long running UR, URCHKTH	96
8-6	DSNJ031I detecting large updates with out commit, URLGWTH	97
8-7	Locking activity block	98
8-8	Data sharing locking block	100
8-9	Data sharing locking	102
8-10	Group buffer pool statistics	103
8-11	Latch counters	106
9-1	DB2 9 SRB time	112
9-2	DB2 10 SRB time	112
9-3	DB2 11 SRB time	112

9-4 DB2 10 customer without any zIIP engines	113
10-1 Sample query to determine placement of objects in buffer pools	126
10-2 OMPE buffer pool statistics.	132
10-3 Calculating size of index buffer pool	134
10-4 DB2 11 sequential page counters	136
10-5 MVS DISPLAY VIRTSTOR,LFAREA	141
10-6 -DIS BUFFERPOOL (*) SERVICE=4	141
10-7 -D XCF,STR,STRNM=groupname*	144
10-8 IXC558I	145
10-9 GBP statistics for synchronous read	146
10-10 -DIS GBPOOL(*) GDETAIL(*) TYPE(GCONN).	147
10-11 Failed writes from statistics report	148
10-12 MSTR JOBLOG message for short on storage	148
10-13 -DIS GROUPBUFFERPOOL(*) GDETAIL(INTERVAL)	149
10-14 Statistics report to get changed pages per second	149
11-1 Enable High Performance DBAT	157
11-2 Disable High Performance DBAT	157
11-3 High performance DBAT activity	158
11-4 DSNL030I	159
11-5 DSNL074I	160
11-6 DSNL075I	160
11-7 -DISPLAY DDF DET displaying connection queue	160
11-8 -DISPLAY DDF DETAIL	162
11-9 Global DDF activity	163
11-10 DRDA remote locations	164
12-1 To see if DB2 used a non-favored table space	168
12-2 Typical 4 KB versus 32 KB distribution	171
12-3 Workfile BP sorting	173
12-4 IFCID 95-96 sorting	174
12-5 Workfile BP stats	175
12-6 OMEGAMON PE V5R2 and DB2 11 workfile stats	176
12-7 RID list processing statistics	178
13-1 List of IFCID 225 fields	187
13-2 Real storage in use	187
13-3 IRLM storage use	189
13-4 xPROC storage	190
13-5 Extended low private storage	191
13-6 DSNVMON message	193
13-7 -DIS THREAD(*) TYPE (SYSTEM).	194
14-1 How OMEGAMON PE shows DB2 class 1 timers	211
14-2 How OMEGAMON PE shows DB2 class 2 timers	213
14-3 How OMEGAMON PE shows DB2 class 3 timers	214
15-1 OMEGAMON PE Accounting Report -JCL	218
15-2 OMEGAMON PE accounting report for call attach connection type	218
15-3 OMEGAMON PE Statistics Report -JCL	220
15-4 OMEGAMON PE Statistics Report	220
15-5 When class 2 SE CPU time is longer than class 2 elapsed time	226
15-6 NOT ACCOUNT time	229
16-1 DASD over utilization causes slow I/O	233
16-2 High suspension for others agents reads	240
16-3 High suspension for others agents write	241
17-1 Locking info in accounting at plan level	246
17-2 Data Sharing Locking Section	247

17-3	Global contention L-locks and P-locks	249
17-4	DB2 latches	250
17-5	DB2 latch counters	251
17-6	Lock Section	251
17-7	PAGE LATCH suspension	252
17-8	Buffer pool information in accounting at plan level	254
17-9	Global buffer pool information	256
18-1	Execution unit switch class 3 timers	258
19-1	Stored procedure accounting - caller and procedure	266
19-2	Autonomous stored procedure accounting - caller	266
19-3	Autonomous stored procedure accounting - stored procedure	267
19-4	Accounting for a stored procedure with a long wait for the WLM application environment	268
19-5	Accounting data from a looping stored procedure (no SQL in the loop)	268
19-6	Application data from a looping stored procedure with SQL in the loop	269
19-7	User-defined function accounting	272
19-8	Trigger Accounting	272
20-1	Sample coding for setting client information using JDBC 4.0	280
20-2	-DISPLAY THREAD(*)	281
20-3	Accounting identification block	281
20-4	Accounting - Truncated Values	281
20-5	Accounting - Initial DB2 Common Server or Universal JDBC Driver Correlation	282
20-6	Start trace command for IFCID 318	285
20-7	Select statement against the DSN_STATEMENT_CACHE_TABLE	285
20-8	DSNTIAUL control statements to load statement cache to delimited flat file	285
A-1	Selecting the values from the PLAN_TABLE	293
A-2	Inserting a PROFILEID	293
A-3	Examples of individual overrides	294
A-4	EXPLAIN a query and then select the CPU speed from each member	294
A-5	EXPLAIN another statement and compare the before and after CPU_SPEEDs	295
B-1	PDB create DB2 z/OS database	302
B-2	PDB generate create table DDL data set	303
B-3	Create table space template	305
B-4	Batch JCL PDB accounting and statistics table creation	305
B-5	OMEGAMON PE extract and transform DB2 trace data into FILE format	306
B-6	Extract and transform accounting SAVE format	307
B-7	Merge statistics and accounting file load utility control statements	308
B-8	Merge accounting save load utility control statements	308
B-9	Image copy batch JCL	309
B-10	Reorg batch JCL	309
B-11	OMEGAMON PE SQL table UDF	310
B-12	Starting UDF for JBC driver Type 4	312

Figures

1-1 DB2 Performance Solution Pack	7
2-1 Accounting trace classes	14
2-2 Statistics Trace Classes	15
2-3 Performance trace classes	15
2-4 Audit trace classes	16
2-5 Monitor trace classes	16
2-6 SMF record types	17
2-7 Start trace syntax diagram	18
2-8 PCIe function and zEDC activity report.	25
3-1 DDF higher than DB2	35
3-2 WLM information in -DIS THREAD	36
3-3 WLM resource group capping.	38
3-4 WLM activity report showing response times	39
3-5 Poor performance index for CICS transaction	40
3-6 CICS response time variance	41
3-7 WLM service policy overview from RMF Spreadsheet reporter	42
3-8 WLM service policy filtered by service class importance	43
3-9 Average response time shown for service class that runs under a velocity goal	43
3-10 Download page for the RMF spreadsheet reporter.	45
3-11 RMF components, focusing on left side of the chart for the Spreadsheet reporter. . .	46
3-12 RMF Spreadsheet Reporter main panel	48
3-13 WLM overview report	49
3-14 DASD Activity Report	50
3-15 DASD response time overview by device.	50
4-1 CPU times by address space	54
4-2 Address space CPU times	57
5-1 EDM pools from DB2 V7 to DB2 V9	62
5-2 EDM pools in DB2 10 and later.	63
5-3 Dynamic SQL without statement caching	68
5-4 Global dynamic statement caching	69
5-5 Local dynamic statement caching.	70
8-1 DB2 data sharing global locking system components	99
9-1 zIIP eligible work.	111
9-2 zIIP and stored procedures.	113
9-3 Peak CPU aligned.	116
9-4 Peak zIIP aligned	116
9-5 Missed zIIP potential.	117
9-6 LPAR trend of zIIP usage	117
9-7 Response time based on CPU utilization	118
9-8 zIIP constrained parallelism	120
10-1 Buffer pool critical thresholds	131
10-2 IFCID 199	135
10-3 PGFIX(YES) candidates	138
10-4 Total pages read and written	139
10-5 DBM1 SRB time in seconds per minute	139
10-6 Buffer pool use of large frames.	143
10-7 Example of rough sizing of GBP.	150
11-1 DB2 and distributed applications	154

11-2 Processing within DDF	155
12-1 DB2 9 workfile and temp picture	168
12-2 DB2 9 NFM and up view	169
12-3 Sort phases	172
13-1 Aux storage	182
13-2 Customer real storage, axis in MB	186
13-3 64-bit virtual	188
13-4 Virtual storage visual breakdown	191
13-5 DBM1 virtual storage	193
13-6 Storage contraction	195
13-7 Thread footprint calculation	196
13-8 Storage creep in DBM1 address space	197
14-1 Response time factors	202
14-2 Response Time Scope	203
14-3 The DB2 view	204
14-4 Thread Deallocate - Normal case	205
14-5 Thread Deallocate - IMS or CICS	205
14-6 Accounting Pitfall	206
14-7 RRS using accounting interval commit it	207
14-8 Accounting record creation when using CMTSTAT=INACTIVE	208
14-9 Accounting record creation when using ACCUMACC >1	209
14-10 DB2 Accounting Trace Class 1	210
14-11 DB2 Accounting Trace Class 2	212
14-12 DB2 Accounting Trace Class 3	214
14-13 Recording class1/class2 elapsed and CPU times and class 3 suspension time ..	216
15-1 Analyzing thread activity time	221
15-2 Time spent outside of DB2 is bigger than time in-DB2	222
15-3 Time spent inside of DB2 is bigger than time outside of DB2	223
15-4 Is most of the time spent in burning CPU or waiting	223
15-5 zIIP off load	225
15-6 Class 3 suspension types	227
15-7 What is left	228
15-8 Effect of lack of CPU resources on CL3/8 timers	229
16-1 Synchronous I/O suspensions large	232
16-2 Synchronous database I/O suspensions	234
16-3 Log read suspensions	236
16-4 Archive Log Mode (Quiesce) suspensions	237
16-5 Synchronous log write I/O suspensions	238
16-6 Suspensions for reads by other agents	240
16-7 Suspensions for writes by other agents	241
17-1 Lock/latch suspensions	245
17-2 Global lock suspensions	248
17-3 "Mixing" Global/Local Lock Suspensions	250
17-4 Page latch suspensions	253
18-1 Synchronous EU Switch Suspensions	258
18-2 Large Suspensions for Synchronous EU Switches	260
18-3 Service Task Wait - Commit Related Timers	261
18-4 OMEGAMON PE report of some service task RMIDs	263
19-1 Nested activity for WITH RETURN	270
19-2 DB2 Accounting Class 1-2 Plan Level	271
19-3 Package level accounting for WITH RETURN cursor in a stored procedure	271
19-4 Accounting class 1,2,3 times with nested activities	273
19-5 OMEGAMON Accounting Report	274

19-6	Stored procedure monitoring	275
20-1	DDF accounting at plan level	278
20-2	Accounting at plan level report	279
20-3	OMEGAMON PE Accounting Report/Trace for a DB2 10 parallel query	283
20-4	OMEGAMON PE Accounting Report/Trace for a V11 parallel Query.....	284
A-1	APAR for production modeling	292
B-1	OMEGAMON PDB ETL overview.....	298
B-2	PDB structure accounting tables	299
B-3	PDB structure statistics tables	301
B-4	Customize a create table DDL	304

Tables

2-1 General PCIE Activity	25
2-2 Hardware Accelerator Activity	27
2-3 Hardware Accelerator Compression Activity	27
3-1 Importance levels and goals	31
3-2 Max velocity goals	33
5-1 Summary of parameter relates to EDM pools	63
5-2 Settings for CACHEDYN_FREELOCAL parameter	71
6-1 Summary of parameters relating to open and close data sets	78
6-2 CLOSE YES versus CLOSE NO	79
7-1 Parameter for logging	84
8-1 Timeout and deadlock detection frequency	91
8-2 z/OS commands for IRLM operations	91
8-3 IRLM and locking related parameters	93
8-4 Latch classes	104
8-5 Latch suspensions	105
10-1 BP setup examples	128
11-1 Summary of DSNZPARM parameters affecting DBATs	155
11-2 Client information field length in DB2 11 for z/OS	166
11-3 DDF work classification attributes for enhanced client information	166
13-1 Total Real in use by DB2 fields to sum	182
13-2 Total Real in use by LPAR fields to sum	183
13-3 Total Real available on the LPAR field	183
13-4 Total AUX in use by DB2	183
13-5 Total AUX on the LPAR	183
18-1 Common DB2 service tasks	264
19-1 External stored procedure diagnostics	267
20-1 Client info property values for type 4 connectivity to DB2	280
B-1 FILE accounting table DDL and load statements	300
B-2 SAVE accounting table DDL and load statements	300
B-3 Statistics table DDL and load statements	302
B-4 IFCIDs in OMEGAMON PE and MEMU2	313

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 grant you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

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

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

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


COPYRIGHT LICENSE:

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

Trademarks

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

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

CICS®	Optim™	RMF™
DB2 Connect™	OS/390®	System z®
DB2®	Parallel Sysplex®	Tivoli®
DRDA®	pureQuery®	Velocity™
FICON®	QMF™	WebSphere®
IBM®	Query Management Facility™	z/OS®
IMS™	RACF®	z10™
InfoSphere®	Redbooks®	z9®
MVS™	Redbooks (logo)  ®	zEnterprise®
OMEGAMON®	Resource Measurement Facility™	zSeries®

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

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

Preface

This IBM® Redbooks® publication discusses in detail the facilities of IBM DB2® for IBM z/OS®, which allow complete monitoring of a DB2 environment. It focuses on the use of the DB2 instrumentation facility component (IFC) to provide monitoring of DB2 data and events, and includes suggestions for related tuning.

We discuss the collection of statistics for the verification of performance of the various components of the DB2 system and accounting for tracking the behavior of the applications.

We have intentionally omitted considerations for query optimization; they are worth a separate document.

Use this book to activate the right traces to help you monitor the performance of your DB2 system and to tune the various aspects of subsystem and application performance.

Authors

This book was produced by a team of specialists from around the world working at the Silicon Valley Laboratory, San Jose, California.

Paolo Bruni is a DB2 Information Management Project Leader at the International Technical Support Organization based in the Silicon Valley Lab. He has authored several IBM Redbooks publications about DB2 for z/OS and related tools, and has conducted workshops and seminars worldwide. During his years with IBM, in development and in the field, Paolo has worked mostly on database systems.

Felipe Bortoletto is a Certified IBM IT Specialist in Information Management and an IBM Certified DBA for DB2 for z/OS V7, V8, V9, and DB2 10. He has 18 years of experience in IT with 13 years of experience with DB2 for z/OS. He joined IBM 9 years ago and is currently a member of the IBM GBS in Brazil. He holds a degree in Computer Science from UNICAMP. Felipe co-authored *Securing and Auditing Data on DB2 for z/OS*, SG24-7720; *DB2 10 for z/OS Performance Topics*, SG24-7942; and *DB2 11 Technical Overview*, SG24-8180.

Adrian Burke is an Open Group Master Certified IT Specialist on the DB2 SWAT team based out of the Silicon Valley Lab, led by John Campbell. In this role as a technical advocate for DB2 for z/OS, Adrian conducts consulting and educational workshops, DB2 health checks, and availability studies for DB2 for z/OS customers. Previously, as a DB2 Advisor, Adrian supported hundreds of DB2 for z/OS customers from a technical sales perspective delivering consultative and educational workshops. Adrian has spoken at numerous DB2 Regional User Groups both in the U.S. and abroad, as well as presenting at IDUG, SHARE, and IOD.

Cathy Drummond is a Software Developer at the IBM Software Group, Information Management, Silicon Valley Laboratory in San Jose, California. She is currently a member of the L2 Support Team for DB2 for z/OS with responsibility in the area of product performance. Cathy joined IBM 20 years ago and has worked in DB2 development and planning.

Yasuhiro Ohmori is an Advisory IT Specialist with IBM Japan Systems Engineering Co., Ltd. (ISE) under GTS in Japan, providing technical support on DB2 for z/OS. He has 12 years of experience in technical support for DB2 for z/OS. Yasuhiro has worked with several major customers in Japan implementing DB2 for z/OS and has conducted workshops for IBM staff in Japan. His areas of expertise include DB2 for z/OS, IBM DRDA implementation, IBM DB2 Connect, and related topics. Yasuhiro co-authored the IBM Redbooks publications DB2 9 for z/OS: Distributed Functions, SG24-6952-01; and DB2 10 Performance Topics, SG24-7942.

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

Jeff Berger
Akiko Hoshikawa
Dan Kozin
Gopal Krishnan
John Tobler
Dan Weis
IBM Silicon Valley Lab

Mark Rader
IBM ATS Dallas

Bart Steegmans
IBM Belgium

John Campbell
Florence Dubois
IBM UK

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- Look for us on LinkedIn:

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

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Part 1

Introduction to DB2 monitoring

In this part of the book, we introduce the concepts of monitoring and tuning, as well as the basic DB2 tracing infrastructure.

This part contains the following chapters:

- ▶ Chapter 1, “Introduction to monitoring and tuning” on page 3
- ▶ Chapter 2, “DB2 traces” on page 13
- ▶ Chapter 3, “System z related information” on page 29



Introduction to monitoring and tuning

There are a few reasons that you might have picked up this book. You may have recently applied maintenance or upgraded to a new release of DB2 or z/OS and you want to compare the before and after cases. You may have experienced general performance degradation and want to find the problem and fix it. Or you may be motivated to find ways to improve performance to save on CPU costs or speed response times to your DB2 subsystem.

How you define good performance for your DB2 subsystem depends on your particular data processing needs and their priority. Performance objectives should be realistic, in line with your budget, understandable, and measurable. Common objectives could include these:

- ▶ Values for acceptable response time (a duration within which some percentage of all applications have completed)
- ▶ Average throughput (the total number of transactions or queries that complete within a given time)
- ▶ System availability, including mean time to failure and the durations of down times

Objectives such as these define the workload for the system and determine the requirements for resources: processor speed, amount of storage, additional software, and so on. Often, though, available resources limit the maximum acceptable workload, which requires revising the objectives.

Presumably, your users have a say in your performance objectives. A mutual agreement on acceptable performance, between the data processing and user groups in an organization, is often formalized and called a service-level agreement. Service-level agreements can include expectations of query response time, the workload throughput per day, hour, or minute, and windows provided for batch jobs (including utilities). These agreements list criteria for determining whether or not the system is performing adequately.

Next, describe a set of preliminary workload profiles that might include a definition of the workload type in terms of its function and its volume. You are likely to have many workloads that perform the same general function (for example, order entry) and have an identifiable workload profile. Other workload types could be more ad hoc in nature, such as SPUFI or IBM QMF™ queries.

Identify the resources required to do the work described for each set of defined workloads, including physical resources managed by the operating system (such as real storage, disk I/O, and terminal I/O) and logical resources managed by the subsystem.

For each workload type, convert the estimates of resource requirements into measurable objectives. Include statements about the throughput rates to be supported (including any peak periods) and the internal response time profiles to be achieved. Make assumptions about I/O rates, paging rates, and workloads.

Next, determine the frequency and level of detail gathered in your collection of performance data. Consider the following cost factors when planning for monitoring and tuning:

- ▶ Trace overhead
- ▶ Trace data reduction and reporting times
- ▶ Time spent on report analysis and tuning action

Inspect your performance data to determine whether performance has been satisfactory, to identify problems, and to evaluate the monitoring process. When establishing requirements and planning to monitor performance, also plan how to review the results of monitoring. Plan to review the performance data systematically. Review daily data weekly and weekly data monthly; review data more often if a report raises questions that require checking. Depending on your system, the weekly review might require about an hour, particularly after you have had some experience with the process and are able to locate quickly any items that require special attention. The monthly review might take half a day at first, less time later on. But when new applications are installed, workload volumes increased, or terminals added, allow more time for review.

The inspection and review process could use data collected by DB2 instrumentation facility traces and summarized and interpreted using the IBM Tivoli® IBM OMEGAMON® XE for DB2 Performance Expert on z/OS (OMEGAMON PE) Performance Reporter or the OMEGAMON PE Performance Warehouse.

Review the data on a gross level, looking for problem areas. Review details only if a problem arises or if you need to verify measurements. When reviewing performance data, try to identify the basic pattern in the workload, and then identify variations of the pattern. After a certain period, discard most of the data you have collected, but keep a representative sample. For example, save the report from the last week of a month for three months; at the end of the year, discard all but the last week of each quarter. Similarly, keep a representative selection of daily and monthly figures.

1.1 Overview of tuning: The four steps

Clearly, application performance is tightly tied to customer satisfaction, revenue generation, organizational productivity, and infrastructure costs. And the ability to have optimized queries in your production applications is a key contributor to application performance. Unfortunately, we are hearing that the performance and scalability of application SQL is getting less and less attention pre-production. Aggressive project delivery schedules, the increasing use of application frameworks that generate SQL, and the disconnection between development organizations, DBA organizations, and test organizations are all contributing factors.

Once in production, application performance is not “set for life”. Applications slow down over time because of increased workload or changes in the database environment. Often applications outgrow the original design parameters. IBM recommends a sequence of four steps to maximize DB2 performance:

- ▶ Identify
- ▶ Diagnose
- ▶ Solve
- ▶ Prevent

1.1.1 Identifying

Performance problems happen without warning. The first step in confronting a performance issue is identification. Where does the problem originate? Is it ongoing or has it concluded; and if so, will it recur?

Identification is not always simple, it can take hours or days without the right tools. Many DBAs have been the first to get blamed for a performance slowdown and then lose time hunting in the database, only to be vindicated by later discovery, when few are paying attention that the problem arose from an application server, network, OS, or CPU issue.

Identifying performance problems calls for a diverse set of capabilities; no single tool provides them all. Fast identification demands automated alerts, proactive notifications on potential problems and 24x7 monitoring. Upon receiving an alert, an operator wants a quick visual scan of what is most likely a complex environment. Unless the problem is obvious, the DBA needs tools that tackle a problem from multiple perspectives.

1.1.2 Diagnosing

A problem identified is not necessarily a problem diagnosed. It's not always obvious which application a particular SQL statement belongs to; today's complex application infrastructures incorporate multiple technology layers that may obscure the links between an application and its SQL executing in the database. The goal is to determine where a transaction is spending too much time and creating performance delays.

To diagnose, the DBA must quickly and accurately determine the specific root causes of poor performance, such as resource shortages or exceptional conditions.

1.1.3 Solving

A performance challenge usually has multiple solutions—and it's not always obvious which one is best. DBAs and teams should not think in terms of one SQL statement or one user's issues. Applying an improvement strategy that worked on one SQL statement to a different type of statement may not produce the same positive outcome.

1.1.4 Preventing

By tuning queries and workloads proactively, organizations can support the business transparently rather than address database performance issues after service has already degraded or an outage has occurred. This is not easy. Traditionally, skilled resources are needed to conduct an adequate review of SQL performance and database physical design before deployment. High skill levels are also required to optimize queries and choose the right preventive measures to avoid degradation but senior DBAs are often busy with more urgent matters, leaving little time to address the large volume of SQL in a workload. Prevention requires a more strategic view than merely responding to visible performance problems.

The ability to analyze historical performance information over time and during heavy-workload periods makes it easier to discern when a proposed performance enhancement has unintended consequences and when it yields a true net benefit. Prevention requires a more strategic view than merely responding to visible performance problems. A historical view of performance using a performance database gives the DBA a benchmark to work against, an understanding of “normal” performance levels and perspective on which areas persistently present problems. The ability to analyze historical performance information over time and during heavy-workload periods makes it easier to discern when a proposed performance enhancement has unintended consequences and when it yields a true net benefit.

1.2 DB2 Performance Solution Pack

Solution packs combine several products into a single consolidated solution providing everything necessary to ensure the execution of a set of data base administration functions. The objectives are to reduce the operational complexity and reduce cost.

IBM DB2 Performance Solution Pack for z/OS, V1.2 (5655-E74) (see Figure 1-1) delivers integrated performance management for DB2 for z/OS by combining function from the following products:

- ▶ IBM Tivoli OMEGAMON XE for IBM DB2 Performance Expert on z/OS, V5.2 3 (5655-W37)
- ▶ DB2 Query Monitor for z/OS, V3.2 (5655-V42)
- ▶ DB2 SQL Performance Analyzer for z/OS, V4.2 (5655-W60)
- ▶ IBM InfoSphere® IBM Optim™ Query Workload Tuner for DB2 for z/OS, V4.1 (5655-AA4)

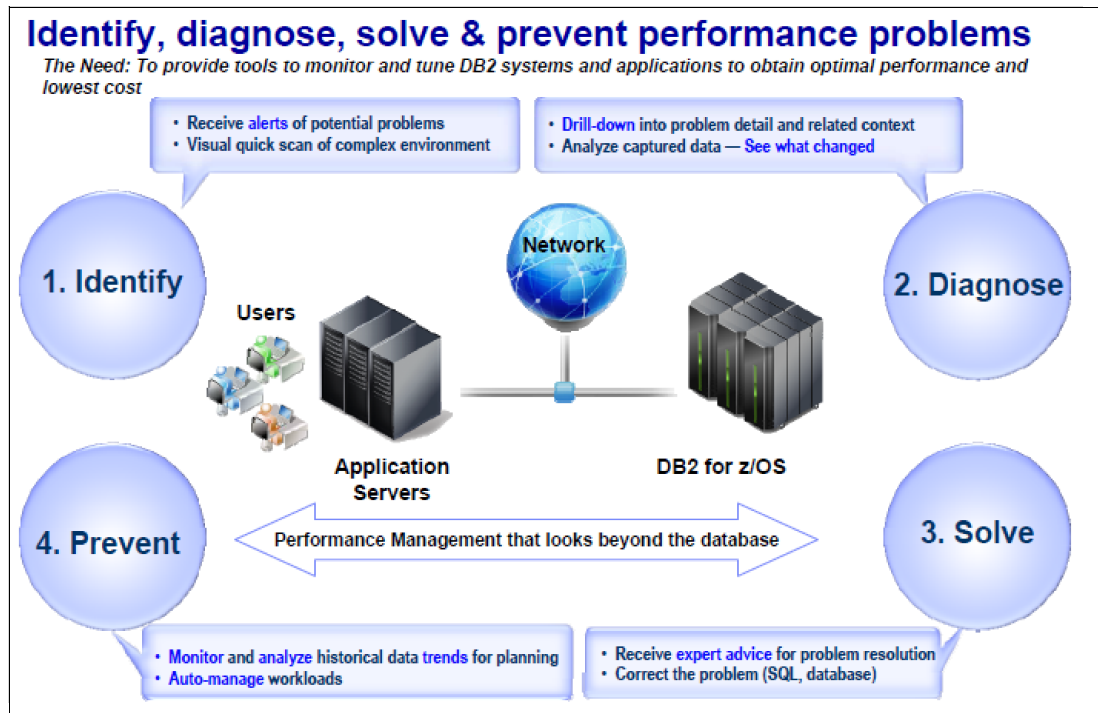


Figure 1-1 DB2 Performance Solution Pack

IBM DB2 Performance Solution Pack for z/OS, V1.2 (5655-E74) offers features, functions, and processes that DBAs can use to more effectively identify, diagnose, and solve performance challenges across DB2 for z/OS environments, as well as help prevent them from reoccurring in the future. Key integration features within individual products, such as in-context launching of other tools, direct data import/export and so on, enable users to maintain problem solving context as they move from one tool to the next.

The DB2 Performance Solution Pack for z/OS is composed of the following tools.

IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS enables users to assess the efficiency of DB2 and optimize its performance, combining sophisticated response time tracking, reporting, monitoring, stored procedure, and buffer pool analysis features, as well as expert database analysis functions.

It offers the following benefits:

- ▶ Delivers end-to-end response-time measurement capability for heterogeneous applications accessing DB2 for z/OS data through SQL or stored procedures
- ▶ Includes performance metrics about the separately available IBM DB2 Analytics Accelerator usage
- ▶ Monitors, analyzes, and tunes the performance of DB2 subsystems
- ▶ Improves productivity with robust views of performance
- ▶ Identifies performance bottlenecks quickly and easily by using predefined rules
- ▶ Offers substantial breadth and depth of monitoring DB2 environments by combining batch reporting capabilities with real-time monitoring and historical tracking functions

- ▶ Supports an enterprise-wide integrated systems management strategy activated by the Tivoli OMEGAMON XE family
- ▶ Stores performance data and analysis tools in a performance warehouse

IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, V5.2 (5655-W37) and IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS, V5.2 (5655-W38) deliver the following capabilities:

- ▶ Support for DB2 11, including these features:
 - Storage accounting enhancements
 - RBA/LRSN expansion
 - Autonomous SQL procedures
 - New and enhanced DSNZPARMs
 - Enhanced client information fields
 - New and enhanced statistics for session, array variables and stored procedures
- ▶ Support for RID list enhancements
- ▶ Ability to restore rundate and jobname to utility screens; previously lost functionality due to internal DB2 changes
- ▶ Further below-the-bar memory reductions
- ▶ Deferred start capability for Application Trace (ATF); this feature is the remaining CUA-unique capability in OMEGAMON DB2 being implemented in preparation for full deprecation of the CUA
- ▶ DB2 security usage to issue DB2 commands in the enhanced 3270 UI, making it more intuitive for DBAs and DB2 users than the previously used session ID used to log on to the enhanced 3270 UI
- ▶ Performance database schema improvements
- ▶ Group Global Buffer Pool support added to the enhanced 3270 UI and classic
- ▶ Updated batch and real-time support for the IBM DB2 Analytics Accelerator
- ▶ New DB2 10 IFCID 225 (DB2 storage) metrics that enhances OMEGAMON DB2 storage displays to account for continued migration of DB2 storage to above the bar
- ▶ Spreadsheet input generator
- ▶ Aggregated accounting statistics
- ▶ DB2 locks and claims by plan
- ▶ The four-character IBM CICS® TransID is now available for active threads
- ▶ Various customer requirements
- ▶ Integration with other separately available DB2 Tools to explain and tune SQL statements or access additional SQL workload related information:
 - DB2 SQL Performance Analyzer for z/OS V4.2
 - InfoSphere Optim Query Workload Tuner for DB2 for z/OS and IBM Data Studio V4
 - InfoSphere Optim IBM pureQuery® Runtime for DB2 for z/OS V3
 - IBM WebSphere® Application Server V6.1.0.27, or later

DB2 Query Monitor for z/OS

DB2 Query Monitor for z/OS enables users to identify problem SQL activity and includes robust statistics for SQL tuning. It can work with InfoSphere Optim Query Workload Tuner for DB2 for z/OS and DB2 SQL Performance Analyzer for z/OS to enable users to customize and tune the poorly performing SQL statement or workload and to support the effectiveness of DB2 subsystems and improve overall performance:

- ▶ Monitors SQL activities and delivers automatic alerts to exceptional events on a monitored DB2 subsystem in real time
- ▶ Monitors DB2 commands, host variables, and SQL CA and SQL code in real time
- ▶ Includes user-friendly ISPF, GUI, and web interfaces with SQL statistics of multiple DB2 subsystems
- ▶ Integrates with other DB2 Tools to reduce resource consumption
- ▶ Supports single sign-on for the ISPF view of data sharing group

IBM DB2 Query Monitor for z/OS V3.2

IBM DB2 Query Monitor for z/OS V3.2 (5655-V42) delivers performance improvements, improved product integration and improved reliability, availability, and serviceability.

Performance improvements are obtained by reducing the collection overhead for fetch-intensive workloads, by expanded zIIP exploitation, and by implementing literal stripping for dynamic SQL text collection.

Product integration advancements focusing on DB2 11 for z/OS exploitation and support for the separately available DB2 Analytics Accelerator.

DB2 Query Monitor for z/OS V3.2 supports DB2 11 for z/OS. It also gives you the ability to identify accelerated queries, enabling key metrics and statistics, showcasing the return on investment for IBM DB2 Analytics Accelerator. In addition, DB2 Query Monitor for z/OS V3.2 offers enhanced integration with InfoSphere Optim Query Workload Tuner for DB2 for z/OS, giving you better end-to-end tuning for SQL queries and work- loads.

Ease-of-use improvements deliver additional value through enhanced productivity.

DB2 Query Monitor for z/OS V3.2 supports IBM Tools Customizer for z/OS, giving you a guided customization. DB2 Query Monitor for z/OS V3.2 also offers user interface improvements including displaying additional statistics on the Object Details Panel, enabling workload comparisons to track query performance over time, and enabling monitoring profiles to be changed via an operator command. The SQL Text offload program is replaced with the DB2 LOAD utility for ease of use and simplification of the process.

IBM DB2 Query Monitor for z/OS V3.2 includes the following new features:

- ▶ Additional zIIP enablement(1):
zIIP frees up general computing capacity and lowers the overall total cost of computing for select data and transaction processing workloads. By running more DB2 Query Monitor for z/OS processing on zIIP processors, the overall cost of SQL monitoring can be lowered.

With the base 3.2 delivery, DB2 Query Monitor for z/OS exploits zIIP processors for storage clean-up activity and the collection of current activity (where “current activity” is the statistical data collected by having the INFLIGHT(Y) parameter set at start up).¹

- ▶ Performance improvements for fetch-intensive workloads:
 - Increase performance of DB2 Query Monitor for z/OS collector code path.
 - Optimize for FETCH calls.
- ▶ Literal stripping:

This enhancement enables users to specify OPTKEYS(PTEXT) to strip literals and multiple white spaces from summary text.
- ▶ Additional delay statistics on objects panel

The Object Detail panel now includes statistics describing lock or latch delays, page latch delays and synchronous I/O delays. This enables you to sort on any of these delay counters and detect which object is having which type of delay problem.
- ▶ New modify commands:

Additional modify commands have been added to enable you to automate the process of changing the monitoring profile. The new modify commands enable you to activate and deactivate monitoring agents as well as to change and refresh monitoring profiles.
- ▶ Workload comparisons:

In summaries perspective, you can specify one workload as a baseline workload and then navigate to another workload (the current work- load) to compare it to the baseline. This enables you to detect and analyze many kinds of changes in workload performance metrics.
- ▶ Sharing user configurations:

User configurations can be created for filters, archive connections and staging table connections. These user configurations can be shared with other users and you can control the way the user configurations are shared through the use of labels.
- ▶ Export Activity Browser data:

DB2 Query Monitor for z/OS CAE users can download data in the current Activity Browser display in either CSV or PDF format using the Tools > Export Data option. CSV format files can be opened in a spreadsheet for further analysis.
- ▶ Load utility support for offload:

This enhancement replaces the CQM@ITXT module with the DB2 load utility. This creates faster loading of SQLTEXT, and is less problematic for the user. Internal changes enable the DB2 load utility to load the SQLTEXT into appropriate SQLTEXT tables for SQL statements less than 32 KB in length. For SQL statements greater than 32 KB in length, the COM@ITXT module is still used.
- ▶ DB2 Query Monitor V3.2 exploits DB2 11 for z/OS.

¹ This information provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g. zIIPs, zAAPs, and IFLs) (“SEs”). IBM authorizes customers to use IBM Specialty Engines (SEs) such as zIIPs, zAAPs, and IFLs only to execute the processing of Eligible Workloads of specific programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines (AUT)” provided at http://www.ibm.com/systems/support/machine_warranties/machine_code/aut.html. No other workload processing is authorized for execution on an SE. IBM offers SEs at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types or amounts of workloads as specified by IBM in the AUT.

- ▶ Support for customization:
Offers a new customization method for easier configuration. This method invokes the IBM Tools Customizer for z/OS, simplifying and consolidating many of the customization processes that are required to customize IBM Tools.
- ▶ Add interval number to interval started/ended messages.
- ▶ New batch reports:
DB2 Query Monitor for z/OS Batch Reports are enhanced in V3.2. The new batch reports are derived from queries and forms developed in IBM Query Management Facility™ (QMF). While the new batch reports offer the same information and report layout as the batch reports delivered in IBM DB2 Query Monitor V3.1, they are now delivered in a manner that enables you to use them as a base for your own customized batch reports.
- ▶ Ability to define CAE actions that launch web pages:
Users can now define actions that can be executed in the CAE web client to take users to specific web pages.
- ▶ Averages based on execution accounts:
Previously, averages in IBM DB2 Query Monitor for z/OS were calculated based on the number of SQL calls. Three new columns have been added: AvgX Elapsed, AvgX CPU, and AvgX Delays. These new columns use the execution accounts to calculate the averages.

DB2 SQL Performance Analyzer for z/OS

DB2 SQL Performance Analyzer for z/OS gives DBAs and application developers the ability to do extensive analysis of SQL queries without executing them. This analysis aids users in tuning queries to achieve maximum performance. It can also help reduce the escalating costs of database queries by estimating their cost prior to execution. Candidate queries can be passed from both DB2 Query Monitor for z/OS and Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS to facilitate the analysis of a problem query once identified.

- ▶ Performs performance analysis and recommendations for DB2 plans, packages, DBRMs, and SQL without executing them
- ▶ Includes several interfaces: Batch, ISPF, and QMF
- ▶ Offers what if scenarios to help determine the performance for various database design alternatives
- ▶ Generates comprehensive and easy-to-use performance reports to assist in analysis
- ▶ Analyzes economic cost for each query statement

Enhanced interface to DB2 Query Monitor for z/OS accepts either a DDNAME or a DSNAME when DB2 SQL Performance Analyzer for z/OS is called.

IBM DB2 SQL Performance Analyzer for z/OS, V4.2 (5655-W60) delivers an extensive analysis of SQL queries without executing them. This analysis aids you in tuning your queries to achieve maximum performance. The functional, usability, and reporting enhancements include these:

- ▶ Customization enhancements for better ease of use with Tools Customizer for z/OS:
 - A more intuitive categorization of the customization parameters
 - Customization support for specifying DB2 objects be created in UNICODE
 - Customization support for JES3
 - Customization support to enable users to specify the volume to be used for the dynamic allocation of data sets

- ▶ Key functional enhancements:
 - Enables users to display existing reports without performing another EXPLAIN. This saves time and the processor costs of performing the associated EXPLAINS.
 - Provides additional new statistics to migration process so that the chosen access paths will match on both systems that are used in the migration.
 - Merges EEE parameters with the user parameters. This simplifies the management of parameters by consolidating the parameter files.
 - Enables users to override installation-level parameters by creating their own user-level user parameter files. This also supports the use of these same user-level parameters for each session of SQL PA. In addition, it supports users having their own job cards saved.
 - With the use of APPLID, product profile variables integration with other products is improved. The migration of existing profile variables is transparent to the user.
 - Includes a virtual index feature/function for what-if analysis.
 - Enables you to create virtual indexes for doing “what-if” analysis. With virtual indexes, there is typically minimal impact on performance and less overhead.
- ▶ Exploitation of new DB2 11 for z/OS features.

InfoSphere Optim Query Workload Tuner for DB2 for z/OS

InfoSphere Optim Query Workload Tuner for DB2 for z/OS empowers users to more efficiently manage performance by proactively optimizing the performance of SQL queries and query workloads. InfoSphere Optim Query Workload Tuner for DB2 for z/OS makes it easy to access candidate queries and define workloads from a number of common sources.

It is integrated with other separately available DB2 Tools, such as DB2 Query Monitor for z/OS and Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS to enable a more complete solution for monitoring and tuning SQL. This integration helps organizations reduce the risk of customer complaints, missed SLAs, and financial losses due to performance problems, as follows:

- ▶ Analyzes SQL queries and workloads to proactively optimize physical database design and improve performance
- ▶ Helps to proactively solve query performance problems by enabling faster and deeper analysis of queries
- ▶ Offers expert recommendations to help maximize application performance while reducing specialized skill requirements
- ▶ Helps reduce the total cost of ownership for DB2 for z/OS
- ▶ Reduces CPU usage for statistics collection by collecting only what is helpful to improve performance
- ▶ Tunes queries holistically with integration with other separately available DB2 performance tools

All the publications for DB2 Tools are available at the following website:

<http://www.ibm.com/support/docview.wss?uid=swg27020910>



DB2 traces

DB2 traces provide the instrumentation you need to monitor, analyze, and tune your DB2 subsystem and the applications it runs.

A trace record represents a DB2 event, and is the single smallest unit of tracing. DB2 has hundreds of individual trace records, each one known by an instrumentation facility component identifier (IFCID). Documentation on all of the DB2 trace records by IFCID is in prefix.SDSNIVPD(DSNWMSGs).

A DB2 trace record is categorized into one of six types of traces: accounting, statistics, performance, monitor, audit, and global. These types are further grouped into individual classes that you can use on the **start trace** command. You can choose one of several destinations to write these trace classes to, depending on the amount of tracing you will be doing and the purpose of the trace.

This chapter helps you to understand trace type, trace classes, trace destinations, and the overhead they can cause on your system. It also helps you decide which traces to start and how long to keep them running.

This chapter contains the following topics:

- ▶ Trace types
- ▶ Trace destinations
- ▶ Managing DB2 traces
- ▶ Compressing your trace records

2.1 Trace types

The following DB2 trace types are included:

- ▶ Accounting
- ▶ Statistics trace
- ▶ Performance trace
- ▶ Audit trace
- ▶ Monitor trace
- ▶ Global

Accounting

Accounting trace records contain thread execution information such as CPU consumed, waiting times, SQL executions, number of lock events, commits, buffer pool requests, resource limit facility data, DDF processes, RID pool information, and start and stop times. The performance impact of accounting trace is up to 5% for starting classes 1, 2 and 3.

The accounting trace classes are summarized in Figure 2-1. Chapter 82. -START TRACE(DB2) in *DB2 11 for z/OS Command Reference*, SC19-4054 lists the IFCIDs that DB2 starts with each trace class.

<p>Class 1 - Standard accounting information Class 2 - Entry or exit from DB2 events Class 3 - Elapsed wait time in DB2 Class 4 - Installation-defined accounting record Class 5 - Time spent processing IFI requests Class 6 - Reserved Class 7 - Entry or exit from event signaling package accounting Class 8 - Wait time for a package Class 10 - Package Detail Class 11 - Plan-level only (V11) Class 12 to 29 - Reserved Class 30 to 32 - Available for local use</p>
--

Figure 2-1 Accounting trace classes

Statistics trace

Statistics trace provides a system-wide view, which limits details, but its overhead is very small. It is usually set to start during DB2 start up. Prior to DB2 Version 10, the DSNZPARM STATIME controls the interval at which all statistics records are written. In DB2 Version 10, most of the statistics trace records are written at fixed, one-minute intervals, and STATIME is only used for the exceptions (IFCIDs 0105, 0106, 0199, and 0365).

Figure 2-2 lists the statistics trace classes.

Class 1 - System services, database statistics, and subsystem parameters
Class 2 - Installation-defined statistics record
Class 3 - Deadlocks, lock escalation, group buffers, data set extension, long-running units of recovery, and active log shortage
Class 4 - Exceptional conditions
Class 5 - Data sharing statistics
Class 6 - Storage usage
Class 7 - DRDA location statistics
Class 8 - Data set I/O statistics
Class 9 - Aggregate CPU and wait time statistics by connection type
Class 10 to 29 - Reserved
Class 30 to 32 - Available for local use

Figure 2-2 Statistics Trace Classes

Performance trace

Performance trace should be started for a very limited period of time and to address specific performance problems. Performance trace is expensive and can generate more than 100% overhead, but you can minimize this cost by narrowing the scope of the trace using filters such as AUTHID, PLAN, LOCATION and so on. Figure 2-3 shows the performance trace classes.

Class 1 - Background events
Class 2 - Subsystem events
Class 3 - SQL events
Class 4 - Reads to and writes from buffer pools and the EDM pool
Class 5 - Writes to log or archive log
Class 6 - Summary lock information
Class 7 - Detail lock information
Class 8 - Data scanning detail
Class 9 - Sort detail
Class 10 - Detail on BIND, commands, and utilities
Class 11 - Execution unit switch and latch contentions
Class 12 - Storage manager
Class 13 - Edit and validation exits
Class 14 - Entry from and exit to an application
Class 15 - Installation-defined performance trace record
Class 16 - Distributed processing
Class 17 - Claim and drain information
Class 18 - Event-based console messages
Class 19 - Data set open and close activity
Class 20 - Data sharing coherency summary
Class 21 - Data sharing coherency detail
Class 22 - Access control auth exit parameters
Class 23 - Reserved
Class 24 - Stored procedure detail
Class 25 to 29 - Reserved
Class 30 to 32 - Available for local use

Figure 2-3 Performance trace classes

Audit trace

Audit trace contains audit data for applications such as unauthorized access attempts, documentation of operations against audited objects, and user ID activity. The performance impact of audit trace is typically less than 5%. Figure 2-4 shows the audit trace classes.

Class 1 - Attempted access denied due to lack of authority
Class 2 - Explicit GRANT and REVOKE statements
Class 3 - CREATE, ALTER, and DROP statements against audited tables
Class 4 - First change made to an audited object
Class 5 - First read made against an audited object
Class 6 - BIND information for SQL statements on audited objects
Class 7 - Assignment or change of an AUTHID
Class 8 - Utility execution
Class 9 - Installation-defined audit trace record
Class 10 - Trusted context information
Class 11 - Audit administrative authorities
Class 12 to 29 - Reserved
Class 30 to 32 - Available for local use

Figure 2-4 Audit trace classes

Monitor trace

Monitor trace is used for online monitors such as OMEGAMON. It collect data about DB2 resource usage and also the same data as accounting trace such as application elapsed time, time spent in DB2, and wait time. The performance impact of monitor trace is typically up to 5% for starting classes 1, 2, and 3. Figure 2-5 shows the monitor trace classes.

Class 1 - Standard accounting data
Class 2 - Entry or exit from DB2 events
Class 3 - DB2 wait for I/O or locks
Class 4 - Installation-defined monitor trace record
Class 5 - Time spent processing IFI requests
Class 6 - Changes to tables created with DATA CAPTURE CHANGES
Class 7 - Entry or exit from event signaling package accounting
Class 8 - Wait time for a package
Class 9 - Enables statement level accounting
Class 10 - Package detail for buffer manager, lock manager and SQL statistics
Class 11 to 28 - Reserved
Class 29 - Controls the subsystem-wide collection of statistics for SQL statements.
Class 30 to 32 - Available for local use

Figure 2-5 Monitor trace classes

Global

Global trace provides a very detailed trace having to do with path length and DB2 module entry and exit. It generates so much data that it can cause system issues, so do not start global trace without guidance from IBM service.

Summary

DB2 externalizes trace records based on these parameters:

- ▶ A timer (statistics)
- ▶ An event (accounting, performance, audit)
- ▶ A real-time request (monitor)

We recommend running with the following accounting and statistics traces on:

```
-START TRACE(ACCTG) CLASS(1,2,3,7,8)
-START TRACE(STAT) CLASS(1,3,4,5,8)
```

2.2 Trace destinations

Here are the available traces destinations, which specify where the trace records are written:

SMF

System Management Facility (SMF) is a z/OS service aid used to collect information from various z/OS subsystems. SMF data is placed into buffers and then is written into data sets. SMF records that are generated by DB2 are all either SMF type 100, 101, or 102. In general, SMF 100 records contain statistics information, and are relatively small compared to SMF 101 records, which contain accounting data.

Figure 2-6 shows what IFCIDs the SMF record types contain. The table in Figure 2-6 maps the DB2 IFCIDs to their respective SMF record types.

IFCID	Description	SMF Record Type
1	System Services Statistics	100
2	Database Services Statistics	100
3	Agent Accounting	101
202	Dynamic System Parameters	100
225	System Storage Summary Statistics	100
230	Data Sharing Global Statistics	100
239	Agent Accounting Overflow	101
All others		102

Figure 2-6 SMF record types

GTF

GTF is a z/OS service aid that collects information to analyze particular situations. GTF can also be used to analyze seek times and supervisor call instruction (SVC) usage, and for other services. DB2 trace records can be written to GTF. In particular, this destination is appropriate for large volumes of trace records, such as when you start a performance trace.

SRV

An SRV is an exit to a user-written routine. For instructions and an example of how to write such a routine, see the macro DSNWVSER in library prefix SDSNMACS.

OPn

OPn is a specific buffer destination. Only applications that start a trace to an OPn buffer can read that buffer. Online monitors are the principle users of these destinations. An OP trace destination is limited in memory (64 MB in DB2 10 and DB2 11) and it may be overrun if the monitor program does not empty the buffer.

OPx

OPx is a generic destination that uses the first free OPn slot.

2.3 Managing DB2 traces

In this section, we discuss the following topics:

- ▶ Start trace
- ▶ Display trace
- ▶ Modify trace
- ▶ Stop trace
- ▶ Loading IFCID field descriptions into a table
- ▶ Reporting performance data

2.3.1 Start trace

You can start DB2 traces in two ways:

- ▶ By specifying the appropriate DSNZPARMs to automatically start the trace when DB2 starts. The DSNZPARM only allows automatic start of statistics, accounting, and monitor traces. Performance traces must be explicitly started with the **-START TRACE** command.
- ▶ By using the **-START TRACE** command to initiate specific traces when DB2 is already running. The basic **-START TRACE** command syntax is shown in Figure 2-7. For details, see *DB2 11 for z/OS Command Reference*, SC19-4054.

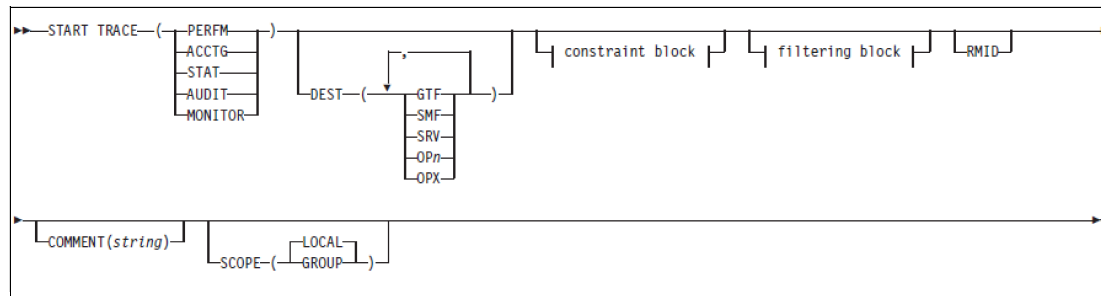


Figure 2-7 Start trace syntax diagram

The SCOPE LOCAL or GROUP means collecting data for local member only or collecting data for all members of a data sharing group.

It is also possible to have multiple traces of the same type, to different destinations, but only 32 traces can run simultaneously.

The constraint and filtering blocks place optional limits on the kinds of data that are collected by the trace. You can apply constraint or filter options to any type of trace except statistics traces. However, note that some IFCIDs ignore filtering options. This is because most of the available filters are meant to filter specific transactions, and some of the non-statistics IFCIDs do not apply at the transaction level. For example, IFCID 373 provides the DSNHDECP load module name for the subsystem, so a transaction filter would not be applicable.

You can also specify a partial name, ending with "*", for example, PLAN(PAOLOPL*), but you cannot specify "*" in the beginning as PLAN(*PL). If that is the case, you can use "_" (underscore) for a single character in the name. There is also the possibility to use eXclude qualifiers (XAUTHID, XPLAN, XPKGLOC, XPKGCOL, XPKGPROG, XLOC, XUSERID, XAPPNAME, XWRKSTN, XCONNID, XCORRID, and XROLE).

Example 2-1 shows a **START TRACE** accounting command for classes 1, 2, 3, 7, 8 with destination SMF.

Example 2-1 START TRACE for accounting

```
-START TRACE(ACCTG) CLASS(1,2,3,7,8) DEST(SMF)
```

Command output

```
DSNW130I  -DB1A ACCTG TRACE STARTED, ASSIGNED TRACE NUMBER 01
DSN9022I  -DB1A DSNWVCM1 '-START TRACE' NORMAL COMPLETION
```

If you only want to trace certain IFCIDs, use an “empty” trace class. That is a class that has no IFCIDs assigned to it by default. For example, for the performance trace, these are classes 29, 30, 31, 32. if you need to only trace IFCID 42,43 (system checkpoint sta/end), use this:

- STA TRA(P) C(32) IFCID(42,43)

The TDATA keyword is used to specify which trace headers you want DB2 to use when writing the trace record. The default trace headers depend on the trace type, so the best practice is to specify them to make sure you have the headers you want. The standard header cannot be specified because they are always present (see DSNDQWHS). The trace header is added for serviceability records, TDATA (COR,TRA,CPU,DIST):

- ▶ COR: Correlation header, it includes connection type, correlation-id, authid, planname, workstation-name and so on (see DSNDQWHC)
- ▶ DIST: Distributed header (see DSNDQWHD)
- ▶ TRA: Trace header, it is mostly used for R14 info in combination with global trace (see DSNDQWHT)
- ▶ CPU: CPU header, CPU time of the currently dispatched execution unit (TCB or SRB), includes (“normalized”) CPU consumed on a specialty engine. It is important to specify for performance traces (see DSNDQWHU)

As an example of the use of the TDATA keyword: If you want to identify the actual latches that are triggering the latch contention. the following trace command will gather this information:

```
-STA TRA(P) CLASS(30) IFCID(56,57) DEST(SMF/GTF) TDATA(CPU COR DIST)
```

2.3.2 Display trace

The **-DISPLAY TRACE** command shows the relevant active trace information. This helps you find the DB2 assigned trace number (TNO), which you need in order to be able to modify or stop a trace. See Example 2-2.

Example 2-2 DISPLAY TRACE

```
-DIS TRACE
```

Command output

```
DSNW127I  -DB1A CURRENT TRACE ACTIVITY IS -
TNO TYPE  CLASS      DEST QUAL IFCID
03  STAT   01,03,04,05, SMF  NO
03              06,08
04  ACCTG   01,02,03,07, SMF  NO
04              08
*****END OF DISPLAY TRACE SUMMARY DATA*****
DSN9022I  -DB1A DSNWVCM1 '-DIS TRACE' NORMAL COMPLETION
***
```

2.3.3 Modify trace

The **-MODIFY TRACE** command actually stops an active trace and starts a new one with the specified new options. See Example 2-3.

Example 2-3 MODIFY TRACE

```
-MODIFY TRACE(A) CLASS(1,2,3,7,8) TNO(4)
```

Command output

```
DSNW130I  -DB1A A TRACE STARTED, ASSIGNED TRACE NUMBER 04
DSN9022I  -DB1A DSNWVCM1 '-MODIFY TRACE' NORMAL COMPLETION
***
```

2.3.4 Stop trace

A trace can be stopped by specifying enough information to identify the particular trace entry. See Example 2-4.

The usual way to stop a trace is to first issue a **DISPLAY** command to check the TNO, then issue a **STOP**.

Example 2-4 STOP TRACE

```
-STOP TRACE(A) TNO(4)
```

Command output

```
DSNW131I  -DB1A STOP TRACE SUCCESSFUL FOR TRACE NUMBER(S) 04
DSN9022I  -DB1A DSNWVCM1 '-STOP TRACE' NORMAL COMPLETION
***
```

2.3.5 Loading IFCID field descriptions into a table

In this section, we show how to load IFCID descriptions into a user-defined DB2 table. This approach has the advantage of providing you with access to the IFCID field descriptions through SQL SELECT statements in whatever order or format you choose.

Before loading the data, you need to create tables for the data. Example 2-5 shows the DDL statements to create a table space and the tables you need. We use DESCRIPTIONS and TYPES. Before you attempt to create these objects, ensure that you have sufficient authority to do it.

Example 2-5 Create IFCID table

```
CREATE TABLESPACE TRACETS
  IN DSN8D11A
  USING STOGROUP DSN8G110;

CREATE TABLE DSN81110.TRACE_DESCRIPTIONS
  (IFCID      INTEGER NOT NULL,
   NAME       CHAR(20) NOT NULL,
   DESCRIPTION CHAR(60) NOT NULL,
   SEQ        INTEGER NOT NULL)
  IN DSN8D11A.TRACETS;
```

```

CREATE TABLE DSN81110.TRACE_TYPES
  (TYPE      CHAR(17) NOT NULL,
   CLASS     INTEGER NOT NULL,
   IFCID      INTEGER NOT NULL,
   COMMENTS  CHAR(45) NOT NULL WITH DEFAULT)
IN DSN8D11A.TRACETS;

```

After creating the tables, you need to load the data. See Example 2-6.

Example 2-6 LOAD IFCID table job

```

/*JOB
//PROCLIB JCLLIB ORDER=DB1AM.PROCLIB
//LOAD EXEC DSNUPROC,SYSTEM=DB1A,
//          LIB='DB1AT.SDSNLOAD',
//          UID='LOADPP' UTPROC='PREVIEW'
//DSNUPROC.SYSREC DD DISP=SHR,DSN=DB1AT.SDSNIVPD(DSNWMSGGS)
//DSNUPROC.SYSIN DD *
      TEMPLATE COPY DSN 'DB2R2.&DB..&TS..T&TIME.'
      DISP (NEW,CATLG,DELETE)
      UNIT SYSDA
      SPACE (5,5) CYL
      TEMPLATE UT1 DSN 'DB2R2.&DB..&TS..SYSUT1'
      DISP (NEW,DELETE,DELETE)
      UNIT SYSDA
      SPACE (5,5) CYL
      TEMPLATE SRTOUT DSN 'DB2R2.&DB..&TS..SORTOUT'
      DISP (NEW,DELETE,DELETE)
      UNIT SYSDA
      SPACE (5,5) CYL
      TEMPLATE DIS DSN 'DB2R2.&DB..&TS..DISDD'
      DISP (NEW,DELETE,DELETE)
      UNIT SYSDA
      SPACE (1,1) CYL
      TEMPLATE ERR DSN 'DB2R2.&DB..&TS..ERRDD'
      DISP (NEW,DELETE,DELETE)
      UNIT SYSDA
      SPACE (1,1) CYL
LOAD DATA INDDN(SYSREC) LOG(NO) REPLACE PARALLEL
EBCDIC CCSID(00037,00000,00000)
WORKDDN(UT1,SRTOUT) DISCARDS 0 DISCARDN DIS ERRDDN ERR
COPYDDN COPY STATISTICS TABLE(ALL) INDEX(ALL)
INTO TABLE DSN81110.TRACE_DESCRIPTIONS WHEN (1) = '0'
(IFCID      POSITION(1:4)  INTEGER EXTERNAL(4),
 NAME       POSITION(6:25) CHAR(20),
 DESCRIPTION POSITION(27:86) CHAR(60),
 SEQ        POSITION(87:92) INTEGER EXTERNAL(6) )
INTO TABLE DSN81110.TRACE_TYPES WHEN (1) = '1'
(TYPE       POSITION(3:19) CHAR(17),
 CLASS      POSITION(27:28) INTEGER EXTERNAL(2),
 IFCID      POSITION(35:39) INTEGER EXTERNAL(5),
 COMMENTS   POSITION(42:86) CHAR(45) )

```

Example 2-7 shows the number of rows loaded from the LOAD job output.

Example 2-7 LOAD IFCID table job output

```
NUMBER OF RECORDS=25139 FOR TABLE DSN81110.TRACE_DESCRIPTIONS
NUMBER OF RECORDS=527 FOR TABLE DSN81110.TRACE_TYPES
```

After loading the tables, you can use SQL to retrieve the trace field descriptions in whatever format or order you need. You can execute the SQL statements as you would any other SQL (for example, through SPUFI, QMF, or DSNTPEP2).

Some sample SQL statements follow that might be useful in your analysis of DB2 trace records. Consider printing a formatted report of the results from the first three queries below; they might provide useful reference material for your system performance analyst or auditor:

- To retrieve all trace field descriptions, ordered numerically by IFCID:

```
select distinct *
      from dsn81110.trace_descriptions
      order by seq;
```

- To retrieve all trace field descriptions, ordered alphabetically by field name:

```
select distinct *
      from dsn81110.trace_descriptions
      order by name, seq;
```

- To retrieve all trace field descriptions, ordered by trace type and trace class:

```
select type, class, a.ifcid, description, seq
      from dsn81110.trace_types a,
      dsn81110.trace_descriptions b
      where a.ifcid = b.ifcid
      order by type, class, seq;
```

- To retrieve only the field descriptions for trace record IFCID 21:

```
select distinct *
      from dsn81110.trace_descriptions
      where ifcid = 21
      order by seq;
```

- To retrieve only the field description for field qwacriniv:

```
select distinct *
      from dsn81110.trace_descriptions
      where name = 'qwacriniv'
      order by seq;
```

- To retrieve field descriptions for audit trace records:

```
select distinct name, description, a.ifcid, seq
      from dsn81110.trace_descriptions a,
      dsn81110.trace_types b
      where a.ifcid = b.ifcid
      and type = 'audit'
      order by seq;
```


- To retrieve field descriptions for monitor trace records:

```
select distinct a.name, a.description,
               a.ifcid, a.seq
  from dsn81110.trace_descriptions a,
       dsn81110.trace_types b
 where a.ifcid = b.ifcid
       and type = 'monitor'
 order by seq;
```

- To retrieve field descriptions for performance trace records, classes 1 through 7:

```
select distinct name, description, a.ifcid, seq
  from dsn81110.trace_descriptions a ,
       dsn81110.trace_types b
 where a.ifcid = b.ifcid and
       (type = 'performance' and
        class in (1,2,3,4,5,6,7))
 order by seq;
```

2.3.6 Reporting performance data

After collecting DB2 trace data, the data can be filtered, sorted, accumulated, and grouped according to your requirements, such as plan name, correlation ID, and connection ID.

2.4 Compressing your trace records

When traces are activated, especially audit policies or performance traces, the volume of data directed to SMF can be very large.

2.4.1 Software compression

DB2 10 introduced the DSNZPARM parameter, SMFCOMP, which directs DB2 to request compression for trace records that are sent to SMF. Trace data to GTF and OPX is not compressed. SMFCOMP is specified on the installation panel DSNTIPN in the field COMPRESS SMF RECS. The default value is OFF.

If compression is enabled, SMF data is compressed such that everything after the SMF header (SM100END, SM101END, or SM102END) is compressed with z/OS compression service CSRCESTRV. A compressed record is identified by a bit in the SMF100, 101, and 102 headers. The trade-off for this function is SMF volume versus a marginal increase in CPU to compress and expand the records.

Performance measurements show a minimal impact of up to 1% with accounting class 1, 2, 3, 7, 8, and 10 active. The disk savings for DB2 SMF data set can be significant with a compression rate of 60% to 80%. Our results showed statistics classes 1, 3, 4, 5, and 6 with a compression rate around 60%.

If you use your own trace formatter, instead of OMEGAMON PE, you need to call the z/OS compression service, which is turned off by default, to decompress the data. APAR PM27872 provided DB2 10 with decompression routine DSNTSMFD and a sample JCL, DSNTSJDS, to execute it. DSNTSMFD takes SMF data as an input to decompress. DSNTSMFD gives you an output message of the trace records distribution with how much your data was compressed and the percentage saved by the compression.

2.4.2 Hardware compression

The IBM System z® platform has a new I/O adapter that compresses and decompresses data using the industry-standard Deflate algorithm. Supported by the IBM zEnterprise® Data Compression (zEDC) feature of z/OS V2.1, the zEDC Express adapter is optional with the zEnterprise EC12 and zEnterprise BC12. It can compress data at more than 1 GB per second while using roughly 100 times less CPU than software implementations of the same algorithm.

zEDC Express gets its enormous throughput and power efficiency by using dedicated compression hardware directly optimized for the Deflate compression algorithm.

For z/OS V2R1, the zEDC software feature must be enabled with your product enablement policy (IFAPRDxx).

Systems Management Facility (SMF) provides the system, applications, and middleware with a standard method for writing out accounting, auditing, monitoring, performance, and application records. A new option for compressing SMF data sent to the system logger enables zEDC Express compression of SMF data. Compression reduces the amount of data written, increasing the throughput and lowering the CPU cost of SMF logging. SMF data is decompressed when read from the logger to maintain compatibility with existing software.

Users who want archival compression of SMF data can store it on compressed media, including tape and compressed extended-format sequential data sets, after it is read from the logger.

This compression is *independent of DB2* and can be activated on the LPAR where the I/O Adapter is installed by adding the COMPRESS keyword to the stream definition in SMFPRMxx, as shown in Example 2-8.

Example 2-8 Definition in SMFPRMxx

```
DEFAULTLSNAME( IFASMF.DEFAULT,COMPRESS,DSPSIZMAX(200M) )
```

In order to monitor PCIe¹ adapter card function and zEDC activity, a new IBM z/OS IBM Resource Measurement Facility™ (IBM RMF™) Postprocessor PCIE Activity Report is available in XML output format to provide measurements about the activity of PCI Express based functions (PCIe functions) and their exploitation of hardware accelerators.

¹ Peripheral Component Interconnect Express (PCIe) is a high-speed serial computer expansion bus standard created by Intel, Dell, HP, and IBM.

Figure 2-8 shows an example of PCIe function and zEDC activity report.

RMF Data Portal for z/OS

[Home](#)

[Explore](#)

[Overview](#)

[My View](#)

[?](#)

RMF

RMF Postprocessor Interval Report [System SC63] : PCIe Activity Report

RMF Version : z/OS V2R1

SMF Data : z/OS V2R1

Start : 10/09/2013-14.14.36

End : 10/09/2013-14.29.36

Interval : 15.00.000 minutes

General PCIe Activity

Function ID	Function PCHID	Function Name	Function Type	Function Status	Owner Job Name	Owner Address Space ID	Function Allocation Time	PCI Load Operations Rate	PCI Store Operations Rate	PCI Store Block Operations Rate	Refresh PCI Translations Operations Rate	DMA Address Space Count	DMA Read Data Transfer Rate	DMA Write Data Transfer Rate
0023	0578	Hardware Accelerator	1014044B	Allocated	FPGHWAM	0013	960	0	33.2	0	584	1	1382	219

Hardware Accelerator Activity

Function ID	Time Busy %	Request Execution Time	Std Dev for Request Execution Time	Request Queue Time	Std Dev for Request Queue Time	Request Size	Transfer Rate Total
0023	1.84	31.5	1.35	622	662	80.3	46.9

Hardware Accelerator Compression Activity

Function ID	Compression Request Rate	Compression Throughput	Compression Ratio	Decompression Request Rate	Decompression Throughput	Decompression Ratio	Buffer Pool Size	Buffer Pool Utilization
0023	584	38.2	4.33	0	0	64	0	

Figure 2-8 PCIe function and zEDC activity report

The PCIe Activity Report is divided into three sections:

- ▶ General PCIe Activity
- ▶ Hardware Accelerator Activity
- ▶ Hardware Accelerator Compression Activity

The General PCIe Activity section shows measurements for all PCIe functions independent from the type of the exploited hardware feature. The measurements reflect the activity of the z/OS system on which RMF data collection took place.

The Hardware Accelerator Activity section and the Hardware Accelerator Compression Activity section have single system scope and are leveraging the measurements displayed in the General PCIe Activity section. They are only displayed if the hardware feature zEnterprise Data Compression (zEDC) is used for compression acceleration. It displays these metrics:

- ▶ Common accelerator metrics, for example, total request execution time, or the amount of transferred data
- ▶ Compression specific metrics, for example, the amount of compressed data and the number and throughput of compression requests

Table 2-1 shows the field descriptions of the PCIe Activity Report.

Table 2-1 General PCIe Activity

Field heading	Meaning
Function ID	Identifier of the monitored PCIe function. This field also identifies applicable PCIe functions in the Hardware Accelerator Activity and Hardware Accelerator Compression Activity sections.
Function PCHID	Physical channel identifier for the PCIe function.
Function Name	Device name for the PCIe function.
Function Type	Device type for the PCIe function.

Field heading	Meaning
Function Status	The PCIE function status can be one of the following values: Allocated: The function is allocated and in use at the end of the reporting interval. Re-Allocated: The function was de-allocated during the interval but has been re-allocated again. It is in use at the end of the reporting interval. De-Allocated: The function was de-allocated during the interval and is unused at the end of the reporting interval. De-Allocate-Pending: The function is in the process of de-allocation. Error: The function is in permanent error status. Unknown: The function status is unknown.
Owner Job Name	Job name of the owner who allocated the PCIE function.
Owner Address Space ID	Address space ID of the owner who allocated the PCIE function.
Function Allocation Time	Time in seconds for which the PCIE function was allocated or de-allocate-pending during this interval.
PCI Load Operations Rate	Rate of PCI Load operations executed during the reporting interval.
PCI Store Operations Rate	Rate of PCI Store operations executed during the reporting interval.
PCI Store Block Operations Rate	Rate of PCI Store Block operations executed during the reporting interval.
Refresh PCI Translations Operations Rate	Rate of Refresh PCI Translations operations executed during the reporting interval.
DMA Address Space Count	Number of defined DMA address spaces.
DMA Read Data Transfer Rate	Number of megabytes per second transferred by DMA reads from all defined DMA address spaces to the PCI function.
DMA Write Data Transfer Rate	Number of megabytes per second transferred by DMA writes from the PCI function to all defined DMA address spaces.
Function Allocation Time	Time in seconds for which the PCIE function was allocated or de-allocate-pending during this interval.
PCI Load Operations Rate	Rate of PCI Load operations executed during the reporting interval.
PCI Store Operations Rate	Rate of PCI Store operations executed during the reporting interval.
PCI Store Block Operations Rate	Rate of PCI Store Block operations executed during the reporting interval.
Refresh PCI Translations Operations Rate	Rate of Refresh PCI Translations operations executed during the reporting interval.
DMA Address Space Count	Number of defined DMA address spaces.
DMA Read Data Transfer Rate	Number of megabytes per second transferred by DMA reads from all defined DMA address spaces to the PCI function.

Field heading	Meaning
DMA Write Data Transfer Rate	Number of megabytes per second transferred by DMA writes from the PCI function to all defined DMA address spaces.

Table 2-2 shows the field descriptions of the Hardware Accelerator Activity.

Table 2-2 Hardware Accelerator Activity

Field heading	Meaning
Time Busy %	The percentage of time that this partition kept the hardware accelerator busy.
Request Execution Time	The average time in microseconds the hardware accelerator used to process a request.
Std Dev for Request Execution Time	The standard deviation of the request execution time.
Request Queue Time	The average queue time in microseconds that was spent for a request. This value has single system scope but is affected by activity from other partitions sharing the hardware accelerator.
Std Dev for Request Queue Time	The standard deviation of the request queue time.
Request Size	The average number of kilobytes transferred per request.
Transfer Rate Total	The number of megabytes per second transferred by DMA operations.

Table 2-3 shows the field descriptions of the Hardware Accelerator Compression Activity.

Table 2-3 Hardware Accelerator Compression Activity

Field heading	Meaning
Compression Request Rate	The number of compression requests per second.
Compression Throughput	The number of megabytes compressed per second.
Compression Ratio	The ratio between input and output bytes compressed within this interval.
Decompression Request Rate	The number of decompression requests per second.
Decompression Throughput	The number of megabytes decompressed per second.
Decompression Ratio	The ratio between input and output bytes decompressed within this interval.
Buffer Pool Size	The total size of memory in megabytes that is allocated to the buffer pool.
Buffer Pool Utilization	The average utilization of the buffer pool that z/OS kept for in-use buffers.

For further information about PCIe and zEDC performance monitoring, see *z/OS Resource Measurement Facility User's Guide Version 2 Release 1*, SC34-2664.

Note: This hardware compression can coexist with the DB2 invoked software compression described at 2.4.1, “Software compression” on page 23.



System z related information

In this chapter, we describe key components of the z/OS infrastructure to be addressed as part of a comprehensive DB2 performance strategy.

Workload Manager (WLM) is a key ingredient to the success of the DB2 subsystem as well as the work that depends on it. WLM is your defense against CPU constraint and competing workloads, and you need to be able to report on it as well.

We also provide a brief description of other tools to analyze RMF data as well as which components it can look at.

This chapter contains the following topics:

- ▶ Workload Manager
- ▶ The DB2 subsystem
- ▶ Thread level
- ▶ General WLM best practices
- ▶ Looking at RMF data

3.1 Workload Manager

Workload Manager (WLM) is the operating system component responsible for managing all the work in your z/OS environment. WLM uses workload classification, service classes, and performance objectives to manage the resources of the z/OS system to meet your business objectives. WLM and the service classes defined for DB2 and the workload dependent on DB2 can seriously impact workload execution and service levels. There are some terms within WLM that you need to be familiar with, including the types of goals, and how goal attainment is measured.

- Here are some examples of service classification goals:

- Average response time goal, for example. you want all of the online transactions to complete in an average of 1 second.

This forces the outlier transactions to conform. It can be very difficult to set accurately, due to the natural skew of the response times.

- Percentile response time goal, for example. you want 90% of the transactions to complete within 1 second.

This allows for the 10 percentile of transactions that cannot make this goal to be ignored and is typical for IBM IMS™ or CICS transactions.

Note: However, if these 10% of transactions fit into an entirely different performance profile, and are predictable, then they should be put into a different service class altogether. See 3.3.3, “Transaction level goal adjustment” on page 38.

- A goal of IBM Velocity™ is a measure of acceptable CPU, I/O (with I/O MGMT= YES), and storage delays. So if you have a velocity goal of 50, it means you can accept the fact that the workload will only get the resources it needs 50% of the time. If you set a velocity goal of 90, then you are saying that 90% of the time when this service class asks for resources, then it should get them. This is under the pretext of resource constraint as it competes with other workloads.

Velocity goals are appropriate for address spaces such as those in DB2, CICS TORs, IMS regions, and batch jobs in general:

$$\text{velocity} = (\text{using CPU} + \text{using I/O}) / (\text{using CPU} + \text{using I/O} + \text{delayed by CPU} + \text{delayed by I/O} + \text{delayed by storage})$$

- Performance index has the following considerations:

The performance index is a rating of how well the WLM goal was attained or how far it was from being met. For instance, if my goal was an average response time of 1 second, but the average actually achieved was 2 seconds, then my performance index = 2. So in essence, I only received half of the resources I needed (velocity goal) or I took twice as long as I was supposed to (response time goal). If my goal was 1 second, but the average response time was 0.5 seconds, then my performance index is 0.5. So actually a lower number is better here.

For customers running a sysplex, there is a local performance index calculated for each service class period, as well as a sysplex performance index. This can be useful when drilling down to transaction response time issues since it can be used to determine if a transaction is suffering on one particular LPAR or not. Within the WLM activity report, you can see which LPARs the service or report class executed on, and its relative performance index there.

- Period is the smallest distinguishable unit of a service class:

You can implement multiple periods in a service class to “punish” longer running executions. For instance, maybe there are queries coming from QMF, some of which are made up of simple SQL, while other queries do large reporting tasks. With multiple periods, you would specify the duration in service units in which the unit of work must complete before it is demoted to the next period. Maybe for the first period, this QMF work starts as importance 3 with a goal of 90% of the transactions finishing in 1 second. Then, after it has exhausted the 500000 service units, which constitutes its duration, then it gets moved to importance 3 to get out of the way for other importance 2 work (Example 3-1).

Example 3-1 QMF service class

* Service Class DDFST - DDF Test for DB
 Base goal:
 CPU Critical flag: NO

The details are shown in Table 3-1.

Table 3-1 Importance levels and goals

#	Duration	Imp	Goal description
1	50000	3	90% complete within 00:00:01.000
2		4	Execution Velocity of 30

The best way to see the number of service units the unit of work consumes is by looking in the WLM activity report (see Example 3-2) so you can see the total, which is made up as follows:

CPU + SRB + IO (service) + MSO (memory)

Example 3-2 Service units shown in WLM activity report

```

---SERVICE---
IOC   515
CPU  181501
MSO    0
SRB   1347
TOT  183363
/SEC   204
  
```

- Importance is defined as follows:

Every unit of work that WLM manages is rated on relative importance. Relative importance runs from 1 to 5, where 1 is the most important work and 5 is the least. WLM will always try to ensure that the units of work achieve their goal, as long as they have one.

- SYSSTC and SYSTEM are classifications that run at a higher priority than anything with a velocity or response time goal, so they are out of the scope of this discussion.
- Discretionary work is work that has a fixed performance index of 0.81 and runs at a relative importance of 6, so it always appears to WLM to be exceeding its goal while being the least important work on the system. The only way to ensure that it gets processing cycles is to include it in a resource group with a minimum setting for service units.

Tip: We suggest avoiding the use of resource group capping for any workload entering DB2, and believe that proper stratification within the WLM policy is the best approach. See Figure 3-3 on page 38.

WLM works on a prioritized donor/receiver concept. Based on the performance index (PI) values and the reason for delay (CPU, for example), a service class period can be selected as a CPU receiver, and service class periods can be selected as CPU donors. As a consequence, the policy adjustment routine changes their relative dispatching priorities. Although the performance index values are sampled every 250ms, the resource adjustment cycle only runs once every 10 seconds. So with a sudden influx of transactions, different importance classes coming in, or CPU constraint for any other reason, then the performance indexes must suffer before WLM will take action.

For details, see *OS/390 Workload Manager Implementation and Exploitation*, SG24-5326.

Common problems are as follows:

- ▶ z/OS systems are run at very high CPU utilization for long periods of time, which can lead to widely varying application or transaction response times.

Benchmarks run with the system operating between 92-95% of the CPU capacity have recorded some of the best throughput (not 100% CPU utilization).

- ▶ There is less and less “discretionary” or unimportant work running in the z/OS environment that can be preempted to allow DB2 transactions or other workload to take precedence.
- ▶ Sporadic slowdowns or system “hangs” can materialize across a DB2 member, an entire LPAR, or an entire DB2 data sharing group spanning several LPARs due to WLM settings.

DB2 applications are allowed to be preempted while holding locks or internal DB2 latches, thus affecting other workload in the group. This latching increase will not be linear in nature as the CPU utilization increases.

DB2 system address spaces may not be protected from being preempted by application workloads that are dependent on the DB2 address spaces being dispatchable (deadly embrace).

We cover some rules of thumb for classifying DB2 and dependent workloads, as well as scenarios that have occurred in customer environments.

3.2 The DB2 subsystem

First and foremost, the WLM policy needs to ensure the protection and importance of the DB2 address spaces. When we speak of the DB2 address spaces, we are including MSTR, DBM1, DIST, and the WLM managed stored procedure address spaces (APPLENVs).

As a general rule, the IRLM is not included in this grouping because it should always run at SYSSTC priority (importance 255). The three main address spaces (MSTR, DBM1, and DIST) should also run as their own service class to help ensure that WLM can manage them at a more granular level.

Grouping them with other started tasks such as IMS regions is acceptable, given that the workload is consistent across all the address spaces. Grouping DB2 with WebSphere, for instance, in 1 service class, may make it difficult to continuously attain a specific velocity goal due to the differing nature of workload volumes.

Attention: If you do not have an entry in your WLM policy for SUBSYS=DDF with at least a generic “catch-all” service class to broadly apply to DDF workload, then all of those transactions will be running as Discretionary. As a general rule, we discourage any workload running against DB2 to be able to fall into a discretionary service class.

Tip: We suggest that MSTR, DBM1, DIST, and the WLM application environments for DB2 storage procedures all run as importance 1 with a high velocity goal. “High” means above any other work that would be accessing DB2, but within the bounds of Table 3-2.

The reason they should be so high in priority is that the address spaces themselves are not CPU or resource intensive. They need to quickly gain access to CPU cycles to allow work to be processed, but only for short durations of time. For instance, a remote connection comes in and the DIST address space needs to schedule a DBAT in enclave SRB mode in order for it to submit its SQL to DBM1. Once established in its enclave, the actual SQL will run at the importance of the enclave, so it is not the high importance DB2 address spaces that will be consuming the majority of the CPU.

Other examples of the address space needing CPU include cross-system P-lock negotiation for space map pages, which could affect tens of applications, or the LPVT latch for storage allocation, which could freeze all the threads within DB2, or the log write latch (LC19), which if preempted could stop any threads from updating objects in DB2.

Table 3-2 Max velocity goals

Number of CPs on LPAR	Suggested max velocity goal
1-5	50-70
6-15	60-80
more than 15	70-90

In the DB2 9 documentation, it was suggested to run MSTR in the service class SYSSTC. This was due to the fact that the virtual storage monitor (DSNVMON) daemon for DBM1 ran under the MSTR address space, and you generally want monitors running at a higher dispatching priority than what it is monitoring. This would not be necessary if the LPAR is less than 95% busy most of the time and blocked workload support is in effect.

3.2.1 z/OS factors helping DB2

Many customers operating at very high system utilization rates (>95%) encountered subsystem or even sysplex wide slow-downs for sometimes unexplained reasons. For several cases that were driven to root cause, promoting either the DB2 address spaces to SYSSTC, marking them CPU Critical, or promoting individual low priority tasks, resolved the issue.

These low importance tasks were taking control of internal DB2 latches and then being preempted by more important work, which in turn needed that same latch. Most of the time, these units of work were running in a Discretionary WLM service class. By promoting them to a higher service class, they were able to be dispatched and release the lock or latch they had been holding. A new z/OS enhancement came in V1.10 (retrofitted back to z/OS 1.9), called blocked workload support, was an effort to automate this promotion of low priority work.

There are two settings incorporated into this support.

- ▶ BLWLTRPCT is the allowable % of CPU capacity on the LPAR used to promote low importance work in order to get it out of the way.
- ▶ BLWLINTHD is the amount of time the unit of work must be preempted in order to qualify for blocked workload support.

Example 3-3 shows an RMF Report with blocked workload.

Example 3-3 Blocked workload from RMF CPU report

-BLOCKED WORKLOAD ANALYSIS									
OPT PARAMETERS:	BLWLTRPCT (%)	0.5	PROMOTE RATE:	DEFINED	76	WAITERS FOR PROMOTE:	AVG	0.000	
	BLWLINTHD	20		USED (%)	0		PEAK	0	

Tip: If the USED (%) is a non-zero number during times of peak workload, then the BLWLTRPCT should be increased to 1% and BLWLINTHD should be decreased to 5 seconds in order to more aggressively promote work out of the system and allow higher importance work in. There is a techdoc on blocked workload available here as well:

<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10609>

DB2 has its own internal version of this ability to promote tasks. Threads that have been suspended for more than a certain number of seconds can be boosted in priority via a WLM service. See Example 3-4 on page 36.

3.2.2 DB2 subsystem consequences

The DB2 address spaces should always be at a higher Importance level than the work coming in. In one customer example, the DDF work entering the system was running in a WLM service class at Importance 1, while the DIST address space was running at Importance 2 with a velocity goal of 60. During an interval of abnormally high DDF activity, this importance 1 workload began to preempt the DB2 address space. When the DDF work took the available CPU resources and the DIST address space was preempted, there were delays in scheduling the DBATs for this remote work to run on. Hence the DB2 address space shows a performance index of almost 14 (receiving 1/14th of the resources it is requesting), while the DDF_HIGH workload also starts to miss its WLM goal (yellow bar is PI > 1). Since the DDF work is dependent on the DIST address space, both service classes suffer and we have a “deadly embrace” scenario. See Figure 3-1.

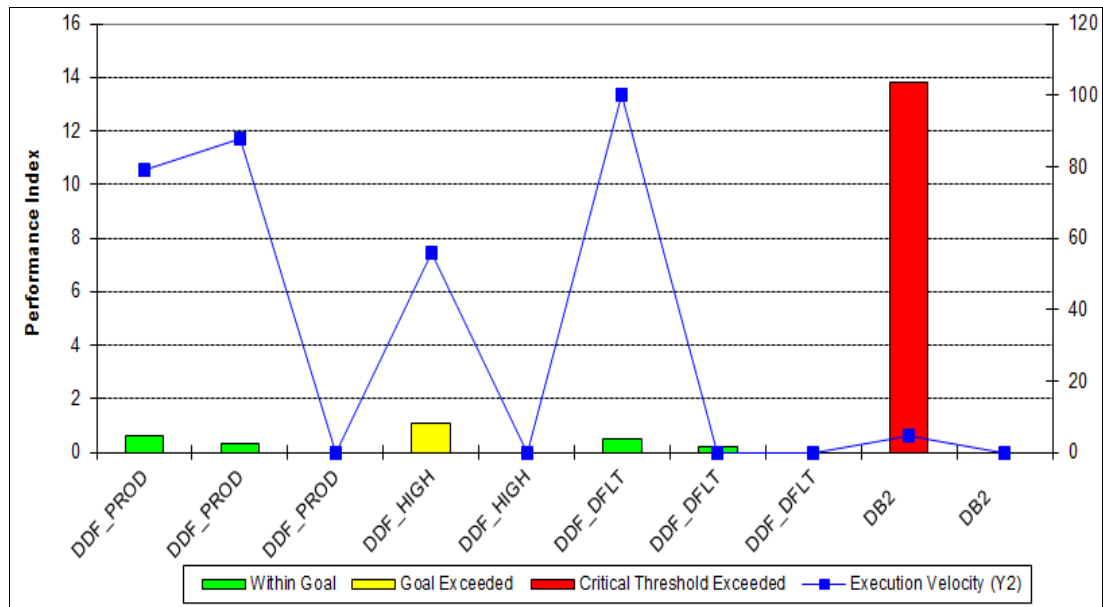


Figure 3-1 DDF higher than DB2

This situation would not have occurred if the DB2 address spaces were all at a higher importance level than the work they were servicing. The response time degradations were not noticed until this particular afternoon where the CPU resources were stressed and the WLM activity report showed that the service levels were being met prior to the flood of DDF work coming in and preempting DB2 DIST. Sometimes this can occur even with work that is of lesser importance than the DB2 address space if there are violent swings in workload activity. The WLM setting CPU Critical is meant to stem this so that no work with a lower importance than the CPU Critical work can preempt it. Overuse of this setting can have a negative effect, however, since there is no longer any workload to preempt when CPU is extremely constrained.

3.3 Thread level

When investigating a performance degradation or monitoring to set a baseline, it is important to understand what workload is associated with what WLM service class.

3.3.1 In DB2

DB2 10 added several new fields in the **DISPLAY THREAD DETAIL** command, in line DSNV482. See Figure 3-2.

The fields include the WLM service class that the thread is associate with, what period of the service class it is currently in, the relative WLM importance, and finally the performance index (multiplied by 100). So in this case, the thread is running under the DDFSVC service class in its first period with an importance of 5 and is exactly achieving its workload goal because the PI is at 100 (PI=1). To determine the WLM service class definition for DDFSVC, you would still have to go back to the actual policy.

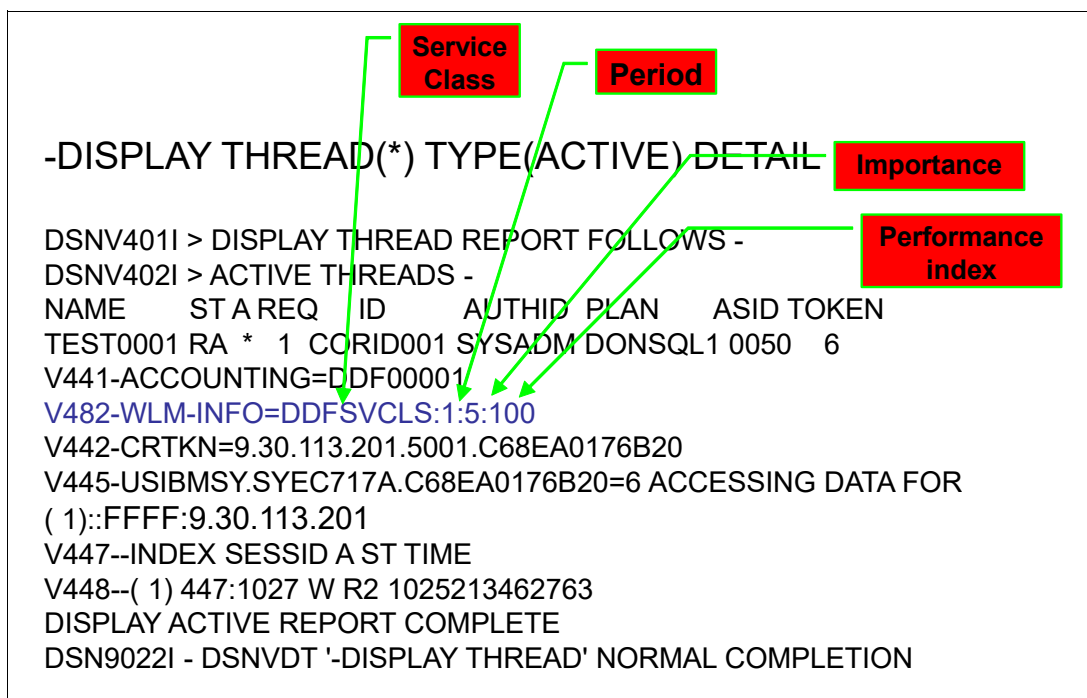


Figure 3-2 WLM information in -DIS THREAD

This simple display can be quite helpful during online browsing of the distributed threads. For instance, you could tell by scrolling through if anything fell into the SYSOTHER category due to a new DDF application being put in that was not properly categorized in the WLM policy. Or if 1 application is having response time issues, is the goal actually being missed, or are the threads running within their described SLA. If you captured the output, you might also be able to search through and determine if many of the transactions complete in the first of a multiple period service class, or if they tend to fall into the later periods (which could also be summarized by interval in the WLM activity report).

The display command **-DIS THREAD(*) SERVICE (WAIT)** (see Example 3-4) was enhanced and automated in DB2 9. This command was available in DB2 V8, and it would identify allied agents and DBATs that had been suspended for more than so many seconds. The time was determined by the maximum of 60 seconds, or 2 times the IRLM resource timeout interval. In DB2 9 and up, this command is executed by DB2 itself once a minute.

Example 3-4 -DIS THREAD() SERVICE (WAIT)*

```

DSNV401I -DT45 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I -DT45 ACTIVE THREADS -
NAME      ST A REQ  ID      AUTHID PLAN      ASID TOKEN
CT45A     T * 12635 ENTRU1030014 ADMF010 REPN603 00C6 90847
V490-SUSPENDED 04061-19:20:04.62 DSNTLSUS +00000546 14.21

```

The built-in monitor will run from startup to shutdown and will check the health of the system in 1-minute intervals. It will identify CPU stalls (for system, DBAT, and Allied agents) that result in latch contention. The monitor will also attempt to clear the latch contention via a temporary priority boost via WLM services to the latch holder. This command can be run with SCOPE(GROUP) to get the overall picture, and the output will be piped back to the originating member via a notify message.

Tip: For proactive monitoring, it would be useful to manually issue the command, **-DIS THREAD (*) SREVICE (WAIT)**, at regular intervals through automation so that you have a written record of the threads being boosted and the common offenders.

The exception in the suspended threads list are DBATs that have been pooled, so you would want to go back and ensure that the token corresponds to an active thread. If your POOLINAC ZPARM value is less than 60 seconds or 2 times your IRLM resource timeout, then the false positives would be removed.

Related information in the DSNV507 message involves the keyword, **BOOSTS**, to let you know the number of tasks that have been boosted using the WLM service since the internal monitor came up. See Example 3-5.

Example 3-5 -DIS THREAD() TYPE(SYSTEM) cont.*

```
V507-ACTIVE MONITOR, INTERVALS=1235, STG=47%, BOOSTS=0, HEALTH=100  
REGION=1633M, AVAIL=1521M, CUSHION=375M
```

3.3.2 Regarding WLM resource group capping

As mentioned previously, the use of resource group capping for workloads entering DB2 should be avoided. Resource groups can be utilized to set a lower or upper limit on the amount of resources a WLM service class can receive.

Many customers have used this tactic against discretionary (Imp 5) workloads to ensure they get a minimum amount of CPU in order to keep the work progressing. If the associated service class has a discretionary goal, WLM achieves the minimum as long as service goals running in any other defined SC are not impacted. But, if other service goals are impacted, then WLM does not maintain the minimum. So in times of high CPU utilization, when you really need the minimum to ensure that work does not stall, the task can be suspended.

When using resource group capping to limit the resources consumed, you can base the limit on the number of CPU service units consumed per second, or as a percentage of the processing cycles available to the LPAR. So as an example, the policy could dictate that the service class receives only 500 service units per second, or only a max of 1% of the available LPAR capacity. This can have a very negative impact on DB2 if the service class hits the cap and is suspended while it is holding an important lock or latch. It and other workloads would be impacted. Even with the best of intentions, if this workload grows over time or there is an increase during end-of-month processing for example, the cap could be enforced at unintended times.

The other thing to consider with capping versus a very low service class goal is that with capping, the Blocked Workload Support (Example 3-3 on page 34) cannot aid that service class. Capping essentially side-steps the z/OS fail safe to ensure that work does not get stalled.

	RESPONSE TIME EX			PERF	AVG	--EXEC USING%--				----- EXEC DELAYS %					
SYSTEM	HHH.MM.SS.TTT	VEL%	INDX	ADRSP	CPU	AAP	IIP	I/O	TOT	CPU	CAP	IIP	Q	I/O	MPL
*ALL	000.00.00.027	15.4	0.0	10.6	4.9	N/A	1.0	0.4	35	22	11	1.3	0.1	0.1	
1E10	000.00.00.015	25.3	0.0	4.0	9.3	N/A	2.6	0.1	35	17	15	3.4	0.0	0.0	
2D11	000.00.00.169	7.6	0.0	6.6	2.2	N/A	N/A	0.7	34	25	9.0	0.0	0.2	0.1	

Figure 3-4 shows an example of a user's WLM activity report. The WLM goal is an average response time of 10 seconds. When we look at the actual response time, we see that they are completing in an average of 0.004 seconds (4ms). The response times are further broken down into buckets, which lump the transaction response times into time intervals to correlate to the performance index.

So in the lowest bucket, there are those transactions that completed in less than 5 seconds, corresponding to a performance index of 0.5. There are enough buckets to ensure that all of the transactions for that service class period are accounted for in that interval. In this example, 100% of the 80,677 transactions complete in this first bucket, which is no wonder with an average response time of 0.004 seconds. The performance index reported for this interval is 0.5, which is a gross understatement of how well this service class actually performed. A performance index of 0.01 is more accurate, but WLM will not take into account anything less than 0.5 for a performance index.

GOAL: RESPONSE TIME 000.00.10.000 AVG														
	RESPONSE TIME EX				PERF	AVG	--EXEC USING%--				-----EXEC			
SYSTEM	HHH.MM.SS.TTT	VEL%	INDX	ADRSP	CPU	AAP	IIP	I/O	TOT	CPU	IIP	CAP		
ABC	000.00.00.004	29.3	0.0	0.6	15	N/A	2.5	0.0	43	33	5.9	4.5		
-----RESPONSE TIME DISTRIBUTION-----														
----TIME----			--NUMBER OF TRANSACTIONS--			-----PERCENT-----			0					
HH.MM.SS.TTT			CUM TOTAL		IN BUCKET	CUM TOTAL		IN BUCKET					
< 00.00.05.000			80677		80677	100		100	>>>>					
<= 00.00.06.000			80677		0	100		0.0	>					
<= 00.00.07.000			80677		0	100		0.0	>					
<= 00.00.08.000			80677		0	100		0.0	>					

Figure 3-4 WLM activity report showing response times

This is certainly an example of our rule of more than 90% of the transactions completing with a PI of 0.5. This means the response time goal should be tightened down considerably. In times of CPU stress, we cannot expect the 0.004 response times to continue, but the customer needs to determine the maximum actual time that is acceptable. If the user saw a hundred fold increase in response time, they would certainly notice a degradation. Perhaps a goal of 0.015 (the Workload Manager minimum response time goal) is best, as it puts the current response time at just below a 0.5 performance index.

Sometimes the goals are too strict

Percentile response time goals are very common in IMS and CICS, as well as online distributed transactions. This is because, given a specific application, 80-90% of the transactions complete in the same amount of time without any interference. Getting the exact percentile correct can be very difficult, and once again is best done by monitoring the WLM activity reports to see what the average time and deviation is. Based on the buckets in the activity report, or graphically in the RMF Spreadsheet Reporter, look at the percentage of transactions that complete within the ideal goal and determine if the service class period goal is mathematically attainable.

Figure 3-5 shows the performance index of CICS transactions in a customer environment. This goal of 95% completing in 0.3 seconds is never achieved during the intervals noted. The performance index of 5 does not mean that the average transaction time was 1.5 seconds ($5 \times 0.3 = 1.5$), but that to get to 95% of the transactions for this interval, we needed to add up the transactions in all of the buckets up to the 1.5 second bucket. The actual performance index is not as relevant as why it is being missed.

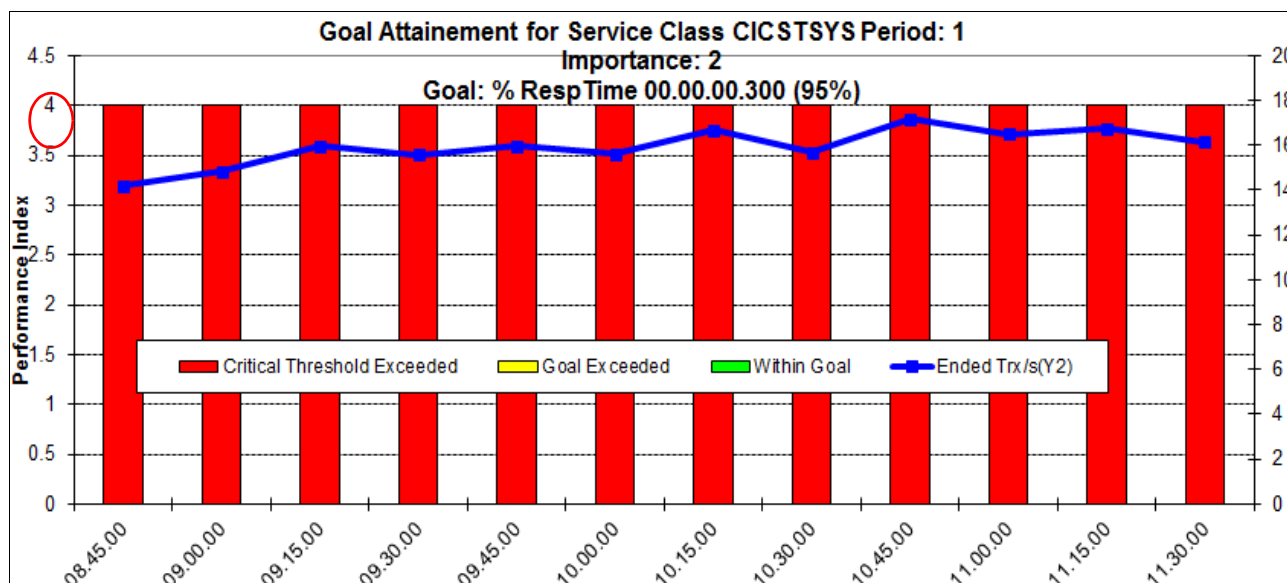


Figure 3-5 Poor performance index for CICS transaction

We can look at the response time distribution in Figure 3-6. The RMF Spreadsheet Reporter Graph shows us that only 90% of the transactions completed in the 0.15 second bucket. The other 5% of transactions needed for the 95% goal are over in the 1.2 second bucket.

So regardless of the CPU constraint, or abundance, on the system, will these 5% of transactions ever be able to complete in the goal of 0.3 seconds? The answer would be no. So here is an example where the goal is too aggressive. The options for the customer are to lower the percentile goal down to 90%, or move those 5% of transactions to their own WLM service class, which can be made attainable for transactions that run in 1.2 seconds.

The downside of leaving this service class in its current state is that the entire service class misses its goal every interval due to a small percent of the total transactions. So what? Well in circumstances where several classes are missing their goals and WLM deems that this service class cannot be helped with the resources available, that workload will be ignored for 3 resource adjustment cycles. This means that for 30 seconds, WLM will do nothing to help our CICS transactions mentioned before. By simply removing the 5% of slower transactions, this workload will exceed its goal every interval, and then be taken care of by WLM during times of CPU constraint.

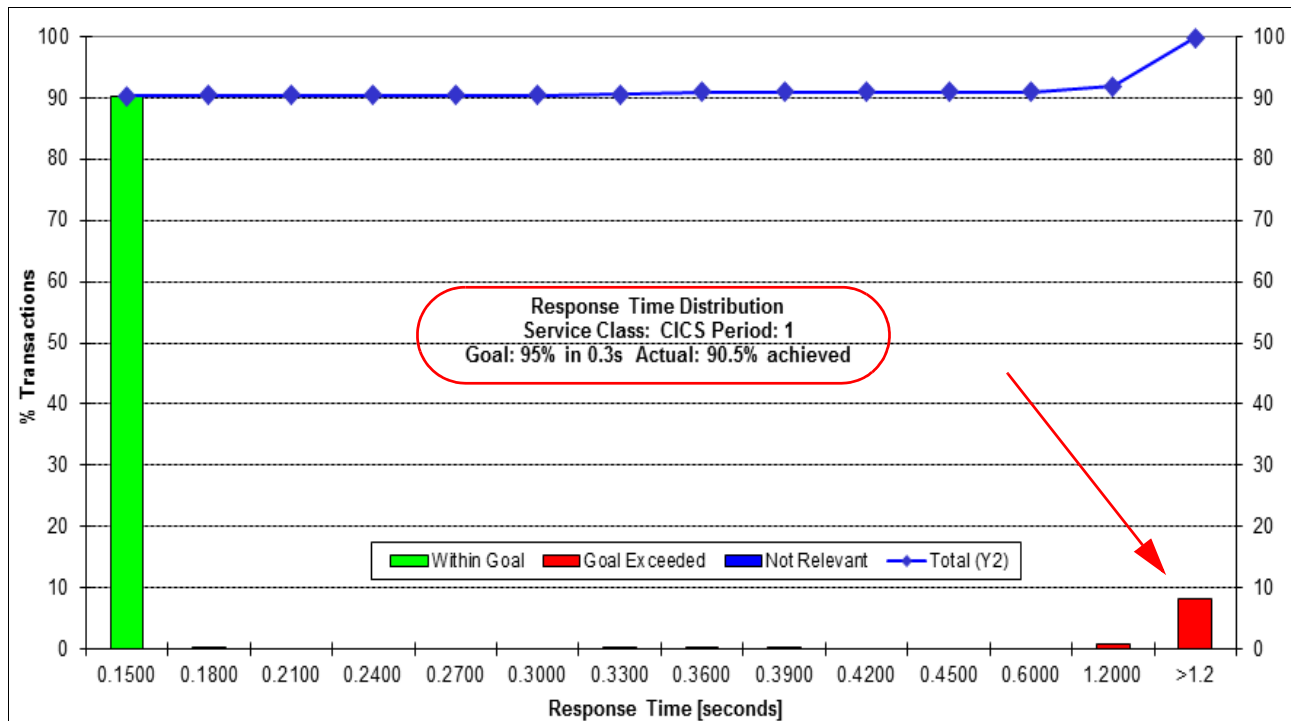


Figure 3-6 CICS response time variance

There are some special cases where tightening the WLM goal for CICS transactions coming from an AOR can have a negative affect on CICS TOR work. When internal CICS queue contention increases on such systems and the CICS TORs and AORs are being managed to the goals of the transactions, WLM currently has no way to give the work managers, the CICS TORs, faster access to resources than the work consumers, the CICS AORs. At higher utilization levels, typically above 85%, a noticeable queue delay within the CICS TORs might be recognized. This increases end-to-end response time of the CICS transactions and reduces throughput of the CICS work. These symptoms become more visible in a HIPERDISPATCH=YES environment. This issue can be addressed with z/OS APAR OA35428, and the new option to have the WLM service class set to 'manage region BOTH'.

<http://www.ibm.com/support/docview.wss?uid=isg10A35428>

3.4 General WLM best practices

Overall the technical team needs to ensure that the business goals and service level agreements align with the service class goals implemented in the WLM policy. The WLM policy must be set up with a worst case scenario of CPU constraint in mind. When the LPAR or CEC is less than 75% utilized, the WLM policy may never come into effect, and hence never be tested to flush out competition for CPU resources.

There are items that can be easily determined from the WLM service definitions that raise performance concerns. A great way to analyze the WLM policy is the WLM Service Definition Formatter from the z/OS website. If you FTP the WLM policy to yourself in .txt format and import it into the Formatter, you are first presented with the overview page shown in Figure 3-7. Here you can see if there are multiple policies, the number of workloads, periods, resource groups, and many other high level stats. We have discussed the use of resource groups already, but what about service class periods?

Service Definition				
Definition:	Q121004	Adjust SC Defs	Workloads	7
Policies:	ASYS1		Service Class Periods	96
			Resource Groups	18
			Service Policies	5
			Classification Groups	158
			Subsystem Types (used)	7
			Report Classes	193
			Application Environments	132
			Scheduling Environments	5
Coefficients			I/O Priority Management	Yes
		1	Alias Management (PAV)	Yes
		1		
		0.1		
		0		

Figure 3-7 WLM service policy overview from RMF Spreadsheet reporter

Tip: There is a long standing rule of thumb that says you should only run 25-35 concurrent service class periods on an LPAR. Notice that we refer to periods and not service classes, so if you have many service classes with multiple periods, this number can grow rapidly:

http://www-03.ibm.com/systems/z/os/zos/features/wlm/WLM_FAQ.html

The reasoning for this is not so much the overhead of managing hundreds of concurrent service classes, or even sampling all of them to determine the performance index. The issue is that during a resource adjustment cycle WLM must be able to make it through the entire list of service class periods, determine who the donor/recipients should be, and reallocate the resources. In this case WLM may not be able to get through the list of service classes needing resources, and they will need to wait another 10 seconds for the chance to be evaluated. This occurs at the LPAR level, and at the sysplex level. The sysplex performance index must be evaluated for each of these service class periods as well, however, this PI is not scrutinized as much by WLM as it was in the past.

A simple way to reduce a high number of service classes is to try and consolidate redundant ones. In the customer example in Figure 3-8, we have 9 importance 1 service classes with a velocity goal of 60. Seven of these are variations on batch processing and are doing very similar work. By consolidating duplicates such as these into less individual service classes, WLM can more accurately interpret the samples from these workloads, as well as the goal.

There are also predictive heuristics in WLM that can be clouded by many redundant service classes. It is much more efficient to break these batch processes out into separate reporting classes, and monitor them that way to see if they are individually making their goal. If being combined with other workloads causes performance to suffer, that is the point when a new service class should be created, not just because there is a new batch process or application coming in.

Service Policy									
Policy	Workload	Service Class	Per	Dur	Importance	Type	Goal		
							Pct		
ASYS1	W_BATCH	BAT_SPCL	1			1 ExVel	60		
ASYS1	W_BATCH	WBATSPCL	1			1 ExVel	60		
ASYS1	W_BATCH	WODBASPC	1			1 ExVel	60		
ASYS1	W_BATCH	WODBBSPC	1			1 ExVel	60		
ASYS1	W_BATCH	WODBCSPC	1			1 ExVel	60		
ASYS1	W_BATCH	WODBOSPC	1			1 ExVel	60		
ASYS1	W_BATCH	WODBVSPC	1			1 ExVel	60		
ASYS1	W_BATCH	WODBZSPC	1			1 ExVel	60		
ASYS1	W_STC	STC1	1			1 ExVel	60		

Figure 3-8 WLM service policy filtered by service class importance

If you are looking to swap some velocity or discretionary goals over to response time goals, z/OS 1.13 can help, as WLM reporting provides a response time distribution for all service classes. For example, an RMF user can retrieve the response time for service class periods with velocity and discretionary goals. With this information, users can, for example, migrate service class with a velocity goal to one based on a specific response time.

In Figure 3-9, the average response time is plotted as the secondary axis, with the performance index of the current velocity goal as the primary axis. Monitoring these actual response times would give you the data points you need to create a response time based goal to better serve the end user, and to be prepared for CPU constraint. This will keep the end user experience more consistent, as WLM will be adjusting resources for the service class based on response time.

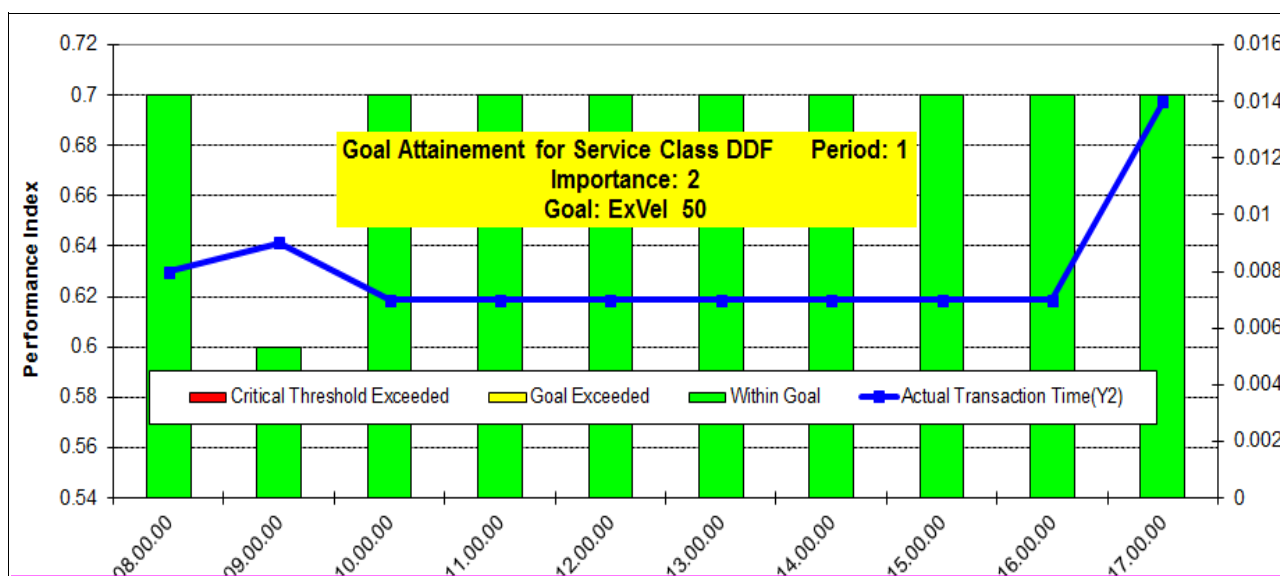


Figure 3-9 Average response time shown for service class that runs under a velocity goal

3.5 Looking at RMF data

The Resource Measurement Facility (RMF) is an optional z/OS product shipped with every release of z/OS. RMF gathers system performance data and can write it out to SMF data sets or an SMF buffer. Items at the system level include CPU utilization, WLM service class attainment (across a sysplex), coupling facility utilization, DASD response time, device utilization, and so on. Some excellent instructions on using the various monitors in IBM *zSeries®*, as well as the RMF Spreadsheet reporter, can be found in the Redbooks publication, *Effective zSeries Performance Monitoring Using Resource Measurement Facility*, SG24-6645. It is important to understand which RMF records house the data from which processes. Generally, during any performance investigation done by IBM level 2, we will ask for the SMF 70-79 records.

Here is a list of SMF records and subtypes that are typically useful for generating RMF reports. Also included is an example of the RMF report control statements that can be used to populate the charts in the RMF spreadsheet reporter. In the examples, if the report says REPORTS, then the data can be gathered from a single LPAR on the CEC, but if it says SYSRPTS, then you will need data from each LPAR in the sysplex for a complete view:

- ▶ 70-1: % CPU, zIIP busy, weightings, number of MSU's consumed, WLM capping, physical and logical busy.

Here is an example of the RMF report command to be used when populating the *LPAR Trend Report*:

```
REPORTS(CPU)
```

Each LPAR you pull data from in this report will have data from all the LPARs on the CEC, so you will have redundant data if you pull from each one.

- ▶ 70-2: crypto HW busy
- ▶ 71: Paging activity:

This is useful in determining if there is an overall shortage of storage on the LPAR (paging more than 10 times per second), but it is more interesting to understand which address space is being affected. We will look at that in Table 13-5 on page 183.

- ▶ 72-3: Workload activity:

Here is an example of the RMF report command to be used when populating the *Workload Activity Trend Report*:

```
SYSRPTS(WLMGL(POLICY,WGROUP,SCLASS,SCPER,RCLASS,RCPER,SYSNAM(SWCN)))
```

Notice that you can specify the system name, but generally you would want the sysplex wide view of the service class, and hence need to collect the RMF data from each LPAR. At the sysplex view, you can still drill in to see which LPARs the work is running on, and if they are making their WLM goal on each one individually.

- ▶ 73: Channel path activity
- ▶ 74-1: Device activity

Here is an example of the RMF report command to be used when populating the *DASD Activity Report*:

```
REPORTS(DEVICE (DASD))
```

► 74-4: Coupling facility:

Here is an example of the RMF report command to be used when populating the *Coupling Facility Trend Report*:

```
SYSRPTS(CF)
```

You will need to gather RMF records from each LPAR that belongs to this sysplex in order to get the complete picture. Each one can report the CF CPU utilization, but you may want to know which LPAR has delays due to channel unavailability.

► Other reports:

- 74-5: Cache activity
- 74-8: Disc systems
- 75: Page data sets
- 78-2: Virtual storage
- 78-3: I/O queueing

Putting these data points and reports into a digestible format is a key component to be able to trend data points and compare relevant metrics, such as CPU utilization and WLM service classes missing their goals.

Tips:

- Here is a link to the Information Center regarding the RMF Spreadsheet reporter:
<http://publib.boulder.ibm.com/infocenter/zos/v1r11/topic/com.ibm.zos.r11.erb200/erbzug91105.htm>
- Here is a link to a Redbooks publication with a good description of the setup of the Spreadsheet Reporter:
<http://www.redbooks.ibm.com/abstracts/sg246645.html>
- Here is a link to the RMF spreadsheet reporter itself with the panel in Figure 3-10:
<http://www-03.ibm.com/systems/z/os/zos/features/rmf/tools/>

Tools

Overview

Product Information

Newsletters

Resources

Library

Tools

Presentation

Contact z/OS RMF

Click [here](#) to contact RMF Development.

On this page the RMF development group provides a number of tools to complement the RMF product. If you have trouble downloading the tools directly from this page, follow the instructions to do a [direct FTP download](#).

Page last updated: September 01, 2011

- ↓ General download and installation instructions
- ↓ RMF Postprocessor XML Toolkit Version 1 for Windows
- ↓ RMF Spreadsheet Reporter Version 5 for Windows
- ↓ RMF PM Java™ Technology Edition to monitor z/OS sysplexes
- ↓ RMF PM Java Technology Edition with additional support to monitor Linux® enterprise servers

Figure 3-10 Download page for the RMF spreadsheet reporter

The RMF Spreadsheet Reporter is a powerful workstation solution for graphical presentation of RMF Postprocessor data. Use it to convert your RMF data to spreadsheet format and generate representative charts for all performance relevant areas. With this Spreadsheet Reporter version, you can create graphical charts from raw SMF data in one single step. You can directly start the RMF provided spreadsheet macros from the Spreadsheet Reporter main window. One impressive aspect of this tool is that it is a no-charge downloadable tool that you can report problems on and get email based support from the development team.

As an example of the Spreadsheet Report's use, Figure 3-11 shows RMF Monitor I data being stored in SMF data sets. Moving up the left side of the chart, you could produce batch RMF reports from these SMF data sets, as we have seen previously in this chapter at Figure 3-4 on page 39. After the RMF Postprocessor creates these historical reports, they can be fed into the RMF Spreadsheet Reporter to create easily interpreted, customizable, and color coded graphs. You may want to look at the response time goals achieved for a specific RMF interval, or trend the CPU consumption on the CEC over the entire day. The source data for the graphs will be displayed below the graph in some cases, or in another sheet of the Spreadsheet, so you can add or delete rows and columns, change the secondary access, and overlay different charts if you so desire.

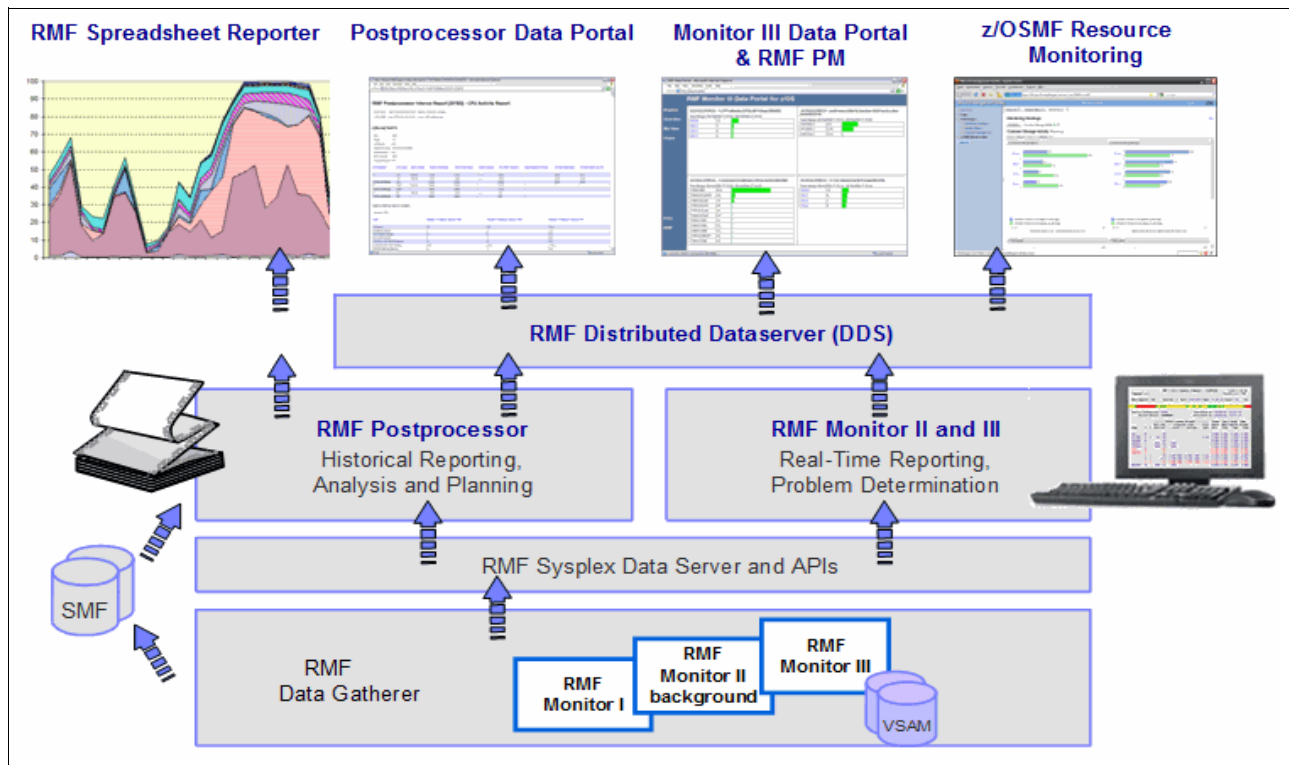


Figure 3-11 RMF components, focusing on left side of the chart for the Spreadsheet reporter

Those of you who are familiar with the standard RMF WLM activity report or CPU activity report may look at the Spreadsheet interpretation and want to go back to the source data. This can be found in the data sheets within the spreadsheet, or you could reference the original reports themselves.

Tips: Here are tips for using the tool AFTER the basic setup has been accomplished.

See the *System Programmer's Guide to: Workload Manager*, SG24-6472-03 and *Effective zSeries Performance Monitoring Using Resource Measurement Facility*, SG24-6465, and use Figure 3-12 on page 48 as a reference.

- The biggest time saver is to run the reports (such as WLM activity, CPU activity, and others) on IBM MVS™ and simply FTP the output to yourself, rather than pointing the tool to an SMF DUMP data set on the MVS system.

This way, you can more easily monitor it in SDSF, and you are not waiting for the Spreadsheet Reporter to FTP the report down in the background, which can be very time intensive. This also allows you to select all the options in the report listing options without each report being run against the RMF data when you just want a working set for a specific RMF report. Also, you can save the report (with a .lis file name extension) locally so that you can reference it, and simply point the tool to the .lis file when you create the working set.

The one set of reports that you should run through the tool on your desktop is to generate the overview records. The JCL for overview records is very long and tedious, but there is a sheet within each of the RMF spreadsheets to be able to have the tool generate the overview record JCL for you. You then put the path for this file in your "Systems" settings and the tool will apply it for you.

- In the settings → options panel, choose "Ignore specified duration period and time interval" so the tool does not filter any more of your data.

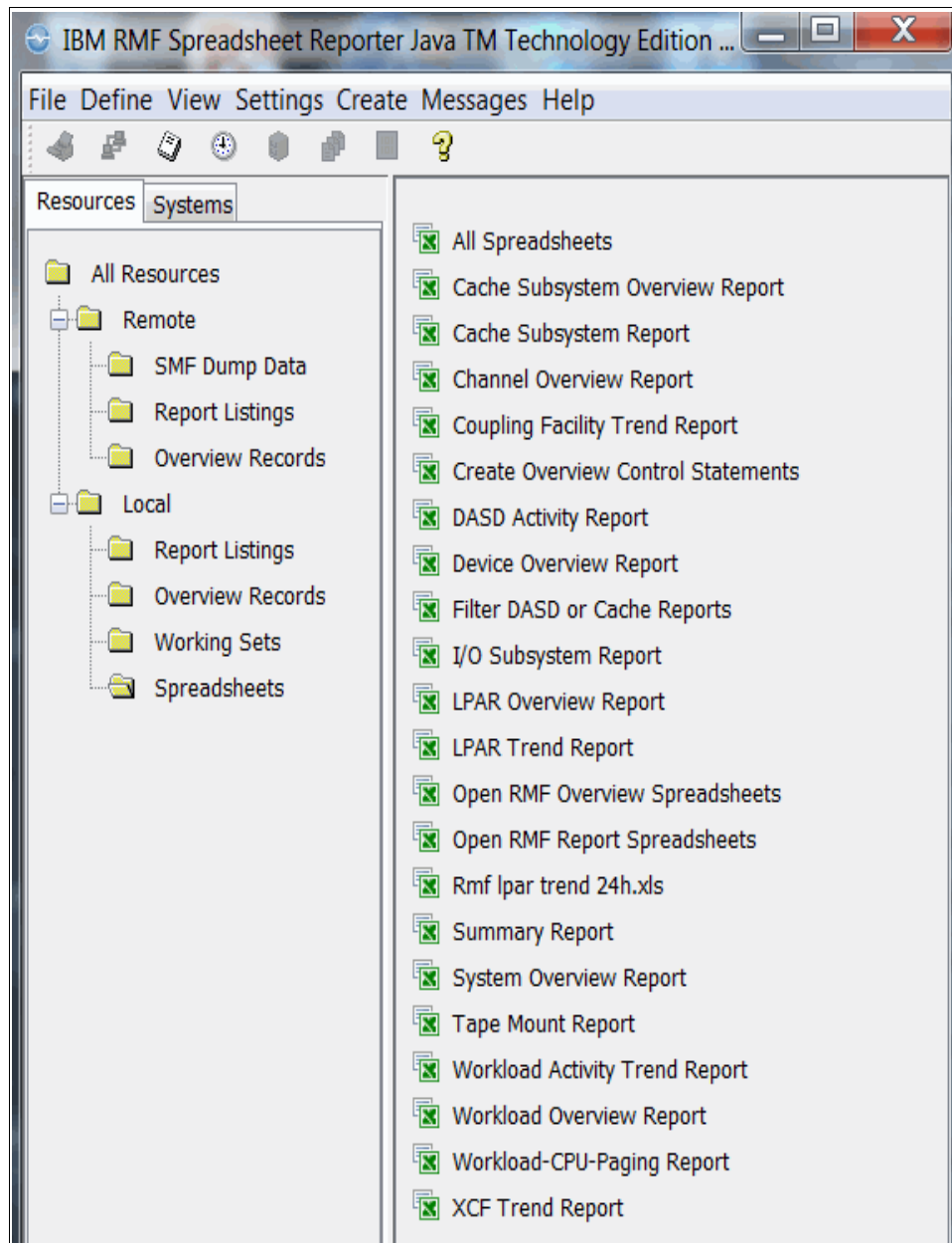


Figure 3-12 RMF Spreadsheet Reporter main panel

3.5.1 Overview records

The overview (OVW) option offers you the capability of tailoring summary-like reports according to your requirements. You can create your own single system and sysplex reports that show exactly the information you need for your performance management tasks. In the same way that you can create Overview reports, you can also create Overview records, just by specifying an additional option called OVERVIEW(RECORD).

Overview spreadsheets can be very useful for summarizing data over hours, days, or weeks, which would otherwise be impossible to do simply running the report. You would need a performance database to query such vast amounts of data.

Suppose that you want to see, over the course of a day, what workloads are consuming the most CPU on the LPAR or CEC. You would need to run a CPU activity report and graph it out, then run a WLM activity report, run it through the tool, and then go through each workload to see the APPL% of a CP the workload is consuming, and correlate that to the percentage utilization of the LPAR. With the WLM Overview report, you can choose which WLM service class periods, report classes, or workloads you want to see stacked during each interval. This way, maybe you can see that an Importance 1 batch job kicked in at 12 AM and consumed over 40% of the LPAR at 12:25AM (Figure 3-13). You would be able to see the other workloads consuming CPU, and if the box was running over 90%, for instance, you would probably see the less important workloads being squeezed out of CPU cycles.

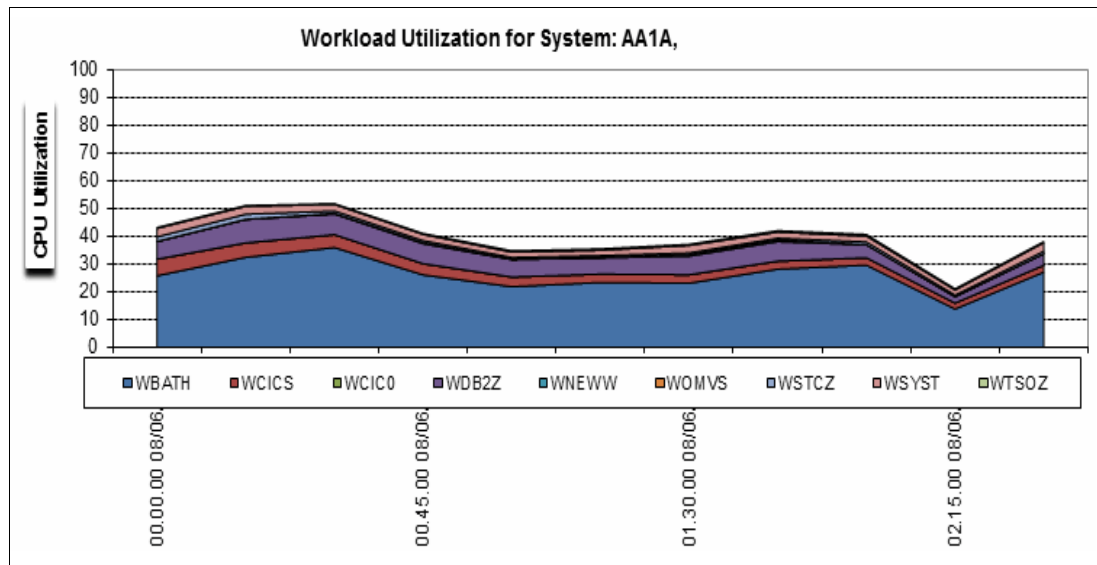


Figure 3-13 WLM overview report

Another key ingredient to DB2 subsystem and application performance is DASD response time. Despite System z's I/O assist processors and IBM FICON® EXPRESS innovations, a saturated device or DASD volume can impact our response times. The individual DASD Activity report (see Figure 3-14) can show the top 5 devices (control units) as well as the top 10 DASD volumes relative to response time and activity. The breakdown of the response time could tell you if you are in need of more UCBs for Parallel Access Volume support based on the IOSQ time, for instance.

LCU Summary										
	LCU	I/O Intens.	ST Intens.	Path Int.	Act. Rt.	Resp. Tm	Serv. Tm	IOSQ Tm	Pend. Tm	Disc. Tm
Top 5	0035	85827.90	923.02	814.34	146.46	585.63	6.30	579.00	0.33	0.74
	0036	4210.37	1464.61	1401.45	300.74	13.99	4.87	8.81	0.31	0.21
	0030	2933.42	1893.53	1090.33	388.02	7.56	4.88	2.38	0.30	2.07
	002F	620.17	431.46	174.53	442.98	1.40	0.97	0.14	0.29	0.58
	0034	326.09	236.88	71.78	51.27	6.36	4.62	1.39	0.35	3.22
sorted by I/O Intensity										
Device Summary Top 10										
LCU	VolSer	I/O Intens.	ST Intens.	Path Int.	Act. Rt.	Resp. Tm	Serv. Tm	IOSQ Tm	Pend. Tm	Disc. Tm
0035	DBS001	84992.57	600.32	516.51	91.10	932.93	6.59	926.00	0.34	0.92
0036	DB2WC1	1842.65	71.56	68.10	16.02	114.80	4.47	110.00	0.33	0.22
0036	WORK23	734.76	393.63	393.14	27.11	27.09	14.52	12.20	0.37	0.02

Figure 3-14 DASD Activity Report

This is great to measure response times over a specific interval, but what if you do not know the actual interval that the incident occurred, or you are not sure what a typical response time would be for these busy volumes and you need to look for outlying situations? Then the RMF Spreadsheet reporter Device Overview spreadsheet could be very useful. At 8 PM, the SPS39 volume (Figure 3-15) goes from a response time of about 1 ms to almost 7 ms during this interval. If applications accessing that volume experienced a relatively sudden response time increase, which could be verified in the accounting report, then we need to get the DASD team involved in our problem determination process.

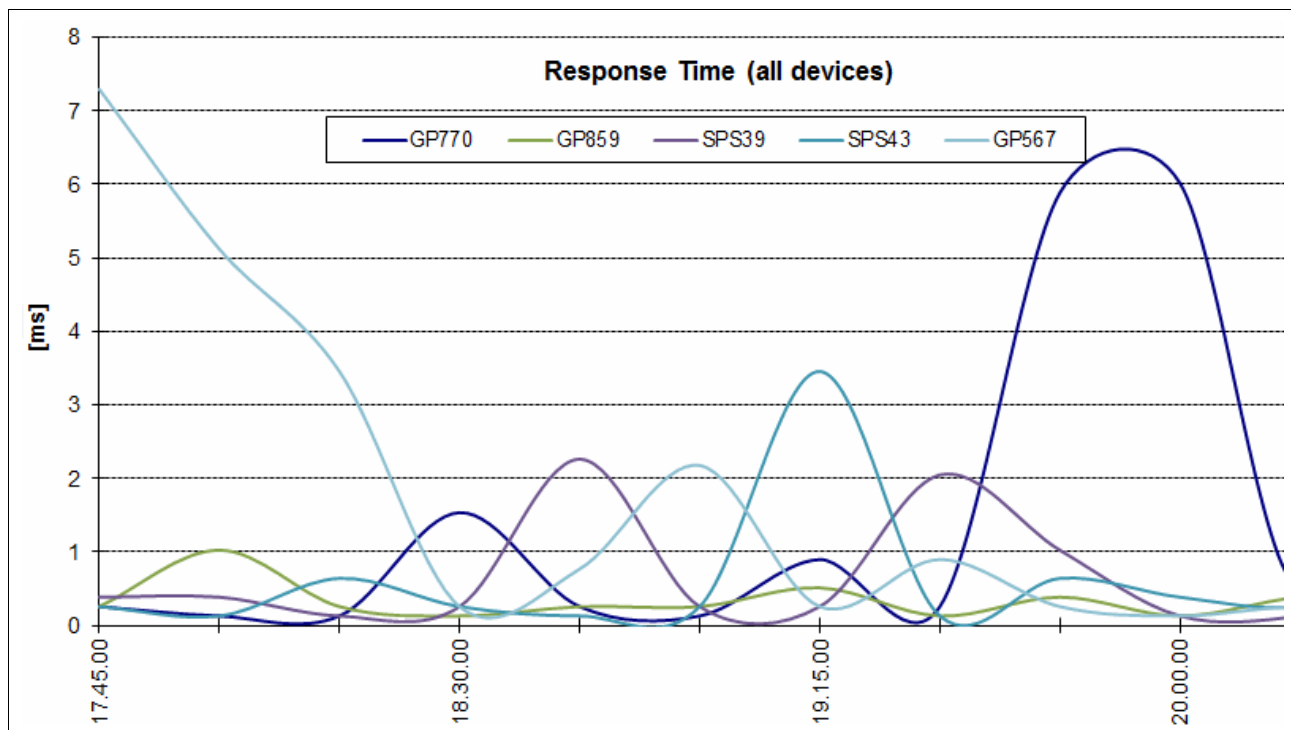


Figure 3-15 DASD response time overview by device



Part 2

Subsystem monitoring

Subsystem monitoring can occur at several different levels. The z/OS system programmer may be monitoring RMF (see Chapter 3, “System z related information” on page 29) to catch CPU constraints, WLM goals being missed, paging activity, as well as DASD device hot spots. All of these could bear evidence of bottlenecks affecting the DB2 subsystem or data sharing group performance. Generally in DB2, when subsystem monitoring is mentioned, we are referring to analysis of the SMF 100 statistics records. These statistics can help point toward DSNZPARMs, buffer pool, and other storage pool constraints, application contention, and wasted CPU cycles burned due to lock/latch contention.

In this part of the book, we describe the information and the actions for DB2 subsystem monitoring by using statistics. We give guidelines on which statistics to gather and how to analyze the health of the system component.

This part contains the following chapters:

- ▶ Chapter 4, “System address space CPU time” on page 53
- ▶ Chapter 5, “EDM pools” on page 61
- ▶ Chapter 6, “Data set open and close” on page 77
- ▶ Chapter 7, “Log activity” on page 83
- ▶ Chapter 8, “IRLM, locking, and latching” on page 89
- ▶ Chapter 9, “The zIIP engine and DB2” on page 109
- ▶ Chapter 10, “Buffer pools and group buffer pools” on page 125
- ▶ Chapter 11, “DDF activity” on page 153
- ▶ Chapter 12, “Workfiles, RID, and sort pools” on page 167
- ▶ Chapter 13, “Virtual and real storage” on page 179



System address space CPU time

This chapter is centered around the system address space level CPU consumption. The statistics report shows each of the 4 core address spaces (MSTR, DBM1, DIST, and IRLM) broken down by TCB, SRB, and zIIP time. We consider how these numbers should compare to each other, and what the DBA or system's programmer can do to affect these numbers.

This chapter contains the following topics:

- ▶ What contributes to address space CPU time
- ▶ How to look at the numbers

4.1 What contributes to address space CPU time

There are some z/OS related terms that you need to understand and be able to relate them to DB2 processes to read the following chart. To identify and keep track of its work, the z/OS operating system represents each unit of work with a control block. Two types of control blocks represent dispatchable units of work: Task control blocks or TCBs represent tasks executing within an address space; service request blocks or SRBs represent higher priority system services. Service request blocks are further broken down into preemptible, non-preemptible, or enclaves. If you are familiar with the statistics report, some of this may make more sense as we break these units out based on the DB2 address space and CPU time.

Figure 4-1 shows what runs in SRB and TCB mode. Certain processes have changed behavior over previous releases as well. For instance, backout processing ran in non-preemptible SRB mode prior to DB2 10. There were scenarios where this behavior could be disruptive, as when the customer is running a single engine LPAR. When a transaction was rolled back in DB2, since backout was non-preemptible, no other task could interrupt it and the system seemed to stall. In DB2 10, it became preemptible, thus it can be interrupted by the dispatcher in favor of another more important task to receive those CPU cycles.

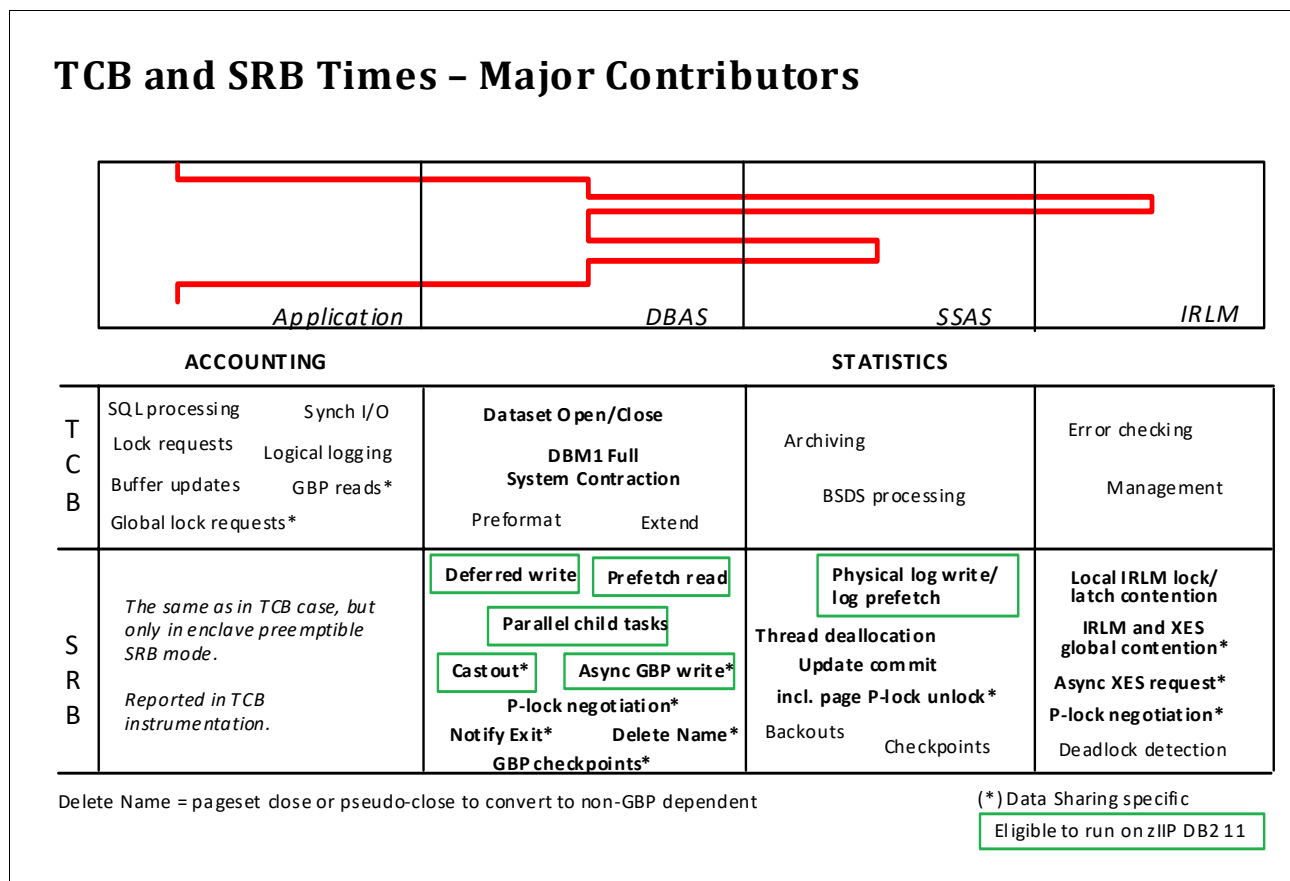


Figure 4-1 CPU times by address space

The dispatchable unit most customers have become familiar with over the past few years is *enclave SRB* work, which is the only type of unit that is zIIP eligible. DDF work is always placed in an enclave, so it was one of the first zIIP eligible workloads, and DB2 moved some of the system agents to enclave SRB in DB2 10 and DB2 11.

As transactions enter DB2, they may hit all four major address spaces (DIST, DBM1, MSTR, and IRLM) as well as WLM managed stored procedure address spaces. Within each address space, difference services or processes consume CPU time in either SRB or TCB mode. In the accounting report, you can find items such as SQL processing, sync I/O, and logical logging in TCB mode reported in the Class 2 CPU time. SRB processing occurs when the thread is running in enclave SRB mode, such as a distributed thread. A portion of these distributed threads becomes zIIP eligible and that time is reported as SE CPU TIME (although it is doing the same processing that would have shown up in TCB time or CP CPU time, without a zIIP).

The DIST address space time is not included in the examples because generally the DIST TCB time is low, and the preempt SRB time is included in the TCB time for the DDF workload, which is roughly 60% zIIP eligible. If by some chance, the DDF TCB time seems elevated, it could be from IBM RACF® user signon authentication or client encryption settings. The ability for the DBA to influence these numbers is minimal and hence we do not discuss it.

Looking at CPU times across release boundaries can be very misleading, and often customers perceive a major drop in CPU, when it may just have been moved around to another processor. Here are some equations to keep in mind:

- ▶ **TOTAL TIME = TCB TIME + PREEMPT SRB + NONPREEMPT SRB**
This is the total CPU Time on *general CP* (so it is not the entire picture).
- ▶ **PREEMPT IIP SRB**
This is the total CPU Time on zIIP.
- ▶ **TOTAL CPU TIME = TOTAL TIME + PREEMPT IIP SRB**
This is the real total processing time and where you should compare across a release boundary.
- ▶ **TOTAL SRB TIME = PREEMPT SRB + NONPREEMPT SRB + PREEMPT IIP SRB**
The total CPU time would be useful when comparing different DB2 or application releases. Users have often ignored the PREEMPT IIP SRB time in the past since it may not apply to their monthly software license charge, but that processing did not disappear. This number should be taken into account to measure the DBM1 and MSTR address space times and understand where CPU cycles are being spent. Total SRB time is one to keep in mind as users cross release boundaries since this is generally when processes switch execution modes in DB2. For instance, with DB2 10, prefetch and deferred write has moved from NONPREEMPT SRB to PREEMPT IIP SRB if a zIIP engine is installed.

Example 4-1 shows the TOTAL CPU TIME with DB2 9.

Example 4-1 CPU times from a DB2 9 customer

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT IIP SRB
SYSTEM SERVICES ADDRESS SPACE	9.110613	0.000000	15.724902	24.835515	N/A
DATABASE SERVICES ADDRESS SPACE	11.225272	0.000000	2:08.598926	2:19.824199	0.000000
IRLM	0.133232	0.000000	7.310644	7.443875	N/A
DDF ADDRESS SPACE	0.174751	6:13.258498	14.847385	6:28.280634	9:01.598026

After moving to DB2 10, there could still be preemptible SRB tasks that are not zIIP eligible. Hence we see time in both PREEMPT SRB and PREEMPT IIP SRB. See Example 4-2.

Example 4-2 CPU times from a DB2 10 customer

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT IIP SRB
SYSTEM SERVICES ADDRESS SPACE	17.316781	10.854370	1.499365	29.670516	N/A
DATABASE SERVICES ADDRESS SPACE	19.647342	10.675517	0.605379	30.928238	6:47.777756
IRLM	0.135340	0.000000	8.655075	8.790415	N/A
DDF ADDRESS SPACE	1.028290	12:44.002157	11.534363	12:56.564811	24:26.104312

Using the formula $\text{PREEMPT SRB} + \text{NONPREEMPT SRB} + \text{PREEMPT IIP SRB}$ on both of the foregoing examples, we get 2:08 and 6:57 respectively (ignoring anything beyond the decimal). If you did not include all of these SRB times, you might assume there was 2:08 of SRB time in DB2 9, and only about 11 seconds in DB2 10. Did the processing cycles disappear? NO. Actually there is 4:49 more processing time in DB2 10, but all of that extra work went to the zIIP engine. So in this case, they were not chargeable CP cycles, but they were cycles nonetheless and, if you are comparing releases, you need to take them into account. Although this was not a perfect benchmark, the example shows roughly the same workload running, and the large increase in dynamic prefetch in DB2 10 added to some of this SRB time increase.

Another key factor is to ensure that the amount of work accomplished in each test is the same. Within a test environment that has a reproducible workload it should be easier to have an *apples-to-apples* comparison. But even if you believe the workload to be symmetric, when comparing release to release performance metrics you should normalize the address space times to get down to the smallest unit of work and look at commit or rollback. This is shown in Example 4-3.

Example 4-3 Divide the address space times by the amount of work coming through the system

For data in the statistics report:
MSTR TCB / (commits + rollbacks)
MSTR SRB / (commits + rollbacks)
DBM1 TCB / (commits + rollbacks)
DBM1 SRB / (commits + rollbacks)
DBM1 IIP SRB / (commits + rollbacks)
IRLM TCB / (commits + rollbacks)
IRLM SRB / (commits + rollbacks)

For data out of the accounting report:
Average Class 2 CP CPU * occurrences / (commits + rollbacks)
Average Class 2 SE CPU * occurrences / (commits + rollbacks)

Regarding the statistics report, it is not as difficult to find the number of commits and rollbacks in the foregoing example. The TOTAL COMMITS field in the "HIGHLIGHTS" section of the statistics report in Example 4-4 shows you this as one number. It contains the number of commits and rollbacks, regardless of single phase, synch, or two phase commit from both the subsystem services side as well as from DRDA® remote locations.

Example 4-4 Where to find commits and rollbacks

---- HIGHLIGHTS -----					
INTERVAL START :	13-09-05 19:37:00.00	SAMPLING START:	13-09-05 19:37:00.00	TOTAL THREADS :	25.00
INTERVAL END :	13-09-05 21:32:00.00	SAMPLING END :	13-09-05 21:32:00.00	TOTAL COMMITS :	173.00
INTERVAL ELAPSED:	1:54:59.999188	OUTAGE ELAPSED:	0.000000	DATA SHARING MEMBER:	N/A

Starting in DB2 10 with APAR PM62797, IFCID 369 can be used to get aggregated accounting and statistics report. The times are all aggregated by connection type, and the quantity column represents the number of transactions that occurred for the connection type. See Example 4-5.

Suppose your end users complain that they are seeing longer elapsed times in their distributed applications. As a starting point, you could run a statistics long report for an hour from the day before, and then today to determine if any variance can be seen from the subsystem level. Depending on the number of transactions, the average elapsed time might have increased.

If the issue was due to CPU consumption, or an issue with the DASD subsystem, then seeing the increase in CL3 SUSP at the aggregate level might help you determine that the issue does not lie within DB2 or the application. An accounting report would be needed to drill down further, but this could be used as a first step.

Example 4-5 IFCID 369 aggregated statistics and accounting

CONNTYPE	CL1 ELAPSED	CL1 CPU	CL1 SE CPU	CL2 ELAPSED	CL2 CPU	CL2 SE CPU	CL3 SUSP	CL2 NOT ACC	QUANTITY
BATCH	17:46:14.752	11:59.797087	0.000000	4:14:31.6061	11:57.803360	0.000000	3:06:16.5194	56:18.579308	29508.00
CICS	N/P	N/P	N/P	N/P	N/P	N/P	N/P	N/P	0.00
DDF	N/P	N/P	N/P	N/P	N/P	N/P	N/P	N/P	0.00
IMS	N/P	N/P	N/P	N/P	N/P	N/P	N/P	N/P	0.00
RRSAF	1:27.766916	0.088142	0.000000	1:27.758517	0.080699	0.000000	0.088142	1:27.589677	285.00
UTILITY	13:00:10.852	8:07.935909	0.418735	5:54:01.6924	5:41.571957	0.418735	2:34:48.3142	3:13:31.8235	38116.00

4.2 How to look at the numbers

With a basic understanding of which address spaces, and in what mode, many of the services within DB2 run, you are better prepared to look for abnormal conditions or opportunities for improvement. Migrating to a new release of z/OS or DB2, and even a large application roll-out, warrants taking another look at the statistics report to ensure nothing has changed drastically. The first step is to examine the address spaces CPU time. See Figure 4-2.

System Address Space CPU Time					
CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT I I P SRB
SYSTEM SERVICES AS	8. 749583	56. 375746	1. 198326	1: 06. 323655	N/A
DATABASE SERVICES AS	1. 807715	14. 305106	1: 54. 209902	2: 10. 322723	1: 40. 832862
IRLM	0. 000931	0. 000000	42. 573512	42. 574443	N/A
DDF AS	10. 682058	13: 58. 424540	1: 59. 796978	16: 08. 903575	14: 56. 025519
TOTAL	21. 240287	21: 07. 030296	4: 37. 778719	20: 08. 124397	16: 36. 858381

Figure 4-2 Address space CPU times

Tip: Some rules of thumb for CPU time, which, if violated, should be investigated by the customer and possibly by IBM:

- All TCB times (System Services, Database Services, IRLM, and DDF) should be low relative to the MSTR and DBM1 SRB times.

One exception to this could be if there are data capture IFI reads running against the member. This would raise MSTR TCB time, without much opportunity for tuning.

- IRLM SRB time should be low relative to MSTR and DBM1 SRB.

The methodology is to monitor and react to unbalance conditions or significant changes over an interval or major event. Often the most severe ones are symptoms of a defect within DB2, but there are some that you can take action against on your own.

If the following suggestions do not apply or cannot be tuned, the backup plan could be to start some traces to find out which service task is responsible for high TCB time. In this case, collect IFCID 49 and 50 for begin/end of TCB service task execution unit switch. If you are interested in investigating high SRB time, then IFCID 47-48 could be used. In general, spikes in address space TCB time are more difficult to tune on the user side.

These are some possible options for lowering the respective CPU times:

► Application:

The tuning perspective here seems rather obvious, but it is not covered in this book. SQL access path tuning to reduce getpages, I/O, and lock requests (lock avoidance covered in 8.3, “Lock avoidance” on page 94) will reduce the chargeable CPU for non-enclave SRB threads, and zIIP cycles for zIIP eligible DRDA requests.

The number of bypassed columns is reported in the miscellaneous section of the statistics report. See Example 4-6.

This represents the number of columns times the number of rows that could have benefitted from fast column processing or xPROCs (zPROCS in DB2 11, since they are above the bar). This could represent up to a 10% performance improvement in the DB2 Class 2 CPU time as it aids in moving large numbers of columns between modules in DB2. In DB2 11, there is a further performance enhancement where DB2 can group up to 7.5x more columns in this process than in previous releases. How do you gain this benefit back? Simply rebind in the current release. IFCID 224 identifies the offending package/collection/number of columns. Note that this is only for SQL SELECTs as DELETE and UPDATE are done dynamically.

Example 4-6 SPROC performance benefit not being utilized

MISCELLANEOUS	VALUE
-----	-----
HIGH LOG RBA	000005AC28073659
BYPASS COL	7351.00

► MSTR- master address space time:

– TCB:

The cost of archiving and BSDS maintenance cannot be avoided but it can be lessened. The rule of thumb is to have at least 6 hours of log data in the active logs. Often times we see customers wrapping through not one, but all of the active data sets within a matter of seconds during insert intensive batch processes. Making the log data sets as large as possible (4GB - 1 byte) and having many of them will reduce this TCB time, lessen the number of system checkpoints, and also give you a safety cushion in the event of an active log offload hang. As of DB2 9, you can have 93 active log data set pairs, and in DB2 10, they can be added dynamically, so there is no excuse for wrapping them within minutes of each other. This can also help in backout or recovery because more of the log undo process will be initiated from DASD instead of tape. Another contributor to the TCB time can be a peer DB2 doing IFI 306 reads against this member's BSDS.

- SRB:

Thread deallocation time can be reduced with more thread reuse, obviously, and items such as protected entry threads, high performance DBATS, and RELEASE(DEALLOCATE) come in to play.

MVS Real Storage Manager (REAL_STORAGE_MANAGEMENT DSNZPARM) can also have an influence here as DB2 storage is contracted and given back to MVS. REAL_STORAGE_MANAGEMENT=ON could have a noticeable effect if it were turned on in a development environment to keep the storage footprint to a minimum.

- DBM1 - DBM1 address space time:

- TCB:

Data set open and close can be seen in the statistics report as part of the buffer pool section: NUMBER OF DATA SET OPENS (rule of thumb is < 0.1 to 1 per second). This can be lowered by ensuring that you do not hit DSMAX ZPARM setting (DSETS CLOSED-THRESH.REACHED), as well as by avoiding data sets ping-ponging in and out of group buffer pool dependency. The PCLOSEN and PCLOSET DSNZPARMs should be set so that DSETS CONVERTED R/W -> R/O is < 10-15 per minute in the statistics report.

Hitting full system contraction is the most counter-productive way to burn CPU cycles in this area. The LPVT (local pool vector table) storage latch is held¹ and does not allow any pool to gain storage while DB2 is encroaching on the storage cushion (FULL SYSTEM CONTRACTIONS in the statistics report).

For preformat and extend processing, setting a generous primary and secondary quantity can reduce this component, as well as DB2 9 going from 2 to 16 preformatted cylinders.

- SRB:

The amount of general CP processing time used by the SRB component of the DB2 address spaces has largely been moved to the zIIP processor since DB2 10. Roughly 80% of the DBM1 address space processing is zIIP eligible starting in DB2 10. Hence, there is less emphasis on tuning, but it should be monitored for noticeable changes during upgrades. Between DB2 9 and 10, there is a noticeable increase due to row level sequential detection for dynamic prefetch, as well as list prefetch for disorganized indexes. So if you do not have a zIIP processor, this can show up as an increase in CPU consumption.

- IRLM - IRLM address space time:

- SRB:

P-Lock negotiation could be exacerbated by data sets constantly going in and out of group buffer pool dependency (and can show up under DBM1 time as well); deadlock contention could be noticeable if, for instance, you are using SAP and you have lowered the deadlock detection cycle down to the millisecond range (their suggestion is 5,1 but DB2's default is 1,1). Regarding the second number, only 1 is accepted for the deadlock detection cycle.

¹ Holding LPVT usually shows up as LC31 contention



EDM pools

The environmental descriptor manager (EDM) pools contain skeleton application plans and packages, database descriptors, and cached dynamic SQL statements. After a discussion of the evolution of the allocation of the EDM pool, we examine its contents.

This chapter contains the following topics:

- ▶ EDM pools overview
- ▶ Plan and package storage and skeleton pool
- ▶ DBD pool
- ▶ DB2 statement caching

5.1 EDM pools overview

The environmental descriptor manager (EDM) pools are the sets of main storage, which contain skeleton application plans and packages (SKCT, SKPT), database descriptors (DBD), and cached dynamic SQL statements.

With DB2 V7, all the objects were placed in a single EDM pool. This has caused some performance issues related to virtual storage constraints.

Starting from V8, DB2 has started making changes to place separate pools above the 2 GB bar as shown in Figure 5-1.

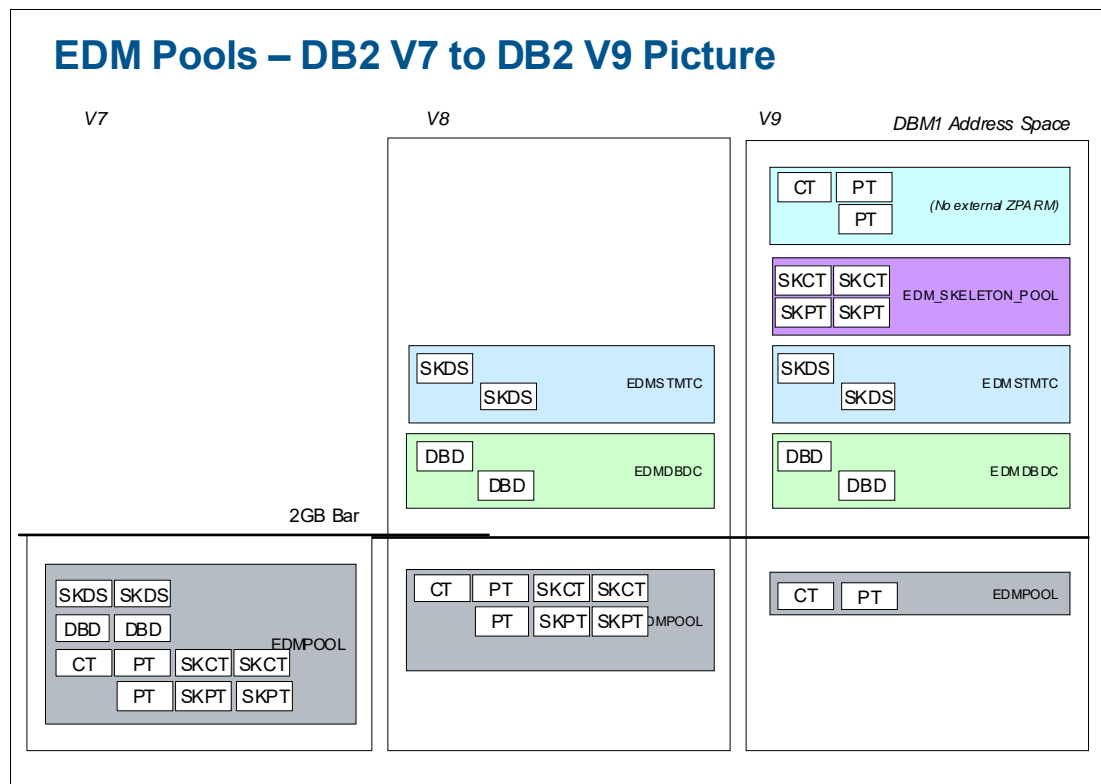


Figure 5-1 EDM pools from DB2 V7 to DB2 V9

In DB2 10, all the EDM pools are placed above the 2 GB bar. What is left below the bar is the agent local storage, which hosts the packages bound prior to DB2 9, for compatibility. You can expect to gradually move those packages to an agent local storage above the bar as applications are bound in DB2 10 or later.

Figure 5-2 shows the structure of EDM pools with DB210.

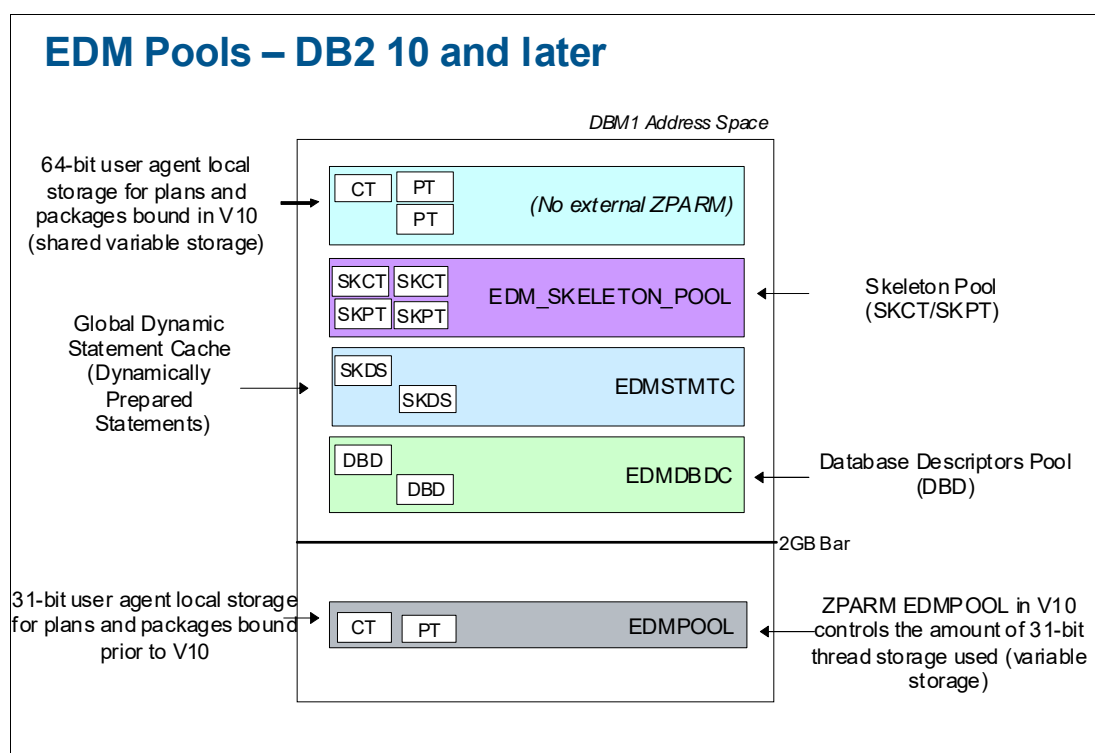


Figure 5-2 EDM pools in DB2 10 and later

Each EDM pool has separate parameters to adjust its maximum/minimum size, and all parameters can be dynamically changed online.

DB2 11 introduced new limits for EDM pools. They can be allocated up to 4 GB, which may benefit those objects that did not fit within the current limit. However, having very large EDM pools might introduce a new bottleneck on EDM pools. When you have a large amount of active objects that might not fit into the assigned large EDM pools, you might see a high LC24 counter resulting from storage needed to be freed often to allow loading new objects into the EDM pools. You need to monitor free pages for EDM pools and latch counters. You should not have this issue as long as you have sufficient spare storage.

Table 5-1 gives a summary of parameters related to EDM pools in DB2 10 and later.

Table 5-1 Summary of parameter relates to EDM pools

Parameter	Description
EDMPOOL	The maximum KB of storage below the 2 GB bar that can be consumed by the EDM. A value of 0 means that 31-bit EDM consumption is unbounded.
EDM_SKELETON_POOL	The minimum size of the EDM skeleton pool in KB.
EDMSTMTTC	The size (in KB) of the statement cache that is to be used by the EDM.
EDMDBDC	The minimum size (in KB) of the DBD cache that is to be used by the EDM.

All EDM pools can be monitored with the OMEGAMON PE statistics report except for the CT/PT storage, which was hosted in the EDM main pool until DB2 9, and moved to agent local storage from DB2 10.

The storage usage for the CT/PT can still be monitored using IFCID 225 field of the OMEGAMON PE statistics report. This is discussed in 5.2, “Plan and package storage and skeleton pool”. There are also different fields related to dynamic statements cache, which are discussed in 5.4, “DB2 statement caching”. Example 5-1 shows the layout of the EDM pool fields from the DB2 statistics trace taken with DB2 11.

Example 5-1 EDM pool block from the statistics report

EDM POOL	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
PAGES IN DBD POOL (ABOVE)	10240.00	N/A	N/A	N/A
HELD BY DBD	204.64	N/A	N/A	N/A
STEALABLE PAGES	110.64	N/A	N/A	N/A
FREE PAGES	10035.36	N/A	N/A	N/A
% PAGES IN USE	0.92	N/A	N/A	N/A
FAILS DUE TO DBD POOL FULL	0.00	0.00	0.00	0.00
PAGES IN STMT POOL (ABOVE)	30720.00	N/A	N/A	N/A
HELD BY STATEMENTS	2149.02	N/A	N/A	N/A
FREE PAGES	28570.98	N/A	N/A	N/A
FAILS DUE TO STMT POOL FULL	0.00	0.00	0.00	0.00
PAGES IN SKEL POOL (ABOVE)	20480.00	N/A	N/A	N/A
HELD BY SKCT	22.00	N/A	N/A	N/A
HELD BY SKPT	2409.00	N/A	N/A	N/A
STEALABLE PAGES	2431.00	N/A	N/A	N/A
FREE PAGES	18049.00	N/A	N/A	N/A
% PAGES IN USE	0.00	N/A	N/A	N/A
FAILS DUE TO SKEL POOL FULL	0.00	0.00	0.00	0.00
DBD REQUESTS	199.00	0.03	7.96	1.15
DBD NOT FOUND	1.00	0.00	0.04	0.01
DBD HIT RATIO (%)	99.50	N/A	N/A	N/A
CT REQUESTS	25.00	0.00	1.00	0.14
CT NOT FOUND	0.00	0.00	0.00	0.00
CT HIT RATIO (%)	100.00	N/A	N/A	N/A
PT REQUESTS	455.00	0.07	18.20	2.63
PT NOT FOUND	0.00	0.00	0.00	0.00
PT HIT RATIO (%)	100.00	N/A	N/A	N/A
PKG SEARCH NOT FOUND	2.00	0.00	0.08	0.01
PKG SEARCH NOT FOUND INSERT	0.00	0.00	0.00	0.00
PKG SEARCH NOT FOUND DELETE	0.00	0.00	0.00	0.00
STATEMENTS IN GLOBAL CACHE	334.97	N/A	N/A	N/A

5.2 Plan and package storage and skeleton pool

DB2 10 removed RDS GETMAINED storage pools. The plan and package structures (CTs/PTs) are now allocated in agent local storage below and above the bar. Because agent local storage is separately owned by each thread, they eliminated the needs of latching, and eliminated LC24 contention.

Although we do not have a separate EDM pool to host CTs/PTs in DB2 10, the EDMPOOL DSNZPARM parameter is still available to limit the storage below the 2 GB bar for CTs/PTs bound prior to DB2 10. The EDMPOOL parameter can be set to 0, meaning below the bar EDM consumption is unbounded. If you have virtual storage constraint below the bar and are migrating to DB2 10, keep the EDMPOOL values to whatever value set in pre-DB2 10 environment for fallback. With DB2 10, the column procedures (SPROC, IPROC, and so on) are still allocated and shared below the 2 GB bar even if your applications is bound on DB2 10. DB2 11 has been enhanced to allocate the column procedures above the 2 GB bar.

Since DB2 10 eliminated pools for CTs/PTs, you need to look for 31 bit virtual storage blocks to understand how much DB2 storage below the bar is consumed against EDMPOOL DSNZPARM.

The “THREAD PLAN AND PACKAGE STORAGE (MB)” field in Example 5-2 shows how much storage is being used below the bar. Make sure to keep enough storage left for CTs/PTs against EDMPOOL and plan to REBIND your packages to ease those constraints after migrating to DB2 10.

Example 5-2 Plan and package storage below 2 GB

DBM1 AND MVS STORAGE BELOW 2 GB		QUANTITY
-----		-----
TOTAL DBM1 STORAGE BELOW 2 GB	(MB)	50.64
TOTAL GETMAINED STORAGE	(MB)	4.57
EDM POOL	(MB)	0.00
TOTAL VARIABLE STORAGE	(MB)	26.75
TOTAL AGENT LOCAL STORAGE	(MB)	6.30
TOTAL AGENT SYSTEM STORAGE	(MB)	5.32
NUMBER OF PREFETCH ENGINES		371.00
NUMBER OF DEFERRED WRITE ENGINES		300.00
NUMBER OF CASTOUT ENGINES		136.00
NUMBER OF GBP WRITE ENGINES		116.00
NUMBER OF P-LOCK/NOTIFY EXIT ENGINES		37.00
TOTAL AGENT NON-SYSTEM STORAGE	(MB)	0.97
TOTAL NUMBER OF ACTIVE USER THREADS		7.02
NUMBER OF ALLIED THREADS		7.02
NUMBER OF ACTIVE DBATS		0.00
NUMBER OF POOLED DBATS		0.00
NUMBER OF PARALLEL CHILD THREADS		0.00
RID POOL	(MB)	N/A
PIPE MANAGER SUB POOL	(MB)	N/A
LOCAL DYNAMIC STMT CACHE CNTL BLKS	(MB)	N/A
THREAD COPIES OF CACHED SQL STMTS	(MB)	0.00
IN USE STORAGE	(MB)	0.00
STATEMENTS COUNT		N/A
HWM FOR ALLOCATED STATEMENTS	(MB)	0.00
STATEMENT COUNT AT HWM		N/A
DATE AT HWM		N/A
TIME AT HWM		N/A

THREAD COPIES OF STATIC SQL	(MB)	0.00
IN USE STORAGE	(MB)	0.00
THREAD PLAN AND PACKAGE STORAGE	(MB)	0.01
BUFFER MANAGER STORAGE CNTL BLKS	(MB)	6.41
TOTAL FIXED STORAGE	(MB)	0.29
TOTAL GETMAINED STACK STORAGE	(MB)	19.03
TOTAL STACK STORAGE IN USE	(MB)	16.39
SYSTEM AGENT STACK STORAGE IN USE	(MB)	11.10
STORAGE CUSHION	(MB)	261.41

Small EDM skeleton pools can result in frequent I/O against DB2 directory tables SCT02 and SPT01 to get the requested plans and packages skeleton structure (SKCTs/SKPTs) into the EDM skeleton pool. This could increase your application's response time.

The Statistics report tells you how much storage has been allocated and how much is held by SKCTs/SKPTs. 'CT HIT RATIO (%)' and 'PT HIT RATIO (%)' are the indicators for how good your EDM skeleton pool size is. 'FAILS DUE TO SKEL POOL FULL' is also a good indicator for keeping your EDM skeleton pool size in good shape. Example 5-3 shows the fields related to plan/packages and skeleton monitoring.

Tip: There are two key metrics for monitoring the skeleton pool:

- ▶ The CT and PT hit ratios indicate how frequently there are I/O operations against the DB2 directory tables; when the respective entries are not found in the pool. Try to keep these numbers as high as possible.
- ▶ Keep 'CT HIT RATIO (%)' and 'PT HIT RATIO (%)' > 95 to 99%

Example 5-3 Monitoring EDM skeleton pool

EDM POOL	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
...				
PAGES IN SKEL POOL (ABOVE)	20480.00	N/A	N/A	N/A
HELD BY SKCT	22.00	N/A	N/A	N/A
HELD BY SKPT	2409.00	N/A	N/A	N/A
STEALABLE PAGES	2431.00	N/A	N/A	N/A
FREE PAGES	18049.00	N/A	N/A	N/A
% PAGES IN USE	0.00	N/A	N/A	N/A
FAILS DUE TO SKEL POOL FULL	0.00	0.00	0.00	0.00
DBD REQUESTS	199.00	0.03	7.96	1.15
DBD NOT FOUND	1.00	0.00	0.04	0.01
DBD HIT RATIO (%)	99.50	N/A	N/A	N/A
CT REQUESTS	25.00	0.00	1.00	0.14
CT NOT FOUND	0.00	0.00	0.00	0.00
CT HIT RATIO (%)	100.00	N/A	N/A	N/A
PT REQUESTS	455.00	0.07	18.20	2.63
PT NOT FOUND	0.00	0.00	0.00	0.00
PT HIT RATIO (%)	100.00	N/A	N/A	N/A
...				

5.3 DBD pool

The database descriptors (DBDs) are the internal representation of DB2 database definitions. They reflect the data definition that is in the DB2 catalog. The size of the DBDs depends on the number of the objects that you create on each database.

As mentioned previously for the EDM skeleton pool, having too small an EDM DBD pool can also result in frequent I/O against DB2 directory DBD01 and can increase the response time or, in an extreme case, an application might result in resource unavailable SQLCODE -904.

The DBD pool size has monitoring fields similar to the EDM skeleton pool. You can apply the same rules when you monitor the DBD pool. Example 5-4 shows the fields related to DBD monitoring.

Tip: The same rules as for the EDM skeleton pool apply to the EDM DBD pool. Keep a high hit ratio to avoid response time degradation and getting errors by having a too small EDM DBD pool:

- ▶ 'DBD HIT RATIO (%)' > 95 to 99%
- ▶ Keep 'FAILS DUE TO DBD POOL FULL' to 0

Example 5-4 Monitoring DBD pool

EDM POOL	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
PAGES IN DBD POOL (ABOVE)	10240.00	N/A	N/A	N/A
HELD BY DBD	204.64	N/A	N/A	N/A
STEALABLE PAGES	110.64	N/A	N/A	N/A
FREE PAGES	10035.36	N/A	N/A	N/A
% PAGES IN USE	0.92	N/A	N/A	N/A
FAILS DUE TO DBD POOL FULL	0.00	0.00	0.00	0.00
...				
DBD REQUESTS	199.00	0.03	7.96	1.15
DBD NOT FOUND	1.00	0.00	0.04	0.01
DBD HIT RATIO (%)	99.50	N/A	N/A	N/A
...				

5.4 DB2 statement caching

The optimizer has become more and more sophisticated as each new release of DB2 has come out. The price you may pay to get the best out of more options of the access path is an increase of the cost of preparing an SQL statement. Applications using dynamic SQL pay this cost at execution time and might have to pay this cost each time they execute.

Back in DB2 for IBM OS/390® Version 5, the feature called *dynamic statement caching* was introduced. Whenever DB2 prepares an SQL statement, it creates a control structure that is used when the statement is executed. When dynamic statement caching is in effect, DB2 stores the control structure associated with a prepared dynamic SQL statement in a storage pool. If that same statement is executed again by the same user, DB2 reuses the cached control structure, avoiding the expense of re-preparing the statement.

There are different levels of statement caching and several functions related to statement cache which can be used to monitor and tune dynamic statements.

In the following sections, we provide a brief overview of statement cache related functions:

- ▶ Behavior of executing dynamic SQL without statement caching
- ▶ Global dynamic statement caching
- ▶ Use of local dynamic statement cache
- ▶ Dynamic SQL literal replacement
- ▶ EXPLAIN MODE special register
- ▶ Capturing data from statement cache
- ▶ Monitoring statement cache

5.4.1 Behavior of executing dynamic SQL without statement caching

Program A prepares a dynamic SQL statement S, executes the prepared statement twice, and terminates. Figure 5-3 shows the execution of dynamic SQL without dynamic statement caching.

After program A had terminated, program B tries to executes exactly the same statement S that program A executed. Program B needs to prepare S and then executes the prepared statement. After issuing a commit, program B wants to execute S again, but program B receives error SQLCODE -514 or -518 (SQLSTATE 26501 or 07003) when S is executed without a prepare. Program B has to prepare the same statement S again, in order to execute statement S.

Each time a prepare has been issued by the programs A and B, via the SQL PREPARE statement, DB2 prepares the statement from scratch. After issuing the commit within a program, the prepared statement is deleted, so the program must prepare the statement again in order to execute the statement.

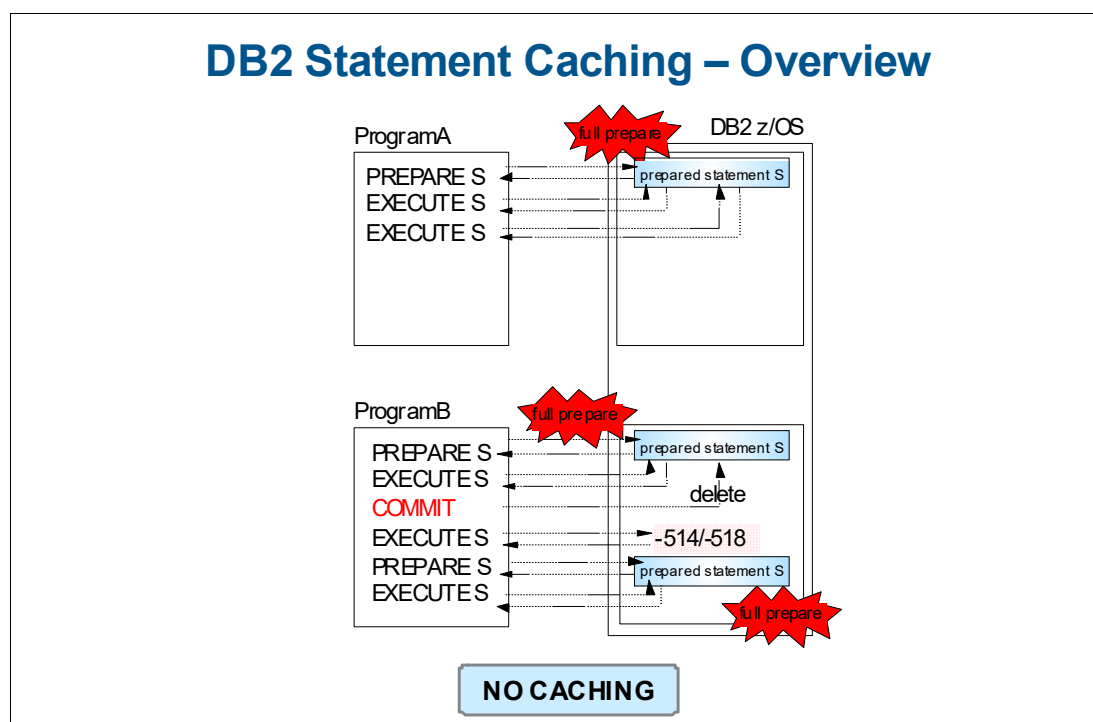


Figure 5-3 Dynamic SQL without statement caching

5.4.2 Global dynamic statement caching

The global dynamic statement cache is allocated in the 64 bit DBM1 address space. You can activate this cache by setting DSNZPARM CACHEDYN=YES. When global dynamic statement caching is active, the skeleton copy of a prepared SQL statement (SKDS) is held in the global dynamic statement cache that forms one of the EDM pools. Only one skeleton copy of the same statement (matching text) is held. The skeleton copy can be used by user threads to create user copies. An LRU algorithm is used for replacement.

If an application issues a PREPARE or an EXECUTE IMMEDIATE (and the statement has not been executed before in the same commit scope), and the skeleton copy of the statement is found in the global statement cache, it can be copied from the global cache into the thread's storage. This reuse process for prepare is called a *short prepare*.

The example in Figure 5-4 shows the same scenario as previously shown without dynamic statement caching where program A prepares a dynamic SQL statement S, executes the prepared statement twice, and terminates.

Program B starts after program A has terminated, prepares the same statement S as program A did, executes the prepared statement and issues a COMMIT. Then program B tries to execute S again. The program receives an error SQLCODE -514 or -518 (SQLSTATE 26501 or 07003) and has to prepare the same statement S again. Then it executes the prepared statement and terminates.

The first time a prepare for statement S is issued by the program A, a complete prepare operation is performed. The SKDS of S is then stored in the global statement cache. When program B executes the prepare of S for the first time, the SKDS is found in the global statement cache and is copied to the local storage of B's thread (short prepare). After the COMMIT of program B, the prepared statement is invalidated in B's local storage, but the SKDS is preserved in the global statement cache in the EDM pool. Because neither the statement string nor the prepared statement is kept after the commit, program B has to repeat the prepare of statement S explicitly. This causes another copy operation of the SKDS in the global cache to the local storage of the thread of application B (short prepare).

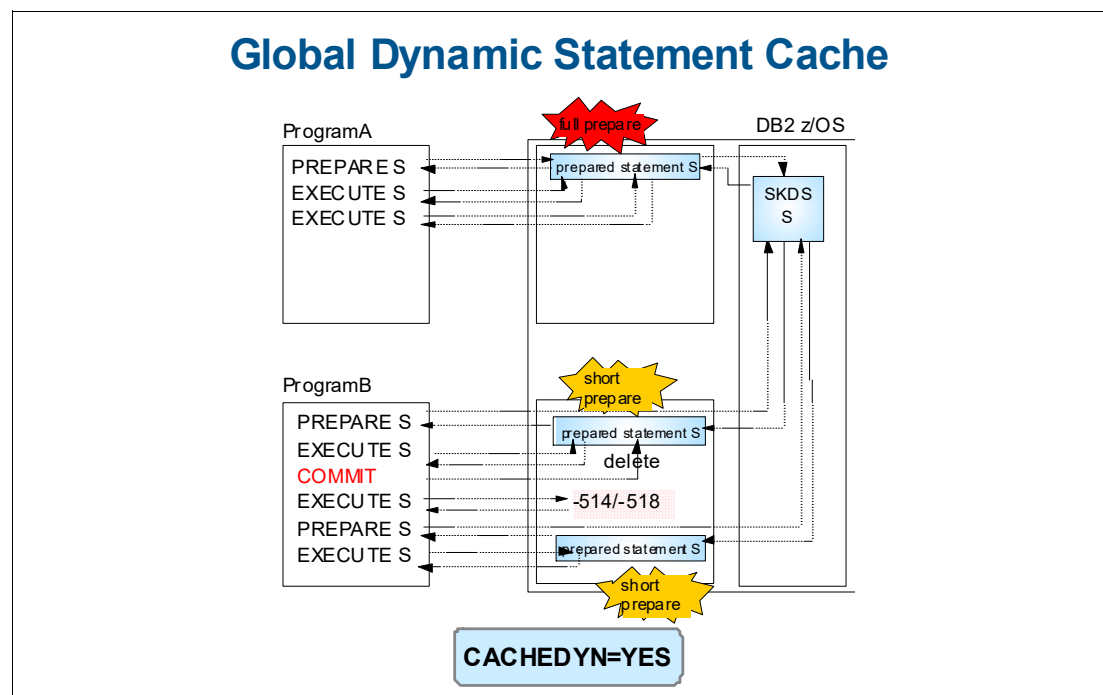


Figure 5-4 Global dynamic statement caching

5.4.3 Use of local dynamic statement cache

A local dynamic statement cache is allocated in the storage of each thread in the DBM1 address space. You can control the usage of this cache by using the `KEEPDYNAMIC` bind option. Also the maximum number of prepared dynamic SQL statements can be limited by `DSNZPARM MAXKEEPD` to help limit the amount of storage that these applications use.

Bound with `KEEPDYNAMIC YES`, an application can issue a `PREPARE` for a statement only once and omit subsequent `PREPARES` for this statement, even after a commit has been issued.

The example in Figure 5-5 shows a similar scenario with local dynamic statement cached turned on with `KEEPDYNAMIC YES`.

Program A prepares a dynamic SQL statement S, executes the prepared statement twice, and terminates. Program B then starts after program A has terminated, prepares the same statement S as A did, executes the prepared statement, issues a commit, executes S again, and terminates.

The first time a prepare for statement S is issued by program A, a complete prepare is done (full prepare). The SKDS of S is stored in the global cache. When program B executes the prepare of S the first time, the SKDS is found in the global statement cache and is copied to the local statement cache of B's thread (short prepare). The `COMMIT` of program B has no effect on the prepared statement. When full caching is active, both the statement string which is also kept for local caching only, and the prepared statement are kept in the thread's local storage after a commit. Therefore, program B does not have to repeat the prepare of statement S explicitly, and it is not necessary to prepare the statement implicitly since the full executable statement is now kept in the thread's local storage. This case is called *prepare avoidance*.

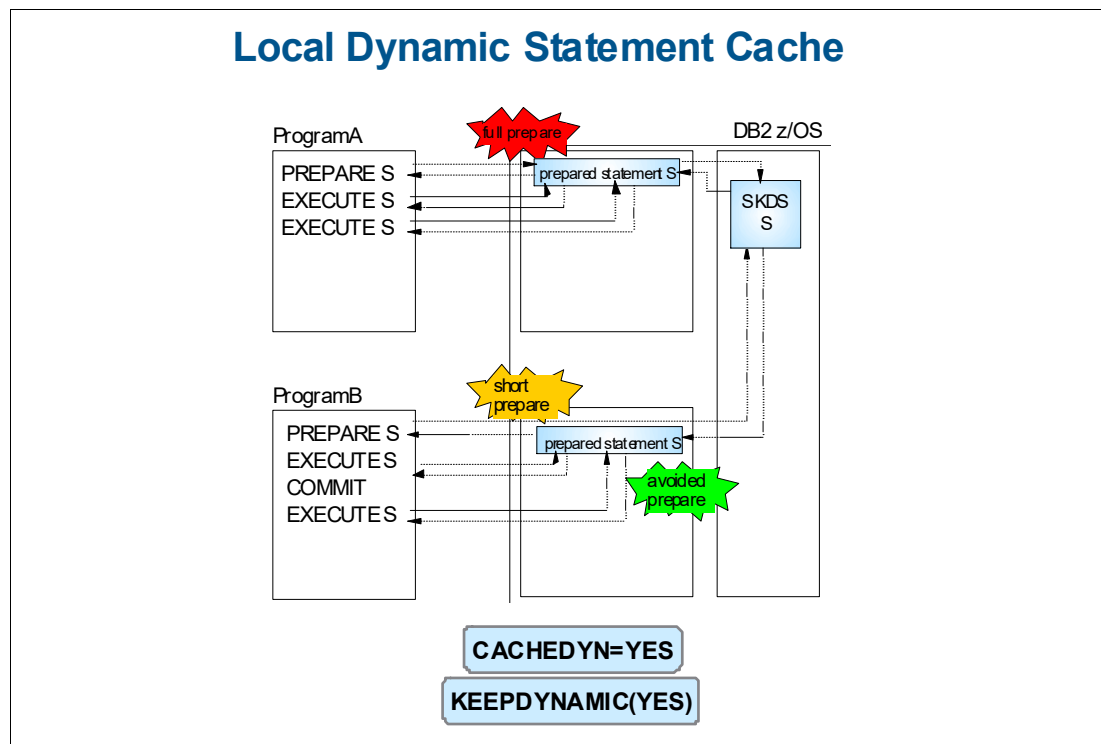


Figure 5-5 Local dynamic statement caching

Certain types of applications, which made heavy use of local dynamic statement cache in a large scale prior to DB2 9 might have storage issues because the whole of this local dynamic statement cache was held below the 2 GB bar. This could lead to virtual storage constraint. If DB2 thinks the DBM1 address space is short of storage, then DB2 starts freeing statement sections at the end of the section as opposed to waiting for commit and using the MAXKEEPD threshold. Table 5-2 shows the differences in behavior depending the setting of values.

Table 5-2 Settings for CACHEDYN_FREELOCAL parameter

Value	Description
0	Off (default for V8)
1	If (LDSC >=500MB & DBM1 Used >=75%) then free >= 100KB statement If DBM1 Used >=85% then free any statement (default for V9 or later)
2	If (LDSC >=500MB & DBM1 Used >=80%) then free >= 100KB statement If DBM1 Used >=88% then free any statement
3	If (LDSC >=350MB & DBM1 Used >=75%) then free >= 100KB statement If DBM1 Used >=88% then free any statement

5.4.4 Dynamic SQL literal replacement

To take advantage of the dynamic statement cache, the SQL statement string has to be identical and executed by the same user. This enforces the requirement of coding guidelines promoting the use of parameter markers (“?”) in the SQL statement, in order to get the maximum performance benefits.

Many applications and development environments generate SQL statements for the developers and often users have no or little control on the way the statements are written. This makes it difficult, if not impossible, to exploit parameter markers, and such applications will use literals instead. Literals are likely to be different at each SQL statement execution with little reuse of the statement cache. This produces a degradation of performance by requiring a PREPARE at most SQL statement invocation.

Starting from DB2 10 in NFM, more SQL statements can be reused in the cache across users. Dynamic SQL statements can now be shared with an already cached dynamic SQL statement if the only difference between the two statements is literal values; when stored in the cache, literals are replaced with an ampersand (“&”) that behaves similar to parameter markers. This avoids a full PREPARE and can provide a performance improvement similar to what is gained by coding SQL statements with parameter markers.

To enable this function, use one of the following methods:

- Change the PREPARE statement to include the new ATTRIBUTES clause, specifying CONCENTRATE STATEMENTS WITH LITERALS (the acronym CSWL appears in the instrumentation records).
- On the Java client, pass the JCC Driver with statementConcentrator property to 2, which can be specified in the data source or connection property.
- On the .NET client, StatementConcentrator string connection keyword with value “Literal”, which are to be specified in the IBM Data Server Driver configuration file.
- On the CLI/ODBC client, StmtConcentrator db2cli.ini keyword set to WITHLITERALS.
- Set LITERALREPLACEMENT in the ODBC initialization file in z/OS, which enables all SQL statements coming into DB2 through ODBC to have literal replacement enabled.

One example is represented by the following five dynamic SQL statements:

1. SELECT COL1,COL2,COL3,COL4,COL5 FROM TEST1 WHERE COL1 BETWEEN 100 AND 200 AND COL2 = 'HU'
2. SELECT COL1,COL2,COL3,COL4,COL5 FROM TEST1 WHERE COL1 BETWEEN 200 AND 250 AND COL2 = 'MT'
3. SELECT COL1,COL2,COL3,COL4,COL5 FROM TEST1 WHERE COL1 BETWEEN 200 AND 400 AND COL2 = 'LS'
4. SELECT COL1,COL2,COL3,COL4,COL5 FROM TEST1 WHERE COL1 BETWEEN ? AND ? AND COL2 = 'HU'
5. SELECT COL1,COL2,COL3,COL4,COL5 FROM TEST1 WHERE COL1 BETWEEN ? AND ? AND COL2 = ?

In this example, statements 1 to 3 are all constructed within the application where all the predicates consist of literal values. Statements 4 and 5 use part or all of their predicate specified with parameter markers. The statement concentrator keyword had been passed as datasource property whereby all these statement are executed with statement concentrate attributes.

SQL extracts from the statement cache as results of executing the sample SQL statements using literal replace are shown in Example 5-5. The five SQL statements are concentrated into three statements within DB2. When SQL with all predicates are given literal values, DB2 replaces those literal values with ampersands, which work like parameter markers. If you have at least one predicate with a parameter marker, DB2 uses statement as is to consider data skew statistics for those predicates specified with literal values. Because DB2 matches cached statements on SQL text, literal replaced SQL will not cache hit with a statement written with parameter markers.

Example 5-5 Sample dynamic statement cache listing for dynamic SQL literal replacement

```
-- display dynamic statement cache
EXPLAIN STMTCACHE ALL;                                00050061
-----+-----+-----+-----+-----+-----+-----+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+
SELECT SUBSTR(COLLID,1,22) COLLID,                      00060460
                                STMT_TEXT                00060562
FROM DSN_STATEMENT_CACHE_TABLE                          00061056
WHERE PRMAUTH = CURRENT SQLID                            00070057
AND PROGRAM_NAME = 'SYSJH200';                          00071058
-----+-----+-----+-----+-----+-----+
COLLID                      STMT_TEXT
-----+-----+-----+-----+-----+-----+
DSNDYNAMICSQLCACHE         SELECT COL1,COL2,COL3,COL4,COL5 FROM TEST1 WHERE COL1 BETWEEN ? AND ? AND COL2 = ?
DSNDYNAMICSQLCACHE         SELECT COL1,COL2,COL3,COL4,COL5 FROM TEST1 WHERE COL1 BETWEEN & AND & AND COL2 = &
DSNDYNAMICSQLCACHE         SELECT COL1,COL2,COL3,COL4,COL5 FROM TEST1 WHERE COL1 BETWEEN ? AND ? AND COL2 = 'HU'
```

You can monitor how much you benefit from dynamic SQL literal replacement using OMEGAMON PE statistics report. Within dynamic SQL statement block, you are able to monitor how many statements had been parsed, replaced, and matched during the execution. You can also monitor where duplicate statement instances have been created within statement cache.

As shown in Example 5-6, all five statements had been parsed based on the attributes passed with the statements. Three statements had literals replaced with parameter markers, and one of them needed a full prepare but subsequent statements had a match. Although overhead should be minimum while parsing statements, to get best out of the function, consider passing attributes only for those applications which might benefit from literal replacement function. The OMEGAMON PE statistics report shows how many statements were replaced and benefitted from the function.

Example 5-6 Statistics trace output sample with literal replacement

DYNAMIC SQL STMT	QUANTITY
-----	-----
PREPARE REQUESTS	17.00
FULL PREPARES	3.00
SHORT PREPARES	14.00
GLOBAL CACHE HIT RATIO (%)	82.35
IMPLICIT PREPARES	0.00
PREPARES AVOIDED	0.00
CACHE LIMIT EXCEEDED	0.00
PREP STMT PURGED	11.00
LOCAL CACHE HIT RATIO (%)	N/C
CSWL - STMTS PARSED	5.00
CSWL - LITS REPLACED	3.00
CSWL - MATCHES FOUND	2.00
CSWL - DUPLS CREATED	0.00

5.4.5 EXPLAIN MODE special register

Starting from DB2 10, special register, CURRENT EXPLAIN MODE can be used to turn on and off the collection of access path information for dynamic SQL statements at the application level. Within your application, you can change the value of the special register by issuing the SET CURRENT EXPLAIN MODE. One of the following three values are acceptable:

- ▶ NO (the default): Indicates that no explain information is captured during the execution of an explainable dynamic SQL statement.
- ▶ YES: Indicates that explain information is captured in the explain tables as eligible dynamic SQL statements are prepared and executed.
- ▶ EXPLAIN: Indicates that explain information is captured in the explain tables as eligible dynamic SQL statements are prepared. However, dynamic statements except SET statements, are not executed.

For distributed applications, the EXPLAIN MODE special register can be also set via currentExplainMode JDBC property or DB2Explain IBM Data Server Driver configuration keyword at data source level or application level. See Example 5-7.

Example 5-7 Collecting explain information while executing dynamic SQL using SPUFI

```
SET CURRENT EXPLAIN MODE = YES;
SELECT ...
```

In order for you to use the EXPLAIN MODE special register, it requires both the PLAN_TABLE and DSN_STATEMENT_CACHE_TABLE qualified with the current SQLID that is used when running the application.

If you set CURRENT EXPLAIN MODE to YES, performance data can be collected in regard to running a specific dynamic SQL statement. If you only need access path information collected and are not requiring the application to be executed, set CURRENT EXPLAIN MODE to EXPLAIN.

Example 5-8 shows part of the performance data from DSN_STATEMENT_CACHE_TABLE with the applications executed using different EXPLAIN MODE settings. Collection ID tells you which EXPLAIN MODE you executed the application with. When used with EXPLAIN MODE YES, it populates performance related statistics for you to evaluate your application while you also get access path information populated in the PLAN_TABLE.

Example 5-8 Sample output of statement cache using EXPLAIN MODE

SELECT SUBSTR(COLLID,1,22) COLLID,SUBSTR(PROGRAM_NAME,1,10) PROGNAME, 00060449					
STAT_ELAP ,STAT_CPU, STAT_GPAG, STMT_TEXT				00060545	
FROM DSN_STATEMENT_CACHE_TABLE				00061032	
WHERE PRIMAUTH = CURRENT SQLID;				00070032	
COLLID	PROGNAME	STAT_ELAP		STAT_CPU	STAT_GPAG STMT_TEXT
DSNEXPLAINMODEEXPLAIN	DSNESM68	+0.0	E+00	+0.0	E+00 0 SELECT * FROM
DSNEXPLAINMODEYES	SYSSH200	+0.2127929520593527E-05		+0.1964355079490262E-05	0 SELECT DEPT.D
DSNEXPLAINMODEYES	SYSLH200	+0.6552734022061843E-05		+0.6385253301214463E-05	0 SELECT * FROM
DSNEXPLAINMODEYES	DSNESM68	+0.2516126844520662E-03		+0.1868808503929233E-03	6 SELECT * FROM

5.4.6 Capturing data from statement cache

DB2 maintains statement caching performance statistics records when dynamic statements are cached. The statistics include cache hit ratio and other useful data points that you can use to evaluate the overall performance of your statement cache and statement execution. Start IFCID 318 to collect performance statistics within statement cache.

The EXPLAIN STMTCACHE ALL statement writes performance statistics from the statement cache, if there are any, and the SQL statement to DSN_STATEMENT_CACHE_TABLE. Once it is determined that a statement needs tuning, you can use the statement identifier found in the DSN_STATMENT_CACHE_TABLE as input to execute EXPLAIN STMTCACHE STMTID statement with the statement identifier uniquely assigned within the DB2 member. This writes access path information from the statement cache to the PLAN_TABLE.

For details, see 20.4, “Capturing performance data with the statement cache” on page 284.

5.4.7 Monitoring statement cache

As described in the previous sections, there are different flavors of statement cache as a function. OMEGAMON PE reports on dynamic SQL statements give you both at the global cache and at the local cache level performance information and literal replacement data as described in 5.4.4, “Dynamic SQL literal replacement” on page 71.

The two key metrics for monitoring statement cache are the global dynamic statement hit ratio and the local dynamic statement hit ratio:

Global statement cache hit ratio = ‘SHORT PREPARES’ / (‘SHORT PREPARES’ + ‘FULL PREPARES’)

Local statement cache hit ratio = ‘PREPARES AVOIDED’ / (‘PREPARES AVOIDED’ + ‘IMPLICIT PREPARES’)

Getting these hit ratios high means avoiding full prepares or short prepares, which saves CPU times. As previously explained, to reuse a cached statement, an SQL statement must be an exact match and must be executed by same authorization ID. Caching depends on how the SQL statement is written within application. When statement cache hit ratio is low, enlarging cache size is first option for tuning, but if the SQL statements generated within application use literal values, literal replacement might be an option.

Tip: Try to keep your statement hit ratio as high as possible, to avoid having full prepare or even short prepare for those uses local statement caching.

As a start point, try to keep global statement hit ratio to be 90 to 95% or higher. Similarly, if you are using local statement caching, try to keep your global statement hit ratio to be 70% or higher. You might want to consider size of virtual storage below the 2 GB bar for those which use local statement cache prior to DB2 10.

Example 5-9 shows a global statement hit ratio of 66%. This case might benefit by enlarging the statement cache. You can see the allocated and used size in the EDM pool block as shown in Example 5-1 on page 64.

Example 5-9 Dynamic SQL statement block from statistics report

DYNAMIC SQL STMT	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
PREPARE REQUESTS	10294.00	0.12	0.19	0.05
FULL PREPARES	2803.00	0.03	0.05	0.01
SHORT PREPARES	5510.00	0.06	0.10	0.03
GLOBAL CACHE HIT RATIO (%)	66.28	N/A	N/A	N/A
IMPLICIT PREPARES	0.00	0.00	0.00	0.00
PREPARES AVOIDED	0.00	0.00	0.00	0.00
CACHE LIMIT EXCEEDED	0.00	0.00	0.00	0.00
PREP STMT PURGED	1517.00	0.02	0.03	0.01
LOCAL CACHE HIT RATIO (%)	N/C	N/A	N/A	N/A
CSWL - STMTS PARSED	0.00	0.00	0.00	0.00
CSWL - LITS REPLACED	0.00	0.00	0.00	0.00
CSWL - MATCHES FOUND	0.00	0.00	0.00	0.00
CSWL - DUPLS CREATED	0.00	0.00	0.00	0.00



Data set open and close

Having the needed data sets open and available for use is important for the performance of transactions. However, the number of open data sets affects the amount of available storage, and the number of open data sets in read/write state affects restart time.

This chapter contains the following topics:

- ▶ Open and close data sets
- ▶ Controlling number of open data sets

6.1 Open and close data sets

Opening needed data sets for table spaces and indexes, and keep them available for use is important for the performance of transactions. However, the number of open data sets affects the amount of available storage DB2 uses, and state of read/write for data sets affects restart time. It is importance to keep track of open and close data sets and tune related parameter as you make changes to the application or a system.

Table 6-1 gives a summary of parameters related to open and close data sets.

Table 6-1 Summary of parameters relating to open and close data sets

Parameter	Description
DSMAX	Maximum number of data sets (up to 200,000). DB2 10 needs PM88166 to reach the limit.
PCLOSEN	The number of consecutive DB2 checkpoints that are to be allowed after a page set or partition is updated. Default is 10 checkpoints.
PCLOSET	The number of minutes that can elapse after a page set or partition is updated. Default is 10 minutes.
CLOSE DDL parameter	YES or NO to specify the priority in which data sets are closed when limits on open data sets are reached.

6.1.1 Pseudo close mechanism

DB2 automatically converts updated page sets or partitions from read-write intent to read-only intent state, which is controlled by the pseudo-close mechanism using the pseudo-close intervals (PCLOSEN/PCLOSET DSNZPARM values). Conversion takes place when the number of consecutive DB2 system checkpoints since a page set or partition was last updated meet PCLOSEN, or elapsed time since a page set or partition was last updated becomes PCLOSET.

Converting to read-only intent means that the SYSLGRNX entry is going to be closed for a page set and any updated pages are going to be externalized to disk.

In the data sharing environment, the pseudo close mechanism also controls deferred close of GBP dependent objects with CLOSE YES specified.

6.1.2 CLOSE specification for the table spaces

In DB2 data sharing environment, the CLOSE parameter is also considered to handle open data sets on members with read-only state. Any table space or index which has remained in a read-only state with no activity for more than the pseudo-close intervals (PCLOSEN/PCLOSET DSNZPARMs) results physical close for those objects to reduce data sharing overhead. (The object can become non-GBP-dependent as a result of it, if it is closed on all but one member or if it is open on multiple members but all become read-only.)

Disadvantage of physically closing those objects is that next time the application accesses the same object, they must go through physical open process again. This can be significant overhead, particularly if a large number of objects are accessed at the same time.

In a DB2 data sharing environment, the CLOSE NO means that page sets will physically stay open unless DSMAX hits and all CLOSE YES page sets are closed.

There are significant differences for a table space with CLOSE YES and CLOSE NO in a data sharing environment, particularly having GBP dependency with 1 updater on one and 1 reader at another. Table 6-2 gives a summary of characteristic differences between specifying CLOSE YES versus CLOSE NO.

Tip: If you know the objects are expected to be GBP dependent for most of the time, consider applying CLOSE NO to avoid extra open/close overhead. For objects infrequently becoming GBP dependent, CLOSE YES provides better performance.

Table 6-2 CLOSE YES versus CLOSE NO

	CLOSE YES	CLOSE NO
Behavior	When the next PCLOSET/PCLOSEN occurs for the reader, physical close of the object occurs.	When the next PCLOSET/PCLOSEN occurs for the reader, there is <i>no physical close</i> .
Pros	Data set ceases to be GBP-dependent. The remaining member can update the data set in its local buffer pool.	There is no physical open of objects on next access.
Cons	Applications may need to make physical open of objects on next access.	Data sets can become GBP-dependent for those objects physically opened from multiple DB2 member. You can avoid it by physically closing pageset using the -STOP DB command or -ACCESS DB MODE(NGBPDEP) . Normally stopping DB2, or if reaching DSMAX and all CLOSE(YES) objects have been closed, can also make those data set closed.

6.2 Controlling number of open data sets

Maximum number of data sets opened by DB2 is controlled using subsystem parameter DSMAX. How DSMAX works is that when the remaining number available for open data sets reaches either 1% of DSMAX for maximum number of open data sets or 100, DB2 begins closing either 3% of maximum number of open data sets or 300 data sets, whichever number is less. First, page sets or objects that are defined with the CLOSE YES option are closed. The least recently used page sets are closed first. When no more CLOSE YES data sets are to be closed, DB2 next closes page sets or partitions that are defined with the CLOSE NO option. The least recently used CLOSE NO data sets are closed first. Note that declared global temporary table (DGTT) indexes are not governed by DSMAX.

With V11 or V10 with APAR PM88166 applied, the new limit for open data sets is 200,000. Although you can set up to 200,000 as a parameter, because DB2 uses storage below the 2 GB bar to open each data set, each system environment will have different limits depending on available storage below the bar. Be sure to monitor your virtual storage usage and ensure that you have enough space remaining when you are expanding your DSMAX. z/OS also has limits for the maximum number of open data sets depending on release and enabled functions.

If you use z/OS Version 1 Release 12 or later, consider improving the opening and closing of data sets by updating the ALLOCxx PARMLIB member as follows:

- ▶ Set the SYSTEM MEMDSENQMGMT to ENABLE
if you want it to remain effective across IPLs.
- ▶ Issue the system command: **SETALLOC SYSTEM, MEMDSENQMGMT=ENABLE**
A DB2 restart is required to make the change effective.

6.2.1 Monitoring open and close activity

If you see frequent data set close because of the DSMAX threshold being reached, first consider increasing the DSNZPARM DSMAX with small incremental steps. If DSMAX is not frequently reached, consider changing CLOSE YES to CLOSE NO on data sets that are used by critical applications. Be sure to study the impact on DBM1 storage below the bar when setting a large value for DSMAX.

The impact of frequently opening data sets is typically high DBM1 TCB times and/or accounting class 3 data set open times. You should also check other fields for impact due to having frequently reaching the DSMAX threshold.

Tip: Monitor the number of times the threshold is being hit and data sets are being closed by looking at the field 'DSETS CLOSED-THRESH.REACHED.'

Consider increasing the DSMAX parameter if you are reaching the threshold frequently, for example, 0.1 to 1 per second.

Another important field to monitor is the change from read-write interest to read-only interest state on a data set. Having frequent changes from read-write interest to read-only interest state on a data set means that the application needs to pay expensive SYSLGRNX processing cost each time an application changes them back to read-write interest state for update processing. This could also mean for those page sets to be switched in and out of GBP dependency frequently and it would increase processing of the SYSLGRNX page set.

Having frequent changes from read-write interest to read-only interest has a different impact against a DB2 data sharing environment. Each time the page set or partition is transitioning inter-DB2 read/write interest, DB2 members need to execute complete scans of the local buffer pool. This can be a huge overhead when GBP dependent transition happens too often. With DB2 10 or later versions, these expensive scans against large local buffer pool are avoided.

Tip: You should carefully monitor the following field and avoid having too frequent conversion from read-write interest to read-only interest state, such as 10 to 15 per minute:

'DSETS CONVERTED R/W -> R/O'

A general suggestion is to set CLOSE YES as a design default for DB2 data sets. Also, take frequent system checkpoints starting from CHKRFREQ=2-5 (minutes) and adjust pseudo-close intervals (PCLOSEN/PCLOSET) when having too frequent pseudo closes.

Example 6-1 shows the monitoring of OPEN/CLOSE activity.

Example 6-1 OPEN/CLOSE activity

OPEN/CLOSE ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
OPEN DATASETS - HWM	4656	N/A	N/A	N/A
OPEN DATASETS	4430	N/A	N/A	N/A
DS NOT IN USE,NOT CLOSE-HWM	4652	N/A	N/A	N/A
DS NOT IN USE,NOT CLOSED	4415	N/A	N/A	N/A
IN USE DATA SETS	15	N/A	N/A	N/A
DSETS CLOSED-THRESH.REACHED	0	0.00	0.00	0.00
DSETS CONVERTED R/W -> R/O	11	0.18	11.00	0.00

You can also monitor the number of data set open from the accounting trace at a buffer pool level. See Example 6-2. If you see a high number counted, check the impact on the application experiencing high number of open. You should check DSMAX threshold being hit.

Example 6-2 Monitoring number of open data sets at buffer pool level

BP1	GENERAL	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----	-----
	CURRENT ACTIVE BUFFERS	2263	N/A	N/A	N/A
	UNAVAIL.BUFFER-VPPOOL FULL	0	0.00	0.00	0.00
	NUMBER OF DATASET OPENS	1	0.02	1.00	0.00
	BUFFERS ALLOCATED - VPPOOL	110080	N/A	N/A	N/A
	DFHSM MIGRATED DATASET	0	0.00	0.00	0.00
	DFHSM RECALL TIMEOUTS	0	0.00	0.00	0.00
	VPPOOL EXPANS. OR CONTRACT.	0	0.00	0.00	0.00
	VPPOOL EXPANS. FAILURES	0	0.00	0.00	0.00
	CONCUR.PREF.I/O STREAMS-HWM	0	N/A	N/A	N/A
	PREF.I/O STREAMS REDUCTION	0	0.00	0.00	0.00
	PARALLEL QUERY REQUESTS	0	0.00	0.00	0.00
	PARALL.QUERY REQ.REDUCTION	0	0.00	0.00	0.00
	PREF.QUANT.REDUCED TO 1/2	0	0.00	0.00	0.00
	PREF.QUANT.REDUCED TO 1/4	0	0.00	0.00	0.00
	NUMBER OF LPL INSERTS	0	0.00	0.00	0.00

6.2.2 Managing and collecting additional information for open and close data sets

Open and close performance in DB2 depends heavily on the performance of the ICF catalog, SVC99, and z/OS enqueues. The open and close time is clocked between the time DB2 switches to the open task until it returns. DB2 provides up to 40 tasks to do parallel open. The majority of the work involved in opening a data set is the dynalloc (SVC99) and the actual physical open of the pageset (VSAM OPEN). Close is done asynchronously, so it does not normally reflect in class 3 waits.

The command **ACCESS DATABASE MODE(OPEN)** forces a physical open of table spaces, index spaces, and partitions, and removes the GBP-dependent status for a table spaces, index spaces, or partitions.

Other work that shows up in data set open and close includes the following processing:

- ▶ Time waiting for server in TCP/IP requester accounting
- ▶ VSAM catalog update
- ▶ Parallel query cleanup

Long waits for OPEN/CLOSE are often related to the number of data sets that are open. The more data sets need to be open, the longer open takes.

Contrary to popular belief, IFCID 107 does not trace open/close activity; it maps data set names to DBIDs and OBIDs. Instead you can use IFCID 370 and 371 to trace data set open and close activity. Example 6-3 shows starting IFCID 370 and 371 by starting performance trace class 19.

Example 6-3 -STA TRACE(P) CLASS(19)

```
-DB1A STA TRACE(P) CLASS(19)
DSNW130I -DB1A P TRACE STARTED, ASSIGNED TRACE NUMBER 02
DSN9022I -DB1A DSNWVCM1 '-STA TRACE' NORMAL COMPLETION
```

IFCID 370 and 371 give you data set name and related performance numbers, such as number of opened data sets, allocation time, and open processing time.

Example 6-4 shows the sample report for IFCID 370 and 371 generated by the OMEGAMON PE command:

```
RECTRACE
TRACE
INCLUDE(IFCID(370,371))
FROM(,19:12:00.00) TO(,19:15:00.00)
EXEC
```

Example 6-4 IFCID 370 and 371 report

OPRMAUTH	CONNECT	INSTANCE	END_USER	WS_NAME	TRANSACTION	DESCRIPTION	DATA
ORIGAUTH	CORRNAME	CONNTYPE	RECORD TIME	DESTNO ACE	IFC		
PLANNAME	CORRNMBR		TCB CPU TIME	ID			
DB2R6	TSO	CBECC1910BEF	DB2R6	TSO			DB2R6
DB2R6	DB2R6	TSO	19:12:03.69842527	174979	1 370	OPEN DATA SET	NETWORKID: USIBMSC LUNAME: SCPDB1A LUWSEQ: 1
DSNESPSC	'BLANK'	N/P	N/P			INFORMATION	

DATABASE OPEN INFORMATION							
DATA SET NAME	:	DB1AD.DSNDBC.DB2R6DB.DB2R6TS.I0001.A001				FLAGS	: X'5F'
ACE ADDRESS	:	X'1FBC5DE0'		DATABASE ID	: 401	OBID	: 2
PART NUMBER	:	X'00000001'		INSTANCE NUMBER	:	X'00000001'	DSMAX : 20000
OPENED DATA SETS	:	403		ALLOCATION TIME	:	0.002319	OPEN TIME : 0.006813

SYSOPR	DB1A	CBECC19A13EA	N/P	N/P			N/P
SYSOPR	020.STOP	'BLANK'	19:12:13.14262194	174982	2 371	CLOSE DATA SET	NETWORKID: USIBMSC LUNAME: SCPDB1A LUWSEQ: 1
'BLANK'	DB09	N/P	N/P			INFORMATION	

DATABASE CLOSE INFORMATION							
DATA SET NAME	:	DB1AD.DSNDBC.DB2R6DB.DB2R6TS.I0001.A001				FLAGS	: X'00'
ACE ADDRESS	:	X'1FBC9760'		DATABASE ID	: 401	OBID	: 2
PART NUMBER	:	X'00000001'		INSTANCE NUMBER	:	X'00000001'	DSMAX : 20000
OPENED DATA SETS	:	404		DEALLOC TIME	:	0.000467	CLOSE TIME : 0.010327



Log activity

DB2 logs changes made to data by the applications and other significant subsystem events, as they occur, to provide data recoverability. The characteristics of the workload have a direct effect on log write performance. Long-running tasks that commit infrequently incur a lot more data to write at commit than a typical transaction. These tasks can cause subsystem impact because of the excess storage consumption, locking contention, resources that may be consumed for a rollback, and longer commit times.

The cost of reading the log directly affects how long a restart or a recovery takes because DB2 must read the log data before applying the log records back to the table space.

In this chapter, we introduce logging, then provide considerations on monitoring log activity.

This chapter contains the following topics:

- ▶ Log activity
- ▶ Log write activity
- ▶ Log read activity
- ▶ Log I/O tuning possibilities

7.1 Log activity

DB2 records in two logcopy data sets all data changes and other significant events in a log. If you keep these logs, DB2 can re-create the changes for you in the event of a failure or rollback the changes to a previous point in time. Most of the log records describe changes to the DB2 databases and are made within units of recovery.

Because logging requires I/O to provide recoverability of the data, sometimes logging can impact your application performance or cause a system bottleneck.

The main system parameter you need to consider related to logging is the log buffer size; see Table 7-1.

Table 7-1 Parameter for logging

Parameter	Description
OUTBUFF	The size of the output buffer that is to be used for writing active log data sets.

In DB2 11, log buffers are moved to 64-bit HCSA in the DBM1 address space, avoiding address space switching during writing log records. Log buffers are page fixed and use 1 MB page frames if available; otherwise, 4 KB page frames are used.

7.2 Log write activity

DB2 writes a log to the active log in unit-of-recovery events such as the end of commit phase 1, or at the begin of commit phase 2, in case the transaction has a two phase commit. In a single phase commit, phase 1 and phase 2 are combined into a single phase, where it only writes one log write for one phase commit.

DB2 is designed to use *write-ahead* logging protocol, which means all the log records with changes must be written out to DASDs before the updated page is written to DASD or the coupling facility.

When we have page P-lock negotiation and index leaf page split in data sharing, before the negotiation is complete, and the requester can actually hold the P-lock, DB2 must actually do forced writes out of the log.

This is done to expose the update to the other members of the data sharing group. There are two forced physical log writes per index leaf page split in DB2 10. The number of forced log writes per index split is reduced to one in DB2 11. When this happens, DB2 forces the log up to what is called the PGLOGRBA, that is the RBA or LRSN of the last update to the page.

DB2 also writes to the active log when taking system checkpoints that occur based on time or number of log records since the last previous system checkpoint. DB2 also takes system checkpoints during active log switch.

When DB2 has not written any log records for a certain period of time or number of buffers, DB2 forces the log records out to the active log. Prior to V7, there was a log write threshold externalized with WRTHRSH DSNZPARM. With DB2 V8 or later, a value for this log write threshold was fixed to 20 buffers. So if there are no log CIs written out to the active log for more than 20 buffers, those log records are written to the active log. Log records can also be forced out using an archive log command or an IFI read request for IFICID 306 from another member in the data sharing group.

When the log writes actually occur, DB2 can write up to 128 4 KB log CIs per write I/O. When you have dual logging, DB2 sometimes will do the write serially and sometimes in parallel:

- ▶ If DB2 is writing CI for the first time, then the writes to the primary and secondary copy of the log are fully overlapped.
- ▶ If DB2 is re-writing a log CI that is partially filled and written previously, then those CIs are written serially.

What this means is that, if you want to get fully parallel overlap of writes to log copy 1 or log copy 2, you have to have at least 4 log CIs to write. That is because the first log CI is a rewrite of one that was previously there. So that has to be done serially. Then CIs 2 and 3 can be written in parallel. And then we have the fourth CI and then go back to rewrite to log copy 1 and log copy 2 being serial.

DB2 10, or later, simply writes all four pages to log copy 1 and log copy 2 in parallel. Hence, log writes for these pages only have two I/Os, and DB2 waits for the duration of only one I/O (that is, whichever of the two I/Os takes the longer.)

The log output buffer, which is allocated in the master address space, is controlled by an OUTBUFF DSNZPARM. When the log output buffer becomes unavailable, all insert, update, and delete activity and other system activities are going to be stopped until the condition is relieved. So you need to make sure you have enough log output buffer assigned to deal with increasing logging volumes, or assign bigger log output buffer when that unavailable condition occurs.

There is an OMEGAMON PE statistics report field for monitoring page-in for the log output buffer, but starting from DB2 10, all the log output buffer is page fixed in memory. When DB2 starts, all the buffers as specified by OUTBUFF are going to be allocated and page fixed. It is necessary to carefully decide the OUTBUFF value to avoid assigning more storage than necessary when your system's storage consumption is high. If you need to calculate the log data volume, from the OMEGAMON PE statistics report, take the number of log CIs created per unit of time multiplied by 4 K, which is the total size of the log CI, then divide by the statistics interval. The log data volume should be considered against device and channels.

Tip: Consider enlarging log buffer (OUTBUFF) when you monitor log buffer unavailable ('UNAVAILABLE OUTPUT LOG BUFF') and log data volume is much less than maximum supported. The log data volume can be calculated as follows:

Log data volume per unit of time = 'LOG CI CREATED' * 4KB / stats_interval

Example 7-1 shows the OMEGAMON PE statistics reports for monitoring log write activities.

Example 7-1 Log write statistics

LOG ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
READS SATISFIED-OUTPUT BUFF	1.00	0.00	0.20	0.04
READS SATISFIED-OUTP.BUF(%)	100.00			
READS SATISFIED-ACTIVE LOG	0.00	0.00	0.00	0.00
READS SATISFIED-ACTV.LOG(%)	0.00			
READS SATISFIED-ARCHIVE LOG	0.00	0.00	0.00	0.00
READS SATISFIED-ARCH.LOG(%)	0.00			
TAPE VOLUME CONTENTION WAIT	0.00	0.00	0.00	0.00
READ DELAYED-UNAVAIL.RESOUR	0.00	0.00	0.00	0.00
ARCHIVE LOG READ ALLOCATION	0.00	0.00	0.00	0.00
ARCHIVE LOG WRITE ALLOCAT.	8.00	0.01	1.60	0.33
CONTR.INTERV.OFFLOADED-ARCH	72000.00	79.98	14.4K	3000.00
LOOK-AHEAD MOUNT ATTEMPTED	0.00	0.00	0.00	0.00
LOOK-AHEAD MOUNT SUCCESSFUL	0.00	0.00	0.00	0.00
UNAVAILABLE OUTPUT LOG BUFF	0.00	0.00	0.00	0.00
OUTPUT LOG BUFFER PAGED IN	0.00	0.00	0.00	0.00
LOG RECORDS CREATED	710.9K	789.63	142.2K	29.6K
LOG CI CREATED	62252.00	69.15	12.5K	2593.83
LOG WRITE I/O REQ (LOG1&2)	1632.00	1.81	326.40	68.00
LOG CI WRITTEN (LOG1&2)	124.5K	138.31	24.9K	5188.33
LOG RATE FOR 1 LOG (MB)	N/A	0.27	N/A	N/A

7.3 Log read activity

Log read activity is driven by a different type of activity of DB2:

- ▶ There is a unit of work related activity such as rollback of the failing transaction.
- ▶ When recovering the data using the RECOVER utility, the utility applies all changed logs forward after restoring the latest image copy, or the BACKOUT option backs out changes starting from the currently operational page set.
- ▶ During RESTART, you might issue the **START DATABASE** command, which reads a log for LPL and GRECP recovery.
- ▶ If you are using online REORG and making some updates while reorganizing table space or partitions, the log phase of the utility reads logs to apply changes done during REORG.
- ▶ Also, an application or a monitoring tool can use the IFI log read interface, such a case being data replication. Replication uses IFCID 129 to read a range of log records CIs from the active log and/or IFCID 306 to support reads from the archive logs, and decompression.
- ▶ Log can be also read using the stand-alone log read service, DSNJSLR, which is also used by the stand-alone utility, DSN1LOGP.

When the log reads are requested, DB2 first looks for needed log records from the log output buffer, and if they are no longer available in the log output buffer, then DB2 searches for the active log data sets. And if the log records are no longer available in the active log data sets, then DB2 reads those log records from the archive log data sets.

The OMEGAMON PE statistics report shows how many times reads were satisfied from the log output buffer, the active logs, or the archive logs. Example 7.2 shows an example for log read, with almost no log read from log buffer (0.6%), with 27% log read from the active logs, and with 72% log read from the archive logs. In this example, there are many log read from archive logs. If the archives have been migrated away to tape, then reading the log records from the archive logs can impact to the elapsed time.

Example 7-2 Log activity in statistics report

LOG ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
READS SATISFIED-OUTPUT BUFF	10615.00	1.68	624.41	72.71
READS SATISFIED-OUTP.BUF(%)	0.60			
READS SATISFIED-ACTIVE LOG	480.6K	76.28	28.3K	3291.83
READS SATISFIED-ACTV.LOG(%)	27.00			
READS SATISFIED-ARCHIVE LOG	1289.1K	204.60	75.8K	8829.60
READS SATISFIED-ARCH.LOG(%)	72.41			
TAPE VOLUME CONTENTION WAIT	0.00	0.00	0.00	0.00
READ DELAYED-UNAVAIL.RESOUR	0.00	0.00	0.00	0.00
ARCHIVE LOG READ ALLOCATION	7.00	0.00	0.41	0.05
ARCHIVE LOG WRITE ALLOCAT.	28.00	0.00	1.65	0.19
CONTR.INTERV.OFFLOADED-ARCH	252.0K	39.99	14.8K	1726.03
LOOK-AHEAD MOUNT ATTEMPTED	0.00	0.00	0.00	0.00
LOOK-AHEAD MOUNT SUCCESSFUL	0.00	0.00	0.00	0.00
UNAVAILABLE OUTPUT LOG BUFF	0.00	0.00	0.00	0.00
OUTPUT LOG BUFFER PAGED IN	0.00	0.00	0.00	0.00
LOG RECORDS CREATED	2844.0K	451.37	167.3K	19.5K
LOG CI CREATED	249.1K	39.53	14.7K	1706.07
LOG WRITE I/O REQ (LOG1&2)	6740.00	1.07	396.47	46.16
LOG CI WRITTEN (LOG1&2)	498.6K	79.13	29.3K	3414.99
LOG RATE FOR 1 LOG (MB)	N/A	0.15	N/A	N/A

If you are impacted by needing to access archive logs, adding more space to the output log buffer may help log read performance and reduce the number of reads from the active or archive logs.

When it comes to reading the log data records, then reading from the active log performs better because DB2 can use prefetch when reading the log CIs. DB2 can also use automatic I/O load balancing across copy 1 and copy 2 of the active log pair. VSAM striping can also be enabled for the active log pair to improve bandwidth.

When reading archive log is required, then archives on DASD perform better than archives on tape. The use of tape destroys parallelism. So if you have multiple recover jobs, which all require the log records already written out to archives on tape, then all the recovery jobs end up serially reading those log records. This also applies to use of virtual tape devices. A virtual tape device still appears to the operating system as a tape device. Also, tape requires serialization in data sharing across concurrent recoveries on different members.

7.4 Log I/O tuning possibilities

When you consider log I/O, try to avoid I/O interference between active logs, or between multiple log reads/writes processes. In other words, you should minimize the interference between active log write and active log archive process.

There are several options to take into consideration when the logging rate is reaching the maximum that your system environment can support:

- ▶ Moving the log data set to a faster device
- ▶ Using DFSMS and striping
- ▶ Reducing the log data volume

The log data reduction can be possible; if you have insert intensive tables having a long record length which has not utilized compression, then you might want to consider using DB2 data compression. DB2 data compression not only reduces the size of the row, but also reduces the size of the log records created. Use DNS1COMP to determine how well your candidate table might compress.

If you are on DB2 9 or later and are still using the basic row format, you might have the possibility to optimize the table design to minimize the log record size with variable length records.

Also, be aware of the database descriptors (DBDs) update if you are making a lot of DDL updates against large databases having a large DBD with many objects defined. Tuning against DBDs means tuning the DDL itself, such as running multiple DDL for the same database or DBD in the same unit of work, or splitting a large database into smaller databases, which in turn makes each of these databases have smaller DBDs.



IRLM, locking, and latching

Locks and latches are used respectively by the application and by DB2 to serialize events and maintain data integrity. The IRLM subsystem manages DB2 locks. You can monitor the use of locks and latches to improve concurrency and prevent problems such as contention, suspensions, timeouts, or deadlocks.

This chapter contains the following topics:

- ▶ DB2 and the internal resource lock manager (IRLM)
- ▶ DSNZPARMs related to locking
- ▶ Lock avoidance
- ▶ Data sharing locking
- ▶ Internal DB2 latch contention

8.1 DB2 and the internal resource lock manager (IRLM)

The internal resource lock manager (IRLM) is both a separate address space and an integral component of DB2.

IRLM subsystem manages DB2 locks. There is one IRLM for each DB2 as specified in the DB2 load module for subsystem parameters. The IRLM is also identified as a z/OS subsystem in the SYS1.PARMLIB member IEFSSNxx. That name is used as the IRLM procedure name (irlmproc) in z/OS commands.

DB2 uses the IRLM to lock various resources in order to control access to owned resources. Locking can be explicit and implicit to the user. Explicit locking allows the users to lock specific table spaces, partitions, or tables by commands or SQL. Implicit locking for database descriptors (DBD), table spaces, partitions, or tables is generally performed during the resource allocation process. IRLM also locks pages or rows in table spaces. The locks prohibit more than one application process from updating the same data at the same time.

For data sharing, DB2 ensures that all members of the data sharing group see the current version of a changed page by forcing the pages out at a commit or as a result of p-lock negotiations.

The IRLM address space always needs to be assigned to the highest service class in MVS dispatching priorities; usually SYSSTC. This is because of the importance of detecting and breaking deadlocks, for error management, and for lock resumption. IRLM needs this priority because it manages resources that are managed by work within the DB2 address spaces.

In V8 and later, DB2 supports 64-bit IRLM support and PC=YES is forced, which means that the locks, including the LOB locks and locks against other data, are held above the 2 GB bar. This has reduced the storage requirement for the extended common services area.

8.1.1 IRLM startup procedure options

You can control how DB2 uses locks by specifying certain options when you start the internal resource lock manager (IRLM).

When you issue the z/OS **START irlmproc** command, the values of the options are passed to the startup procedure for the DB2 IRLM. These options are relevant to DB2 locking:

SCOPE	Whether IRLM is used for data sharing (GLOBAL) or not (LOCAL). Use LOCAL unless you are using data sharing. If you use data sharing, specify GLOBAL.
DEADLOK	The two values of this option specify: The number of seconds between two successive scans for a local deadlock. The number of local scans that occur before a scan for global deadlock starts.
LTE	Maximum size of CF lock structure table entries (LTE) up to a maximum value of 2 GB.
PC	Ignored by IRLM. However, PC is positional and must be maintained in the IRLM for compatibility.
MAXCSA	Ignored by IRLM. However, MAXCSA is positional and must be maintained in the IRLM for compatibility.

The maximum amount of storage available for IRLM locks is limited to 90% of the total space given to the IRLM private address space during the startup procedure. The other 10% is reserved for IRLM system services, z/OS system services, and “must complete” processes to prevent the IRLM address space from abending, which would bring down your DB2 system. When the storage limit is reached, lock requests are rejected with an out-of-storage reason code.

Over time, the default values for timeout (IRLMRWT) and deadlock detection (DEADLOK) frequency had been shortened with the timeout period becoming 30 seconds and deadlock detection time of 1 second.

Table 8-1 gives an overview of allowable range and default values as of today. Commonly suggested those values are to be reduced from defaults, such as in a high-volume OLTP environment where performance and availability is important. If your setting is based on old defaults, you might want to have those values get revised to detect and break the deadlocks as fast as possible. Reducing the deadlock detection frequency prioritize availability over performance. By bringing down deadlock detection frequency, it may end up increasing number of IRLM latch contention for some environments. See 8.3.1, “IRLM latch contention” on page 97 for details on checking how your settings might affect your environment.

Table 8-1 Timeout and deadlock detection frequency

	IRLMRWT	DEADLOK
Range allowed	1 to 3600 sec	0.1 to 5 sec
DB2 9 defaults	60 sec	1 sec
DB2 10 and 11 defaults	30 sec	1 sec

8.1.2 z/OS commands on IRLM

You can use various z/OS commands to modify and monitor the IRLM connection, as summarized in Table 8-2.

Table 8-2 z/OS commands for IRLM operations

Command	Description
MODIFY irImproc,SET,PVT=nnn	Sets the maximum amount of private virtual (PVT) storage that this IRLM can use for lock control structures.
MODIFY irImproc,SET,DEADLOCK=nnnn	Sets the time for the local deadlock detection cycle.
MODIFY irImproc,SET,LTE=nnnn	Sets the number of LOCK HASH entries that this IRLM can use on the next connect to the XCF LOCK structure. Use this command only for data sharing.
MODIFY irImproc,SET,TIMEOUT=nnnn, subsystem-name	Sets the timeout value for the specified DB2 subsystem. Displays the subsystem-name by using MODIFY irImproc,STATUS .
MODIFY irImproc,SET,TRACE=nnn	Sets the maximum number of trace buffers that are used for this IRLM.
MODIFY irImproc,STATUS,irImnn	Displays the status of a specific IRLM.
MODIFY irImproc,STATUS,ALLD	Displays the status of all subsystems known to this IRLM in the data sharing group.
MODIFY irImproc,STATUS,ALLI	Displays the status of all IRLMs known to this IRLM in the data sharing group.

Command	Description
MODIFY irIproc,STATUS,MAINT	Displays the maintenance levels of IRLM load module CSECTs for the specified IRLM instance.
MODIFY irIproc,STATUS,STOR	Displays the current and high-water allocation for private virtual (PVT) storage, as well as storage that is above the 2 GB bar.
MODIFY irIproc,STATUS,TRACE	Displays information about trace types of IRLM subcomponents.

Use the z/OS command **MODIFY irIproc,STATUS,STOR** to see how much storage IRLM is using and the **MODIFY irIproc,SET** command to dynamically change the maximum amount of IRLM private storage to use for locks. See Example 8-1.

Example 8-1 Modify irIproc,STATUS,STOR

```

F DB1AIRLM,STATUS,STOR
DXR100I ID1A001 STOR STATS 534
PC: YES  LTEW:n/a LTE:      M RLE:      RLEUSE:
BB PVT: 1314M AB PVT (MEMLIMIT): 2160M
CSA USE: ACNT:      OK AHWM:      OK CUR:  518K HWM:  518K
        ABOVE 16M:  20  518K  BELOW 16M:  0    OK
PVT USE: BB CUR: 4393K AB CUR:  5M
CLASS  TYPE SEGS  MEM  TYPE SEGS  MEM  TYPE SEGS  MEM
ACCNT   T-1   2    4M   T-2   1    1M   T-3   1    4K
PROC    WRK   5    25K  SRB   1    1K   OTH   1    1K
MISC    VAR  12   5546K N-V   12   323K  FIX   1    24K
DXR100I End of display

```

The timeout and deadlock detection frequency can be dynamically changed via the **MODIFY irIproc,SET,DEADLOCK=** or **TIMEOUT=** command. Example 8-2 provides an example of dynamically changing and showing current timeout and deadlock frequency values using the commands.

Example 8-2 Dynamically changing deadlock or timeout frequency

```

F DB1AIRLM,SET,DEADLOCK=500
DXR177I ID1A001 THE VALUE FOR DEADLOCK IS SET TO      500  MILLISECOND

F DB1AIRLM,SET,TIMEOUT=20,DB1A
DXR177I ID1A001 THE VALUE FOR TIMEOUT  IS SET TO      20  FOR DB1A

F DB1AIRLM,STATUS
DXR101I ID1A001 STATUS SCOPE=LOCAL 928
        DEADLOCK: 0500
SUBSYSTEMS IDENTIFIED
NAME    T/OUT  STATUS    UNITS    HELD    WAITING  RET_LKS
DB1A    0020   UP        1        1        0        0
DXR101I End of display

```

8.1.3 Accounting and tracing IRLM

For IRLM storage accounting, see the statistics report as shown in Example 8-3.

Example 8-3 IRLM storage accounting block from statistics report

IRLM STORAGE BELOW AND ABOVE 2 GB		QUANTITY
-----	-----	-----
EXTENDED CSA SIZE IN USE	(MB)	5.15
HWM EXTENDED CSA SIZE IN USE	(MB)	10.00
31 BIT PRIVATE IN USE	(MB)	0.00
HWM 31 BIT PRIVATE IN USE	(MB)	0.00
THRESHOLD 31 BIT PRIVATE	(MB)	0.00
64 BIT PRIVATE IN USE	(MB)	0.00
HWM 64 BIT PRIVATE IN USE	(MB)	0.00
THRESHOLD 64 BIT PRIVATE	(MB)	0.00
64 BIT COMMON IN USE	(MB)	0.00
HWM 64 BIT COMMON IN USE	(MB)	0.00

There are occasions where you want to take the IRLM trace. You should be aware that the IRLM trace has a very high CPU overhead of up to 25 percent, and it could also increase IRLM latch contention. The overhead is not on the transaction, but on the individual lock request. So for example, if the application is not making many lock requests because they are taking good advantage of lock avoidance, the overhead can be small. On other hand, if the application is a lock intensive application, having overhead against each lock request can be significant.

8.2 DSNZPARMs related to locking

Table 8-3 shows the system parameters related to locking. Some need to be monitored and be revised because they can change the behavior of the applications. Be sure you review how DB2 behaves and causes impact on current applications before changing those parameters. For more information, see *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725.

Table 8-3 IRLM and locking related parameters

Parameter	Description
MEMLIMIT	The maximum amount of private storage above the 2 GB bar for the IRLM lock control block structure.
DEADLOCK	IRLM PROC parameter that controls the amount of time for which local deadlock detection cycles are to run.
IRLMRWT	DB2 DSNPARM that controls the number of seconds that are to elapse before a resource timeout is detected.
NUMLKTS	Default maximum number of page, row, or LOB locks that an application can hold simultaneously in a table or table space.

Parameter	Description
NUMLKUS	The maximum number of page, row, or LOB locks that a single application can hold concurrently for all table spaces.
RRULOCK	U-lock on SELECT FOR UPDATE for ISOLATION(RRIRS).
XLKUPDLT	X-lock for searched UPDATE/DELETE.
SKIPUNCI	Skips uncommitted inserts for ISOLATION(CSIRS).
EVALUNC	Controls whether predicate evaluation is to be allowed on uncommitted data of other transactions.

8.3 Lock avoidance

Taking a lock carries a cost for both concurrency and CPU. When certain conditions are met and DB2 can guarantee the consistency of the data, DB2 uses a combination of techniques to retrieve data without requiring to hold a shared lock on behalf of the application process. This function, which only applies to low level locks, is known as *lock avoidance*. The application process has no direct control over when lock avoidance occurs.

Reasons for taking read locks are to make sure that uncommitted data is not retrieved by the application, so no other applications can make changes to the data while reading it. In order to read the consistent data without taking locks, DB2 must be certain that the data the application is about to read is committed.

There are several benefits of having lock avoidance, such as increasing concurrency, a decrease in the number of locks and unlocks, which results reducing CPU consumptions as well as data sharing overheads. It is particularly important for a data sharing environment to decrease the number of lock and unlock activity requests. So, the value of lock avoidance is much more important in a data sharing environment where your object is group buffer pool dependent.

For understanding the techniques that DB2 uses to determine when lock avoidance is to be applied, is important to understand how lock avoidance can be eligible and how effective lock avoidance is. We start with the definition of these techniques:

- ▶ Page latching and P-locks in data sharing:

This technique, page latching and page physical lock or page P-Locks, is used in a data sharing environment for inter-DB2 consistency. This technique is used by DB2 to ensure the physical consistency of an individual index or data page.

- ▶ Commit log sequence number (CLSN):

This technique is used by DB2 to guarantee the logical consistency of the page, so DB2 knows that the update on that particular page had been committed. DB2 uses a two part technique for CLSN; the first is to track the last updates against an individual page, and the second is to keep track of the oldest uncommitted activity at an individual page set partition. For a page set partition, non-data sharing and data sharing with non-GBP-dependent page set partition use CLSN, and data sharing with GBP-dependent data sets uses a global CLSN value across all members. DB2 compares the value in the page with the CLSN value at a global level or at the data set level; if the value in the page is lower DB2 knows that the page is committed.

- Possible uncommitted bits (PUNC bits):

This technique keeps track of each row in the data page or index page being updated, by setting a PUNC bit when the data is updated. When the PUNC bit is not set, then the row or key is guaranteed to be committed. The PUNC bit is periodically reset when DB2 gets a successful CLSN and more than 25 percent of the rows having the PUNC bit being set, by scanning the data with a query running under ISOLATION RR, or by running REORG against table spaces or indexes.

The application prepared with the following bind options or condition is going to be eligible for lock avoidance:

- Read-only or ambiguous cursor with ISOLATION(CS) and CURRENTDATA(NO)
- Non-cursor or singleton-select with ISOLATION(CS), regardless of CURRENTDATA
- For any nonqualifying rows accessed by queries bound with ISOLATION CS or RS
- When DB2 system managed referential integrity (RI) checks for dependent rows when either the parent key is updated

Lock avoidance monitoring and tuning

Basically locks are triggered by the application but it is of benefit monitoring the locking activity at DB2 subsystem level using the statistics trace. If lock avoidance is effectively working with most of your applications, you should see low numbers in the lock and unlock requests.

If your application is not effectively using lock avoidance, you should check if the applications are eligible for lock avoidance. If the application is running at ISOLATION RS or ISOLATION RR and your application needs are granted, you should consider changing your ISOLATION to CS to gain the benefit of lock avoidance.

Even if your application is eligible for lock avoidance, there are cases where your application may not gain the benefit of lock avoidance. There can be other reasons for not being able to apply lock avoidance.

The database having many pointer records is one case where updates against compressed or variable length records results in relocation, which may result in getting many lock and unlock requests to those pointer records.

Another case is where pseudo-deleted entries exist in unique indexes and needing to insert a key to the index, DB2 needs to lock and unlock for those data. When the SQL query scans a pseudo deleted index entry, it needs to know if the delete transaction that created the pseudo deleted entry is committed. It will first try to use commit LRSN checking, but if that fails, it needs to get an S lock on the pseudo deleted RID. If the lock can be acquired, that means the delete transaction has committed, so the pseudo deleted entry will be skipped. If the lock is not acquired, the SQL has to wait for the lock, which could cause a deadlock situation.

These cases can both be eliminated by reorganizing those table spaces and/or indexes.

For the first case, DB2 11 has added special free space for updates to reduce relocation with the new PCTFREE_UPD subsystem parameter.

For the latter case, DB2 11 has added a new capability of asynchronously removing pseudo-empty indexes pages and pseudo-deleted index entries by DB2 system tasks independently from SQL DELETE transaction. This function reduces the need for frequent REORG INDEX.

To monitor your lock efficiency, look at the lock and unlock requests from locking activity. See Example 8-4.

Tip: Lock avoidance may not be working effectively if unlock requests/commit is high, meaning that you may have opportunity to gain benefit by lock avoidance. such as unlock requests/commit > 5.

Example 8-4 Locking activity

LOCKING ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
SUSPENSIONS (ALL)	466.00	0.16	51.78	0.01
SUSPENSIONS (LOCK ONLY)	0.00	0.00	0.00	0.00
SUSPENSIONS (IRLM LATCH)	466.00	0.16	51.78	0.01
SUSPENSIONS (OTHER)	0.00	0.00	0.00	0.00
TIMEOUTS	0.00	0.00	0.00	0.00
DEADLOCKS	0.00	0.00	0.00	0.00
LOCK REQUESTS	763.8K	259.78	84.9K	11.30
UNLOCK REQUESTS	194.5K	66.16	21.6K	2.88
QUERY REQUESTS	0.00	0.00	0.00	0.00
CHANGE REQUESTS	92.00	0.03	10.22	0.00
OTHER REQUESTS	0.00	0.00	0.00	0.00
LOCK ESCALATION (SHARED)	0.00	0.00	0.00	0.00
LOCK ESCALATION (EXCLUSIVE)	0.00	0.00	0.00	0.00
DRAIN REQUESTS	12.00	0.00	1.33	0.00
DRAIN REQUESTS FAILED	0.00	0.00	0.00	0.00
CLAIM REQUESTS	62478.00	21.25	6942.00	0.92
CLAIM REQUESTS FAILED	0.00	0.00	0.00	0.00

As already mentioned, having lock avoidance is particularly important to the data sharing environment. If there is a long running unit-of-recovery sitting in the system, this can be a cause of stopping global CLSN values forward for GBP dependent objects, which leads to not able to use lock avoidance. To effectively use lock avoidance, consider aggressively monitoring long-running unit-of-recovery using the following methods, and make those applications fixed to commit frequently based on elapsed time and/or CPU time.

The first method is to detect a long running unit-of-recovery by setting URCHKTH DSNZPARM with the number of checkpoint cycles that are to complete before commit. DB2 issues a warning message to the console when a unit-of-recovery exists after a specified number of checkpoints. In DB2 10 and later, the default value had been changed to 5, which is a suggested starting point for monitoring a long running unit-of-recovery. Consider tuning those applications being detected by appropriately issuing commit. Example 8-5 shows a DSNR035I message written to the console.

Example 8-5 DSNR035I detecting long running UR, URCHKTH

```

DSNR035I  -DB1A DSNRPBCW WARNING - UNCOMMITTED UR 394
AFTER 5 CHECKPOINTS -
CORRELATION NAME = DB2R6
CONNECTION ID   = TSO
LUWID = USIBMSC.SCPDB1A.CBE2D3BE40B8 = 728
PLAN NAME = DSNESPCS

```

```
AUTHID = DB2R6
END USER ID = DB2R6
TRANSACTION NAME = DB2R6
WORKSTATION NAME = TSO
```

Another method is using URLGWITH DSNZPARM, with which DB2 can detect applications that are doing large amounts of update without issuing commit. In DB2 10 and later, the default value had been changed to 10K, which is a suggested starting point for monitoring a mass update unit-of-recovery. Example 8-6 shows a DSNJ031I message written to the console. DSNJ031I will be issued each time the number of log records exceeds the URLGWITH value from the last DSNJ031, so there were DSNJ031I messages written 34 times before getting that message when URLGWITH is being set to 10K.

Example 8-6 DSNJ031I detecting large updates with out commit, URLGWITH

```
DSNJ031I  -DB1A DSNJW011 WARNING - UNCOMMITTED UR 884
HAS WRITTEN 350000 LOG RECORDS -
CORRELATION NAME = DB2R6
CONNECTION ID   = TSO
LUWID = USIBMSC.SCPDB1A.CBF47F3724D6 = 1994
PLAN NAME = DSNESPCS
AUTHID = DB2R6
END USER ID = DB2R6
TRANSACTION NAME = DB2R6
WORKSTATION NAME = TSO
```

8.3.1 IRLM latch contention

IRLM latch contentions are suspensions happening during serialization of resources within IRLM using latches. Trace fields are counted when a transaction has to wait for an IRLM latch. High IRLM latch contention can be a cause of a high IRLM CPU time. A high number of IRLM latch contention could be caused by various reasons. Some of the known causes of high IRLM latch contention are as follows:

- ▶ The IRLM trace is turned on. Make sure the IRLM trace is not permanently turned on.
- ▶ The IRLM is not running with proper dispatching priority in the system. The IRLM address space should be higher priority than all other DB2 address spaces, the WLM service class SYSSTC should be assigned to the IRLM.
- ▶ Frequent IRLM query requests as a result of issuing a command such as **DISPLAY DATABASE LOCKS** or **MODIFY ir1mproc,STATUS** can result in high IRLM latch contention.
- ▶ The deadlock detection cycle is very low and the locking rate is very high, because the IRLM main latch is used during deadlock detection. If IRLM has to check for deadlocks frequently, such as less than a second, and the number of locks are very high, the main latch is held for a longer time and IRLM latch contention is more likely to occur. If this can be the cause of a high IRLM latch contention, investigate if the application can apply lock avoidance to reduce the lock requests rates; consider reducing the deadlock detection cycle if you cannot reduce lock request rates.
- ▶ The IRLM latch contention can also increase when many RELEASE(DEALLOCATE) packages are used. Since they keep their page set level locks around for a longer time than RELEASE(COMMIT), this can increase the total number of concurrent locks to be managed by IRLM, making longer resource chains to process, which increases the time an IRLM latch is held and can cause contention with another requester during that time. IRLM APAR PM83690 gives enhancement for batch unlock processing that reduces IRLM latch invocation. DB2 11 also makes RELEASE(DEALLOCATE) optimization at commit.

Tip: The IRLM latch contention is considered high if the number of IRLM latch contentions is more than 1% to 5% of the total number of IRLM lock requests, where IRLM lock requests are a total of lock, unlock, query, and change requests:

IRLM latch contention rate (%) = ('SUSPENSIONS (IRLM LATCH)' / ('LOCK REQUESTS' + 'UNLOCK REQUESTS' + 'QUERY REQUESTS' + 'CHANGE REQUESTS')) * 100

Example 8-7 shows an example of locking activity from a statistics trace; the IRLM latch contention for this example shows less than 1%, which means that DB2 is working fine.

Example 8-7 Locking activity block

LOCKING ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
SUSPENSIONS (ALL)	6.00	0.00	0.24	0.03
SUSPENSIONS (LOCK ONLY)	0.00	0.00	0.00	0.00
SUSPENSIONS (IRLM LATCH)	6.00	0.00	0.24	0.03
SUSPENSIONS (OTHER)	0.00	0.00	0.00	0.00
TIMEOUTS	0.00	0.00	0.00	0.00
DEADLOCKS	0.00	0.00	0.00	0.00
LOCK REQUESTS	4601.00	0.67	184.04	26.60
UNLOCK REQUESTS	1836.00	0.27	73.44	10.61
QUERY REQUESTS	0.00	0.00	0.00	0.00
CHANGE REQUESTS	190.00	0.03	7.60	1.10
OTHER REQUESTS	0.00	0.00	0.00	0.00
LOCK ESCALATION (SHARED)	0.00	0.00	0.00	0.00
LOCK ESCALATION (EXCLUSIVE)	0.00	0.00	0.00	0.00
DRAIN REQUESTS	24.00	0.00	0.96	0.14
DRAIN REQUESTS FAILED	0.00	0.00	0.00	0.00
CLAIM REQUESTS	1391.00	0.20	55.64	8.04
CLAIM REQUESTS FAILED	0.00	0.00	0.00	0.00

8.4 Data sharing locking

A data sharing lock or a global lock provides serialization across the data sharing group, where a DB2 data sharing member is made known to other members in the data sharing group. Unlike a single DB2 system where everything can be tracked by itself, DB2 needs to keep track of a data shared between two or more DB2 subsystems. DB2 uses System z IBM Parallel Sysplex® technology to implement a data sharing locking mechanism, which consists of several components including z/OS components, thus more considerations for monitoring data sharing environment.

Figure 8-1 shows the z/OS and DB2 system main components used to keep track of global locking in a DB2 data sharing environment.

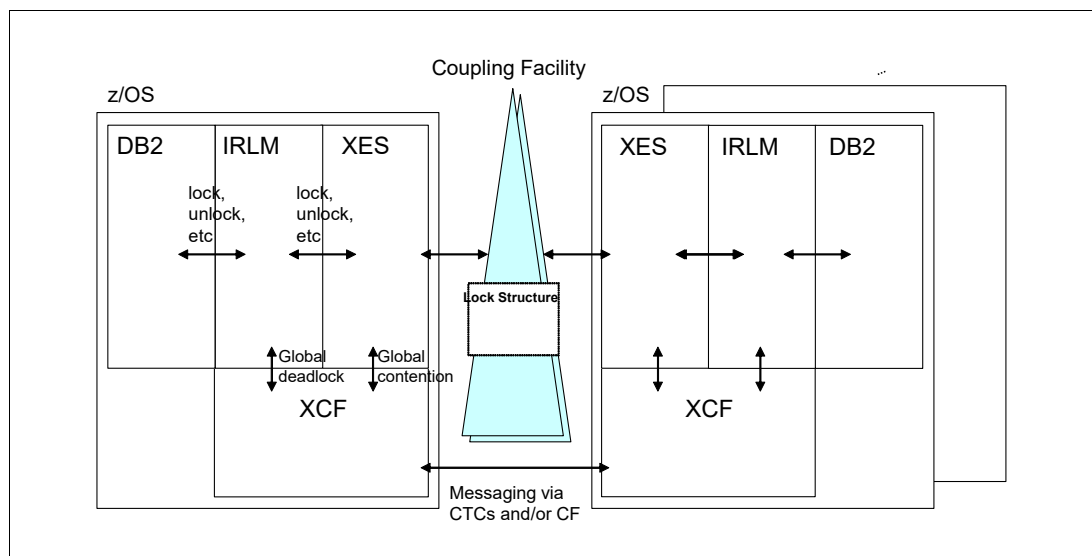


Figure 8-1 DB2 data sharing global locking system components

8.4.1 Global contentions

In a DB2 data sharing environment, contention can occur when different members need to get hold of the same resource at the same time in a state that is not compatible. When the lock holder is on a different DB2 member while lock requests are being made, this is called global lock contention.

Global lock contention in a data sharing environment comes mainly in three types. They are listed in increasing order of the importance to resolve them:

- ▶ *False contention* occurs when the hashing algorithm provides the same hash value on a lock table anchor point for two different resources. The lock structure and the number of lock hash entries can be increased to minimize the false contention.
- ▶ *XES contention* occurs because XES only has two lock modes, share and exclusive (S and X), where IRLM locking supports many more additional lock modes (S, X, U, IX and so on). Therefore, it can appear to XES that contention exists, but when IRLM looks at the lock states, they can be compatible. V8 and later, "locking protocol 2" had been introduced, changing mapping of XES to IRLM locks, which reduced the common cause of XES contention between parent locks. In general, locking protocol 2 has contributed to reduce XES contention. However, there are some exceptions that may result in more lock requests, such as applications that start a database or table space in read only mode, use SQL LOCK TABLE, use LOCKSIZE TABLESPACE, or table space scan with repeatable read.
- ▶ *Real contention* occurs when lock requests are incompatible, for example, one program holds an S L-lock on a page while another program requests an X L-lock on the same page. When real contention occurs, the requester remains suspended until the incompatible lock is released or the program abends because of timeout or deadlock. The application tuning is needed to reduce the contention, such as making frequent commit, access path tuning, and so on.

It is important to keep track of number of global contentions within a data sharing environment, since they are the biggest constraint for the scalability and performance. The OMEGAMON PE statistics reports give the following two fields that calculate global contention and false contention rates. If multiple members from the same data sharing group run on the same LPAR, the global contention rate should be ignored for a member for it will over report false contention:

GLOBAL CONTENTION RATE (%)
FALSE CONTENTION RATE (%)

Tip: The global contention rate should be less than 3% to 5% of the total XES IRLM requests. If you monitor high global contention rate, you should also see a false contention rate, which should be less than 1% to 3% of the total XES IRLM requests, or less than a half of the global contention.

To manually determine the global contentions and false contentions, first derive the total number of XES IRLM requests, which is equal to the total number of synchronous XES requests and suspends requests, both broken into an IRLM counter, an XES counter, and a false contention counter. The global contention rates can be derived by dividing the total number of suspensions, sum of IRLM, XES, and false contention counters, with the total number of synchronous XES requests. The false contention rates can be also derived from false contention counters divided by the total number of synchronous XES requests:

Total # of suspensions = ‘SUSPENDS - IRLM GLOBAL CONT’ + ‘SUSPENDS - XES GLOBAL CONT.’ + ‘SUSPENDS - FALSE CONT. MBR’

Total # of XES IRLM requests = (‘SYNCH.XES - LOCK REQUESTS’ + ‘SYNCH.XES - CHANGE REQUESTS’ + ‘SYNCH.XES - UNLOCK REQUESTS’) + (‘BACKGROUND.XES -CHILD LOCKS’ + ‘ASYNCH.XES -CONVERTED LOCKS’) + (‘SUSPENDS - IRLM GLOBAL CONT’ + ‘SUSPENDS - XES GLOBAL CONT.’ + ‘SUSPENDS - FALSE CONT. MBR’)

Global contention rate (%) = (Total # of suspensions) / (Total # of XES IRLM requests) * 100

False contention rate (%) = ‘SUSPENDS - FALSE CONT. MBR’ / (Total # of XES IRLM requests) * 100

Example 8-8 shows the data sharing locking counters. It shows global contention of 0.70% and false contention of 0.22%, which is in good status.

Example 8-8 Data sharing locking block

DATA SHARING LOCKING	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
GLOBAL CONTENTION RATE (%)	0.70			
FALSE CONTENTION RATE (%)	0.22			
P/L-LOCKS XES RATE (%)	25.56			
LOCK REQUESTS (P-LOCKS)	118.00	0.02	N/C	1.46
UNLOCK REQUESTS (P-LOCKS)	79.00	0.02	N/C	0.98
CHANGE REQUESTS (P-LOCKS)	37.00	0.01	N/C	0.46
SYNCH.XES - LOCK REQUESTS	1776.00	0.37	N/C	21.93
SYNCH.XES - CHANGE REQUESTS	12.00	0.00	N/C	0.15
SYNCH.XES - UNLOCK REQUESTS	6686.00	1.38	N/C	82.54
BACKGROUND.XES -CHILD LOCKS	9.00	0.00	N/C	0.11
ASYNCH.XES -CONVERTED LOCKS	4924.00	1.01	N/C	60.79

SUSPENDS - IRLM GLOBAL CONT	64.00	0.01	N/C	0.79
SUSPENDS - XES GLOBAL CONT.	0.00	0.00	N/C	0.00
SUSPENDS - FALSE CONT. MBR	30.00	0.01	N/C	0.37
SUSPENDS - FALSE CONT. LPAR	N/A	N/A	N/A	N/A
NO DELAY LOCK REQ REJECTS	0.00	0.00	N/C	0.00
INCOMPATIBLE RETAINED LOCK	0.00	0.00	N/C	0.00
NOTIFY MESSAGES SENT	0.00	0.00	N/C	0.00
NOTIFY MESSAGES RECEIVED	80.00	0.02	N/C	0.99
P-LOCK/NOTIFY EXITS ENGINES	500.00	N/A	N/A	N/A
P-LCK/NFY EX.ENGINE UNAVAIL	0.00	0.00	N/C	0.00
PSET/PART P-LCK NEGOTIATION	31.00	0.01	N/C	0.38
PAGE P-LOCK NEGOTIATION	0.00	0.00	N/C	0.00
OTHER P-LOCK NEGOTIATION	5.00	0.00	N/C	0.06
P-LOCK CHANGE DURING NEG.	34.00	0.01	N/C	0.42

8.4.2 P-lock contention negotiations

In a non-data-sharing environment, the latch is a low level serialization mechanism used to ensure physical consistency of shared DB2 data structures or resources (for example, pages in the buffer pool), implemented to be very inexpensive to acquire (short path length), and very basic in functionality. Because DB2 data sharing requires inter system consistency, the latch cannot be used and needs an additional mechanism to ensure physical consistency. A DB2 data sharing environment uses P-locks to track inter-DB2 data coherency at the page set or partition level, called the *page set/partition p-locks*, and serialization between members, ensuring the physical consistency of a page with another type of P-locks, called the *page p-locks*. Other types of P-locks include DBD, castout, GBP structure, index tree, and repeatable read tracking P-locks.

Key characteristics of P-locks are they are negotiable, for example, one of the DB2 members changes the state of its P-lock on a resource (page set or partition) due to a change in the physical access characteristics on the resource page set P-lock occurs. The other DB2 members that hold a page set P-lock on that resource will be notified by their respective IRLMs of this change in the inter-DB2 interest on the P-lock, and each DB2 member can then dynamically make the necessary adjustments in the cache coherency processing for the resource and then downgrade or upgrade its P-lock state (negotiate the P-lock) accordingly.

P-lock contention and negotiation can be a cause of IRLM latch contention, page latch contention, asynchronous GBP write, active log write, GBP read. Page P-lock contention by one thread causes page latch contention for all other threads in the same member trying to get to the same page.

P-Locks are reported separately in the data sharing locking section of the OMEGAMON PE statistics report, and it breaks out the negotiations at the page-set partition level, the page level, and also for other P-locks. You should keep track of amount of total P-lock negotiation within a data sharing system.

Example 8-9 shows monitoring fields for P-lock negotiations.

Tip: Keep total P-lock negotiation less than 3% to 5% of the total number of XES IRLM requests.

As previously noted in 8.4.1, “Global contentions” on page 99, the total number of XES IRLM requests is equal to the total number of synchronous XES requests and suspends requests:

P-lock Negotiation = ‘PSET/PART P-LCK NEGOTIATION’ + ‘PAGE P-LOCK NEGOTIATION’ + ‘OTHER P-LOCK NEGOTIATION’

P-lock Negotiation Rate (%) = (P-lock negotiation) / (Total # of XES IRLM requests) * 100

Example 8-9 Data sharing locking

DATA SHARING LOCKING	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
GLOBAL CONTENTION RATE (%)	0.70			
FALSE CONTENTION RATE (%)	0.22			
P/L-LOCKS XES RATE (%)	25.56			
LOCK REQUESTS (P-LOCKS)	118.00	0.02	N/C	1.46
UNLOCK REQUESTS (P-LOCKS)	79.00	0.02	N/C	0.98
CHANGE REQUESTS (P-LOCKS)	37.00	0.01	N/C	0.46
SYNCH.XES - LOCK REQUESTS	1776.00	0.37	N/C	21.93
SYNCH.XES - CHANGE REQUESTS	12.00	0.00	N/C	0.15
SYNCH.XES - UNLOCK REQUESTS	6686.00	1.38	N/C	82.54
BACKGROUND.XES -CHILD LOCKS	9.00	0.00	N/C	0.11
ASYNCH.XES -CONVERTED LOCKS	4924.00	1.01	N/C	60.79
SUSPENDS - IRLM GLOBAL CONT	64.00	0.01	N/C	0.79
SUSPENDS - XES GLOBAL CONT.	0.00	0.00	N/C	0.00
SUSPENDS - FALSE CONT. MBR	30.00	0.01	N/C	0.37
SUSPENDS - FALSE CONT. LPAR	N/A	N/A	N/A	N/A
NO DELAY LOCK REQ REJECTS	0.00	0.00	N/C	0.00
INCOMPATIBLE RETAINED LOCK	0.00	0.00	N/C	0.00
NOTIFY MESSAGES SENT	0.00	0.00	N/C	0.00
NOTIFY MESSAGES RECEIVED	80.00	0.02	N/C	0.99
P-LOCK/NOTIFY EXITS ENGINES	500.00	N/A	N/A	N/A
P-LCK/NFY EX.ENGINE UNAVAIL	0.00	0.00	N/C	0.00
PSET/PART P-LCK NEGOTIATION	31.00	0.01	N/C	0.38
PAGE P-LOCK NEGOTIATION	0.00	0.00	N/C	0.00
OTHER P-LOCK NEGOTIATION	5.00	0.00	N/C	0.06
P-LOCK CHANGE DURING NEG.	34.00	0.01	N/C	0.42

8.4.3 Data sharing lock tuning

When you end up having excessive global lock contention on the space map pages or on data pages when using row-level locking, there is a reason for the space maps becoming hot spots, where a regular space map page for the partitioned table space can serve up to 10,000 data pages.

The OMEGAMON PE statistics report gives suspensions and negotiations at the individual group buffer pool level and broken down into space map page, data page, and index leaf pages. See Example 8-10. If index leaf page splits happen often because the leaf pages become full, you are likely to see a high number of P-lock negotiation with index leaf pages.

Example 8-10 Group buffer pool statistics

GROUP BPO	CONTINUED	QUANTITY	/SECOND	/THREAD	/COMMIT
-----		-----	-----	-----	-----
WRITE AND REGISTER		32.00	0.01	N/C	0.40
WRITE AND REGISTER MULT		6.00	0.00	N/C	0.07
CHANGED PGS SYNC.WRTN		44.00	0.01	N/C	0.54
CHANGED PGS ASYNC.WRTN		0.00	0.00	N/C	0.00
PAGES WRITE & REG MULT		12.00	0.00	N/C	0.15
READ FOR CASTOUT		12.00	0.00	N/C	0.15
READ FOR CASTOUT MULT		12.00	0.00	N/C	0.15
PAGE P-LOCK LOCK REQ		56.00	0.01	N/C	0.69
SPACE MAP PAGES		0.00	0.00	N/C	0.00
DATA PAGES		41.00	0.01	N/C	0.51
INDEX LEAF PAGES		15.00	0.00	N/C	0.19
PAGE P-LOCK UNLOCK REQ		62.00	0.01	N/C	0.77
PAGE P-LOCK LOCK SUSP		39.00	0.01	N/C	0.48
SPACE MAP PAGES		0.00	0.00	N/C	0.00
DATA PAGES		28.00	0.01	N/C	0.35
INDEX LEAF PAGES		11.00	0.00	N/C	0.14
PAGE P-LOCK LOCK NEG		0.00	0.00	N/C	0.00
SPACE MAP PAGES		0.00	0.00	N/C	0.00
DATA PAGES		0.00	0.00	N/C	0.00
INDEX LEAF PAGES		0.00	0.00	N/C	0.00
PAGES IN WRITE-AROUND		0.00	0.00	N/C	0.00

Here are some remedies for having excessive global lock contentions:

- ▶ Setting TRACKMOD to NO. This will turn off the tracking of modified pages in the space map page, thus it reduces the number of updates to the space map page and helps reduce contention. The down side of setting TRACKMOD to NO is because DB2 is no longer tracking modified pages; the incremental image copy will now have to run for longer, because it will have to use a table space scan or partition scan to identify the modified pages, as opposed to using the space map to identify those modified pages.
- ▶ Specifying MEMBER CLUSTER to partitioned table spaces, with DB2 10 and later, MEMBER CLUSTER can be specified to universal table spaces. This option might provides relief for insert intensive workloads, reducing page P-lock and page latch contention, better space use, and working set of pages in buffer pools.

Results of specifying this option is that it increases the number of space map pages, where instead of servicing one space map page per 10,000 pages, DB2 serves one space map page per 199 data pages. In addition, it creates a loose affinity between data sharing members, and a space map page and the associated data pages. As a result, you get relief, but the trade-offs of using MEMBER CLUSTER is that you lose clustering of the data rows.

Because of these considerations, it is good practice to use LOCKSIZE PAGE as a design default and use LOCKSIZE ROW as needed, and specify MEMBER CLUSTER for additional relief. Having workloads inserting the rows at the end of the table space by design with APPEND option or distributed free space exhausted will end up having excessive page P-lock on the space map page, and on the data pages. If you want to have LOCKSIZE ROW for APPEND table spaces, specify MEMBER CLUSTER. For mixed INSERT/UPDATE/DELETE workloads, it is suggested not to use LOCKSIZE ROW, for MEMBER CLUSTER does not help the update and delete cases. Instead, consider using MAXROWS 1 with LOCKSIZE PAGE to let DB2 behave like row level locking. However, this should not be applied to all cases, having inefficient space usage, likely to be applied only to small sized tables.

Tip: In general, if an object is likely to become GBP dependent, use LOCKSIZE PAGE, unless MEMBER CLUSTER specified.

8.5 Internal DB2 latch contention

DB2 uses latches to serialize access to different memory resources, such as the log output buffer, storage, or control block chains. A latch class hierarchy is used to prevent a “deadlatch” by making sure that latches are always obtained in the same sequence by all requestors.

DB2 counts the number of times a process has to wait for a latch in a number of different statistics latch class counters. Since APAR PK77514, which removed the serviceability specification for latch class counters, the description of each class is documented in the DSNDQVLS macro and can be found in SDSNMACS. Table 8-4 gives a description for each of the latch classes.

Table 8-4 Latch classes

LC	Primary content	LC	Primary content
LC1	Infrequently used	LC18	DDF resync list
LC2	Global authorization cache	LC19	Log write
LC3	DDF disconnect	LC20	System checkpoint
LC4	SYSSTRING cache	LC21	Accounting rollup
LC5	IRLM data sharing exits or RLF	LC22	Internal checkpoint
LC6	Data sharing index split	LC23	Buffer manager: <ul style="list-style-type: none"> ► Add page latch waiter on timer queue ► Add remove to/from deferred write queue for GBP-dependent objects

LC	Primary content	LC	Primary content
LC7	Index latch and OBD allocation	LC24	EDM pool LRU chain Buffer Manager page unlatch and prefetch
LC8	Query parallelism	LC25	Workfile allocation EDM hash chain
LC9	Utilities or stored procedure URIDs	LC26	Dynamic statement cache
LC10	Allied agent chain or sequence descriptors Create thread queued by reaching CTHREAD limit	LC27	Stored procedures queue
LC11	DGTT allocation Sequence or identity column	LC28	Stored procedures or authorization cache
LC12	Global transaction ID table	LC29	Field procs and DDF transaction manager
LC13	Pageset operations	LC30	Agent services
LC14	Buffer pool hash chain and LRU chain	LC31	Storage manager
LC15	ARCHIVE LOG MODE(QUIESCE)	LC32	Storage manager
LC16	UR chain	LC254	Index latch
LC17	RURE chain		

There are many more types of latches than the number of latch classes, currently 33, and so each latch class is counting the number of times DB2 has to wait for a latch for multiple types of latches. In order to be sure which type of latch is actually causing the counter to be incremented, it is necessary to run a trace.

You can use the following IFCIDs shown in Table 8-5 to obtain detailed information about DB2 latch suspensions.

Table 8-5 Latch suspensions

IFCID	Description
51	Shared latch resume
52	Shared latch wait
56	Exclusive latch wait
57	Exclusive latch resume

These IFCIDs have the real DB2 latch number and the address of the latch. If the system experiences a high degree of DB2 latch contention, and if you want to identify the actual latches that are triggering the latch contention, it is normally sufficient to trace only IFCID 56 and 57. As there are many latch contentions, there must be X latch requests that will be suspended (since S and S are compatible), so in order to reduce the amount of trace data being gathered, there is normally no need to trace IFCID 51 and 52.

The following trace command will gather this information:

```
-STA TRA(P) CLASS(30) IFCID(56,57) DEST(SMF/GTF) TDATA(CPU COR DIST)
```

The latch class statistics counters are just that: counters. They are reported by OMEGAMON PE statistics long report as the number of times a process waits per second. If you need to know the latch duration, then take a look at the accounting report class 3 suspensions, where you can find the field LOCK/LATCH(DB2+IRLM). Example 8-11 shows OMEGAMON PE statistics report output sample for latch class counters.

Example 8-11 Latch counters

LATCH CNT	/SECOND	/SECOND	/SECOND	/SECOND
-----	-----	-----	-----	-----
LC01-LC04	0.00	0.00	0.00	0.00
LC05-LC08	0.00	0.00	0.00	0.00
LC09-LC12	0.00	0.00	0.00	0.00
LC13-LC16	0.00	3.73	0.00	0.00
LC17-LC20	0.00	0.00	0.04	0.05
LC21-LC24	0.00	0.00	188.38	19.04
LC25-LC28	0.28	0.00	0.00	0.00
LC29-LC32	0.02	0.02	0.02	0.05
LC254	0.00			

Prior to DB2 10, the way that DB2 latch contention management works is that when multiple requestors are contending for the latch, all the waiters are suspended. When a holder gives up the resource, it starts over contending for the latch. So having high latch contention means the waiters are continually woken up to contend for the latch, and then suspended again. And if you turn on the accounting trace class 3, it will also aggravate the cost. It is not a general recommendation to turn off accounting class 3, so get to the root cause of the problem.

Tip: A contention rate of 10,000 per second needs to be looked into for tuning possibilities.

The following sections give the commonly reported high latch classes and the possible cause.

LC06: index tree P-lock held during an index page split

This is latch type 70 (or X'46') in the performance trace. This only applies to data sharing. The equivalent for non-data sharing is counted in latch class 254, which is also latch type 254 (or X'FE'). The latch is held during index tree structure modification.

These are remedies for high values:

- ▶ Reduce index splits by use of the REORG utility to add free space.
- ▶ Minimize index key size, especially for a unique index.
- ▶ Use the NOT PADDED option on CREATE INDEX or ALTER INDEX for indexes with VARCHAR columns with a length of more than 18 bytes. If you have VARCHAR columns with a length more than 18 in indexes, consider altering these indexes to NOT PADDED, rebuild the indexes, and rebind dependent packages that may be able to exploit the index-only access.
- ▶ Consider using larger page sizes in DB2 9 or later to reduce the frequency of index splits.
- ▶ Note that DB2 9 introduced this.

The number of index splits can be estimated from LEAFNEAR/FAR in SYSINDEXPART and also REORGLAUFNEAR/FAR in the real-time statistics table INDEXSPACESTATS.

DB2 10 introduced IFCID 359 to monitor index splits.

LC14: virtual buffer pool latch

This class is used to count the latch waits for managing the hash chains when PGSTEAL=LRU.

These are remedies for high values:

- ▶ Use FIFO, if the objects are resident in memory, if there is no or minimal read I/O that occurs, for example, object(s) are entirely in a buffer pool. DB2 10 introduced NONE to be used for “in memory” objects, where data will be preloaded into the buffer pool at the time the object is physically opened and remains resident, which eliminates the need to maintain LRU chain.
- ▶ Increasing the size of the pool can reduce hash contention. You must ensure that any increase is backed by real storage and does not result in z/OS paging to auxiliary.
- ▶ Assign the objects across more buffer pools to balance the workload as measured by the rate of GETPAGE requests.

LC19: log write latch

This latch class counts the number of times a wait occurs for the log buffer to become available. There is only one latch for the whole buffer, so every time an agent or resource manager in DB2 wants to generate a log record, DB2 has to acquire a latch to move the strings or bytes related to the log record into the buffer.

These are remedies for high values:

- ▶ Minimize the number of log records created as follows:
 - LOAD RESUME/REPLACE with LOG NO instead of massive INSERT/UPDATE/DELETE.
 - Segmented or universal table space if mass delete occurs.
- ▶ Increase the size of the log output buffer if there is a non-zero unavailable count:
 - Unavailable is reported in field UNAVAILABLE OUTPUT LOG BUFF in the statistics report.
 - When available, the first agent waits for the log write; all subsequent agents' waits are counted in LC19.
- ▶ Allocate more real storage or reduce the size of the output log buffer if there is non-zero output log buffer paging. Paging is reported in field OUTPUT LOG BUFFER PAGED IN in the statistics report.
- ▶ Reduce use of class 3 accounting if there are more than 10,000 latch contentions per second.

V9 might reduce LC19 for relief by log latch not held while spinning for unique LRSN. And DB2 10 might give further improvement by minimizing the time log latch being held, and conditional attempts to get log latch before unconditional requests.

LC24 latches

This is one of the common forms of latch contention, but there are at least two types of latch contention being mapped onto that one counter. We need to understand which one is contributing to the counter.

LC24: EDM pool LRU latch

This is latch type 18 (or X'24').

These are remedies for high values:

- ▶ Enable thread reuse with RELEASE DEALLOCATE instead of COMMIT for frequently executed packages (greater than 20 per second). Be aware that RELEASE DEALLOCATE prevents some utilities from executing. In DB2 11, enhancement has enabled those prevented agents to break in.
- ▶ Consider increasing the EDM pool size.
- ▶ Use DSNZPARM EDMBFIT=NO. Parameter removed in DB2 10 and later.

DB2 10 has moved CT/PT from EDM RDS pool to agent local storage; this has significantly reduced LC24.

LC24: Pre-fetch scheduling

This is latch type 56 (or X'38'). The latch is obtained when any prefetch engine is scheduled or ended, or a GETPAGE to check if a page is being prefetched. The user threads get the latch when scheduling a prefetch and when checking for a prefetch in progress. The prefetch engine gets the latch to remove itself from the chain when the prefetches are complete. High contention may occur with these conditions:

- ▶ Many dynamic prefetch in star joins
- ▶ CPU query parallelism with many degrees
- ▶ Small page sets scanned frequently by many concurrent threads.

These are remedies for high values:

- ▶ Reducing the many concurrent prefetches.
- ▶ Disabling dynamic prefetch if not required by setting VPSEQT=0. This may be possible if there is minimal real I/O and no need for parallelism. Prefetch engines for list and dynamic prefetch are still started even if all pages are resident in the pool. Sequential prefetch is not started until the first page miss occurs, but then continues until the prefetch operation is completed.
- ▶ Using more partitions, as there is one latch per data set.
- ▶ DB2 10 introduced PGSTEAL=NONE for “in memory” data or index buffer pools.



The zIIP engine and DB2

Since 2007, the System z Integrated Information Processor (zIIP) has gained notoriety from customers and assisted with more functions both in and outside of DB2. This chapter describes the function of the zIIP specialty purpose engine and discusses customer scenarios, as well as best practices to exploit them.

This chapter contains the following topics:

- ▶ What a zIIP is
- ▶ What work is zIIP eligible
- ▶ How many zIIPs are enough
- ▶ Avoiding self-imposed bottlenecks
- ▶ Getting more zIIP eligibility

9.1 What a zIIP is

The System z Integrated Information Processor (zIIP) became generally available in 2006 with the z9® architecture. It was originally introduced to sway new workload onto the mainframe, by allowing certain workloads to run on the zIIP capacity, without that capacity incurring IBM software costs. The engine itself is a repurposed general purpose engine microcode enabled to be a zIIP. The z/OS dispatcher controls which applications, and how much of them get processed by the zIIP. zIIP engines also have a strong total cost of acquisition argument compared with general purpose (GP) engines and always run at the full uniprocessor speed of the CEC.

For instance, a z196 model 701 runs at 1,202 MIPS, while a model 401 runs at 240 MIPS. The model 401 is quite a bit less expensive and the monthly license charge for software running on it would be less as well. This is referred to as subcapacity engines and are artificially limited in speed to reduce costs. So a customer could purchase a model 401, but if there were zIIPs installed, the zIIPs would run at the 1,202 MIP rating. This 5x difference in speed could be quite beneficial for a long running warehouse type query, and DB2 has released PTFs to ensure that the response time is aided by the zIIP but will remain consistent during different executions.

The old (pre-V8) rule of thumb was that distributed transactions incurred roughly 50% more processing cycles than those applications that ran and attached to DB2 locally. Hence the original number for zIIP offload of DRDA workload was 50-55%, to offset customer reluctance to new workloads. That number has since been increased to 55-60% offload for remote requests.

Tip: See the zIIP home page with prerequisites and potential workloads.

<http://www-03.ibm.com/systems/z/hardware/features/ziip/index.html>

9.1.1 What drives customers to purchase zIIPs

Here are some reasons for purchase:

- ▶ Opportunity to lower monthly license charge by affecting the 4-hour peak rolling average CPU utilization with subcapacity pricing.
- ▶ Avoid/delay a processor upgrade or expansion by offloading some of the peak workload, which can also avoid higher software costs associated with a higher capacity machine.
- ▶ Improve the performance of lower importance workload by freeing up processing space on the GCPs as well as processing latent demand (discretionary DDF work).

9.2 What work is zIIP eligible

In order for work to be zIIP eligible, the work must run in enclave service request block (SRB) mode. This is how the z/OS dispatcher is able to send the work off to the zIIP engine. Remote DRDA work was one of the first exploiters of the zIIP engine because when it enters DB2 as a DBAT (database access thread), it has already executed in enclave SRB mode. These enclaves are a logical grouping of attributes that allow WLM to manage the work as it crosses through multiple address spaces, doing work on behalf of the parent task or address space.

Parallel tasks ran as client SRBs prior to the zIIP engine, because DB2 needed to tie the child tasks back to the parent task. Parts of utility work, such as the index build phase, were also modified to be moved to an enclave SRB. As time went on, more of the utilities, native stored procedures, and finally some DB2 system agents were moved to run in enclave SRB mode to become eligible for ZIP redirect.

9.2.1 zIIP eligibility from DB2 V8 until now

The table in Figure 9-1 divides the DB2 functions by version and zIIP eligible workload to show what can be offloaded and what, if any, prerequisites exist. Each new release of DB2 builds on the eligible functions from the last release and adds new eligible code as development is able to move the agent to run in SRB mode. DB2 V8 was the initial vessel and allows up 50-55% of remote DRDA work, utility work related to the index BUILD phase, and parallel child tasks. DB2 9 added remotely invoked native stored procedures. DB2 10 added basic RUNSTATS statistics as well as some of the system agents. DB2 11 adds the majority of the rest of the system agents, and distribution statistics as well as inline statistics.

<u>zIIP Eligible</u> APAR <u>II14219</u>	<u>Function</u>	<u>Amount Redirected</u>	<u>Prerequisites</u> z/OS 1.8 – base feature z/OS1.9 – WLM weights on zIIPs
<u>DB2 10</u>	<ol style="list-style-type: none"> 1) All of DB2 v8 and 9 offload++ 2) RUNSTATS 3) Prefetch and deferred write processing 4) Parallelism enhancements 5) Clean-up of multi-version XML columns after update/delete 	<ol style="list-style-type: none"> 1) BUILD phase, up to 60% remotely called native SQL procs, parallelism, DRDA requests 2) Basic RUNSTATS for table, NO Histogram, DSTATS, COLGROUP... BUT index stats almost all offloaded (not DPSIs) 3) 100% (roughly 70% of DBM1 SRB time) 4) Parallelism limitations lifted (80% of child tasks eligible) 5) 100% 	<ol style="list-style-type: none"> 1) DB2 10/ z/OS 1.10 2) Run RUNSTATS, no inline STATS 3) Shows up in DBM1 SRB time 4) DB2 10 NFM with rebind 5) PM72526
<u>DB2 11</u>	<ol style="list-style-type: none"> 1) More RUNSTATS 2) LOAD REPLACE with dummy input 3) Most of the system engines (GBP write, castout, notify/exit) 4) Log write, log prefetch 5) Index pseudo delete cleanup 6) XML versions clean up 	<ol style="list-style-type: none"> 1) COLCARD, FREQUAL, HISTOGRAM statistics, including inline stats (80%, possibly more) 2) 100% of delete processing eligible on NPI 3) 100% eligible 4) 100% eligible 5) 100% eligible 6) 100% eligible 	<ol style="list-style-type: none"> 1) DB2 11, z/OS 1.13, z10 2) “ “ 3) “ “ 4) Shows up in MSTR IIP SRB time 5) Be aware of INDEXCLEANUP_THR EADS affect on zIIP capacity

Figure 9-1 zIIP eligible work

There are many other zIIP eligible workloads coming from other IBM products as well as vendor products. Some examples are OMEGAMON PE Near Term History processing, the uncompress and decompress operation of Capture in Q Replication, DFSORT Memory Object Sorting, portions of DB2 Sort processing, Data Mover functions of XRC, Hiper Sockets for large messages, and so on. We focus on the DB2 eligible work and a methodology for sizing based on zIIP utilization.

9.2.2 System agents

Looking in the statistics report from OMEGAMON PE, we can see the effect of the system agents becoming zIIP eligible. If you are going from DB2 8 to DB2 10, there is a large increase in the amount of prefetch done, as DB2 switched to dynamic prefetch for anything other than a table space scan. This change actually occurred in DB2 9, but row level sequential detection in DB2 10 encourages dynamic prefetch even more. This results in up to roughly 80% of the DBM1 SRB time becoming zIIP eligible, depending on the proportion of CPU time spent doing prefetch in your environment.

In the reports in Example 9-1 and Example 9-2, you notice the time moving from NONPREEMPT SRB to PREEMPT IIP SRB. The zIIP eligible time is likely more than the NONPREEMPT SRB time you saw in DB2 9 due to increases in dynamic and list prefetch. If you use compressed indexes and those pages were brought in via prefetch, then the expansion of those pages in the buffer pool would be charged to the prefetch engine, and hence the zIIP.

Example 9-1 DB2 9 SRB time

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT IIP SRB
SYSTEM SERVICES ADDRESS SPACE	9.110613	0.000000	15.724902	24.835515	N/A
DATABASE SERVICES ADDRESS SPACE	11.225272	0.000000	2:08.598926	2:19.824199	0.000000
IRLM	0.133232	0.000000	7.310644	7.443875	N/A
DDF ADDRESS SPACE	0.174751	6:13.258498	14.847385	6:28.280634	9:01.598026

Example 9-2 DB2 10 SRB time

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT IIP SRB
SYSTEM SERVICES ADDRESS SPACE	17.316781	10.854370	1.499365	29.670516	N/A
DATABASE SERVICES ADDRESS SPACE	19.647342	10.675517	0.605379	30.928238	6:47.777756
IRLM	0.135340	0.000000	8.655075	8.790415	N/A
DDF ADDRESS SPACE	1.028290	12:44.002157	11.534363	12:56.564811	24:26.104312

In Example 9-3, notice the PREEMPT IIP SRB time in the SYSTEM SERVICES ADDRESS SPACE which is the MSTR address space time attributed to log writes and log prefetch in DB2 11. The higher the proportion of “writer” to “reader” threads, the higher the proportion of SRB time becomes zIIP eligible. Test cases show around 80% of the MSTR SRB time moving to the zIIP (50% of total TCB+SRB) between DB2 10 and DB2 11.

Example 9-3 DB2 11 SRB time

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	CP CPU TIME	PREEMPT IIP SRB
SYSTEM SERVICES ADDRESS SPACE	3:17.800882	36.817951	8.594475	4:03.213309	23.816602
DATABASE SERVICES ADDRESS SPACE	6:57.869274	1:54.865921	56.978017	9:49.713212	2:55.829592
IRLM	0.012570	0.000000	3:48.668250	3:48.680820	0.000000
DDF ADDRESS SPACE	2.745241	0.257366	0.257859	3.260466	0.000000

What if you do not have a zIIP installed? Example 9-4 shows a user on DB2 10 without any zIIP installed, so there is over an hour and a half of PREEMPT SRB time that ran on a general purpose engine. Not all of that would have been zIIP eligible, but roughly 70% was probably prefetch, and that could have made it. Notice that, unlike in the accounting reports, we do not report on zIIP eligible time that ran on a GP, it just shows up in the PREEMPT SRB column. To determine how much of this work really could have gone to the zIIP, you would need to look in the WLM Activity report, in the service or report class where the DB2 DBM1 address space lives. There the time would show up as APPL% IIPCP.

Example 9-4 DB2 10 customer without any zIIP engines

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT IIP SRB
SYSTEM SERVICES ADDRESS SPACE	1:30.516956	12.441676	7.644849	1:50.603480	N/A
DATABASE SERVICES ADDRESS SPACE	3:35.651891	1:36:16.888767	12.718051	1:40:05.258709	0.000000
IRLM	0.005676	0.000000	1:54.374389	1:54.380065	N/A
DDF ADDRESS SPACE	4.053989	1.749048	2.036603	7.839641	0.000000

9.2.3 Native stored procedures

With the rise of stored procedures as a way to bridge the gap between dynamic JDBC access and statically bound packages, native stored procedures in DB2 9 have grown in popularity.

The table in Figure 9-2 shows the relative billable or general CP cost of different types of stored procedures. We use COBOL as the base line, 'x', due to its popularity despite C being now the cheapest language, since we actually compile the stored procedures into C internally. Native SQL stored procedure runs entirely in the DBM1 address space, so it ends up being a little cheaper than the external SQL stored procedure which runs in a WLM managed address space. With the addition of the zIIP, the native stored procedures begin to look even more attractive: since if they are invoked remotely, up to 55-60% of the SQL is zIIP eligible. This is the same proportion as other DRDA compliant remote requests. In our example here it is 56% cheaper moving to native stored procedures if they are invoked remotely. COBOL stored procedures called remotely on a system where a zIIP is present can see 10% or less zIIP offload as only the CALL statement and result set processing is eligible, not the SQL itself.

Stored procedures with zIIPs		
Language	Base Billable Cost	Billable Cost after zIIP and/or zAAP acceleration
COBOL stored proc	X (Baseline)	.88x
C stored proc	.95x	.83x
Remote SQLJ	1.78x	1.06x
SQLJ stored proc	1.21x	1.15x (zIIP + zAAP)
JDBC stored proc	2.11x	1.76x (zIIP + zAAP)
External SQL stored proc	1.62x	1.49x
Native SQL stored proc	1.14x	.65x

Figure 9-2 zIIP and stored procedures

9.2.4 zAAP on zIIP

The System z Application Assist Processor (zAAP) debuted in 2004 to encourage customers to run JAVA workload natively on z/OS. Basically anything that ran within the JVM was eligible to run on the zAAP specialty engine. Two years later the zIIP engine emerged to alleviate the CPU burden of massive distributed SQL workloads; some utility executions; as well as complex data warehouse queries. Originally even the PARMLIB settings of these engines differed. You were able to segregate and force zAAP eligible work to the zAAP engine (IFACROSOVER), whereas with the zIIP, you simply had to configure the LPAR correctly for the zIIPs to absorb all of the work.

Over time, similar PARMLIB settings were adopted; settings such as IIPHONORPRIORITY came into the picture to force zIIP eligible work to wait for the zIIP engine. With the convergence of system parameters and the common underlying theme of offloading “new” workloads such as JAVA and distributed requests, it was logical to coalesce these two specialty engines into a single entity. Starting with z/OS 1.9 and APAR OA27495, and native on z/OS 1.11, a new IEASYSxx PARMLIB setting zAAPZIIP=YES allows zAAP eligible workload to run on a zIIP. This capability is beneficial for customers who have zIIP engines installed and no zAAP engines installed, but have some zAAP eligible (Java) workload.

Tip: If you are planning on running zAAP on zIIP work, see the “Capacity Planning for zAAP and zIIP Specialty Engines” techdoc at this website:

<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD103548>

9.3 How many zIIPs are enough

The design point of the zIIP (or zAAP) was as an *assist* processor. The processor can certainly run at 100% utilization, just as any general CP (GCPs), and is not capped in anyway. But the intent is to alleviate stress, and free up cycles on the GCPs. Attempting to drive the utilization and capacity plan as you would with GCPs is a flawed philosophy. Generally speaking, we assume you have excess zIIP capacity and the dispatcher attempts to get zIIP eligible work to run there first, before allowing it to overflow to a GCP.

If you are capacity constrained on a zIIP, the effects are seen at lower utilization than would be seen on a GCP, because as a general rule, customers have less zIIPs in their system than GPs. When the zIIP debuted, many customers had one or two zIIPs and a zAAP in their System z environment as part of a trial or to allow for testing. Now with more work, and more important work going to the zIIP, capacity planning must be considered for the zIIP engines.

Overall tuning tips for zIIP processing:

1. Ensure everything that is zIIP eligible is running on the zIIP (see Figure 9-5 on page 117):
 - This could be due to sheer capacity, or configuration of LPAR weights, and sharing of zIIPs across LPARs.
 - Utilize WLM activity reports to see if anything is being missed.
2. If there is opportunity, test and increase zIIP eligible work (see 9.5, “Getting more zIIP eligibility” on page 122):
 - Scan the statistics report to see if there is any parallel activity currently, maybe it can be expanded
3. Investigate new technologies and do not be afraid of remote access:
 - Try native stored procedures.
 - Use newer complex SQL, which might also encourage parallelism.
 - Do an internal benchmark of T2 versus T4 attach to ensure that applications use what is most beneficial for the overall system.

9.3.1 Enough zIIP for the business

The business case for purchasing and tuning zIIP eligible workload is not complicated. There is no need to worry about zIIP utilization, especially compared to GCP utilization, to validate the purchase or upgrade. It should *not* be viewed as “Our current zIIPs are only 20% utilized, so we will not need another until they are 70% utilized.” This might work for GCPs, but not for zIIPs. For the business case, if just a portion of the zIIP eligible work executes when the CEC (box) is the most utilized, then there could be opportunities for savings.

This all depends on pricing and contractual specifics, but if you use subcapacity pricing, then you want to lower the 4 hour peak monthly rolling average of processor utilization. Going back to our example (9.1, “What a zIIP is” on page 110), if you had a z196 401 with a 240 MIP GCP and a 1,200 MIP zIIP, then if the zIIP were only 12% utilized at peak, it would be reducing the chargeable MIPs during that peak by 50%! That could have a significant impact on the software costs.

The IBM Techline¹ does hardware sizings for customers. They provide assistance for capacity expansion, or model upgrades as well as to determine the usefulness of a zIIP/zAAP engine. In the example in Figure 9-3, a customer collected RMF 30, 70, and 72 records from their peak processing day of the month. Techline compared the time intervals using LPAR trend reports, for GCP and zIIP utilization, and the WLM activity report to look for zIIP eligible workloads.

Using the WLM activity report, at a WLM level, it is possible to see if zIIP eligible work overflowed to a GCP (Figure 9-5 on page 117). This is reported in the 72 subtype 2 record as APPL% IIPCP.

Next, in Figure 9-3 we see a peak utilization of around 95%. In Figure 9-4 we see that the zIIP peaked at 45% utilization, with another 5% of *Potential* depicted as the green tip on the peak. This means that during the peak time on the peak day, there was another 5% of processing that could have been saved off the GCPs if there had been enough zIIP capacity, so there were potential software savings missed.

These percentages are based on the cumulative processing power of the box, so if there are multiple GCPs and zIIPs, you would need to determine what percentage of AAPL% IIPCP seen on the zIIP equates to in terms of GCP MIPS that could be offloaded. Even if the total zIIP utilization on the box is single digit, most of the time it makes sense to have enough zIIP capacity to absorb every possible eligible MIP during peaks.

¹ <https://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler/LLIE-6LLS4T>

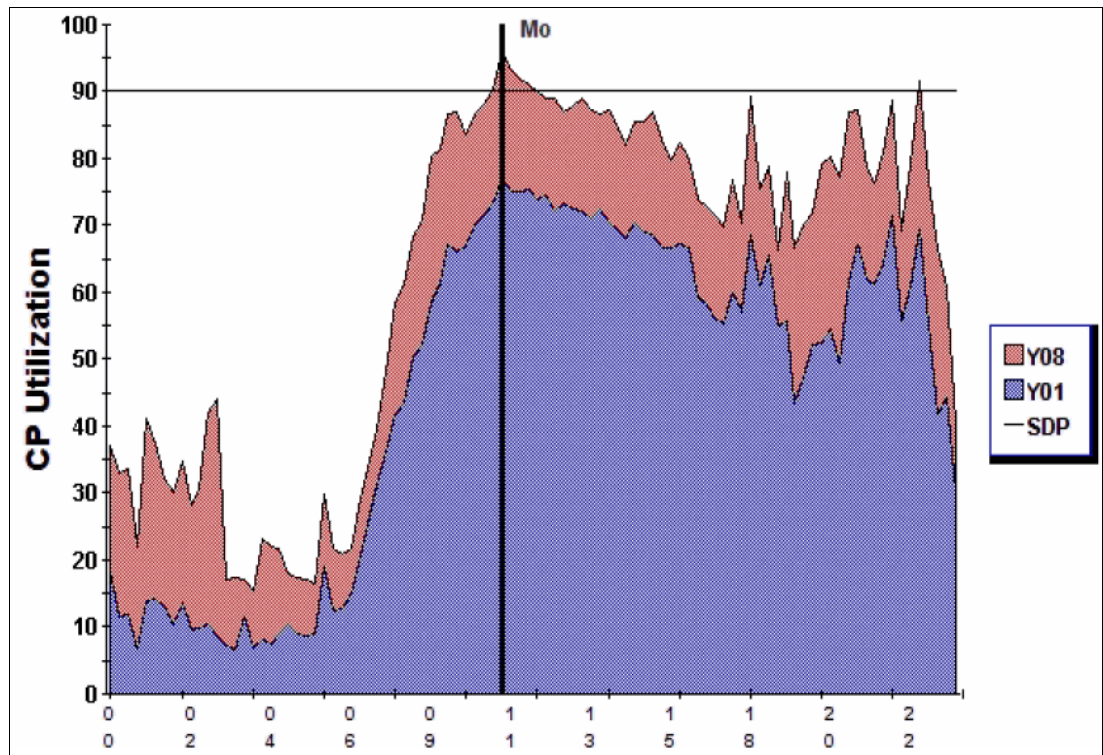


Figure 9-3 Peak CPU aligned

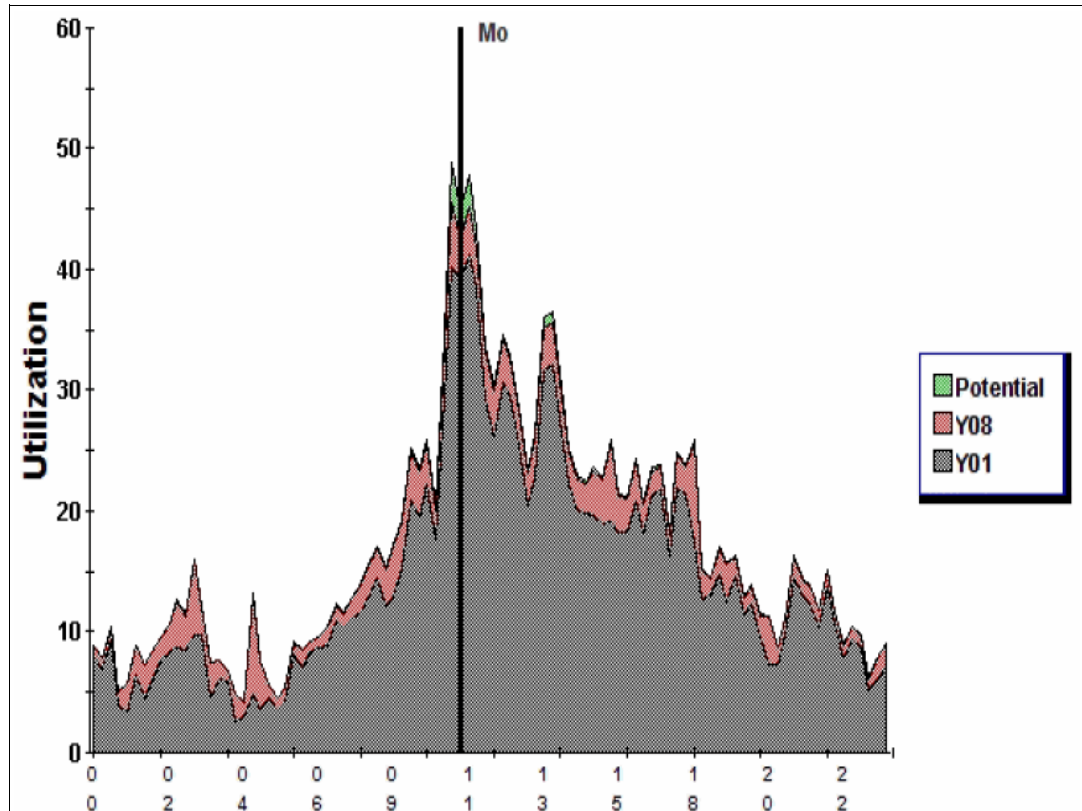


Figure 9-4 Peak zIIP aligned

9.3.2 Enough zIIP for the workload

In this section, we look at the technical side and repercussions of not having enough zIIP capacity. Figure 9-5 shows another example of APPL% IIPCP, meaning that roughly 40% of a single GCP was occupied by zIIP eligible work that overflowed to the GCP (at 15:45).

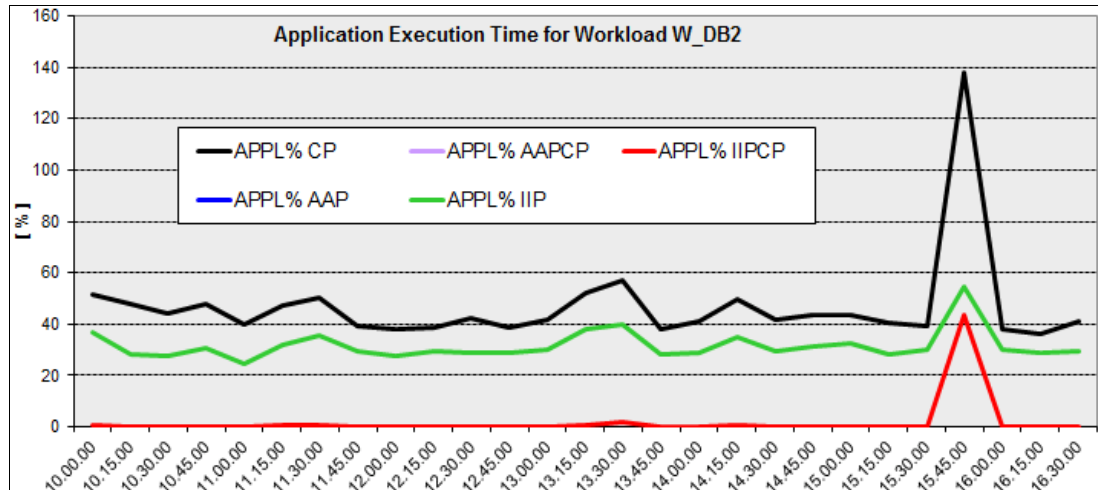


Figure 9-5 Missed zIIP potential

Attention: LPAR trend reports as shown in Figure 9-3 on page 116 are based on the percentage of total capacity on the machine (% of multiple processors), whereas WLM activity reports and APPL% IIPCP is based on the uniprocessor capacity of the machine (% of 1 processor).

You might be thinking that during this 15 minute interval at 15:45, the LPAR trend report would show that the zIIP capacity had been exhausted and it was running at 100% utilization. That, however, is not the case. Figure 9-6 shows that at the same time (15:45) the zIIP was only about 35% utilized. Why is this so?

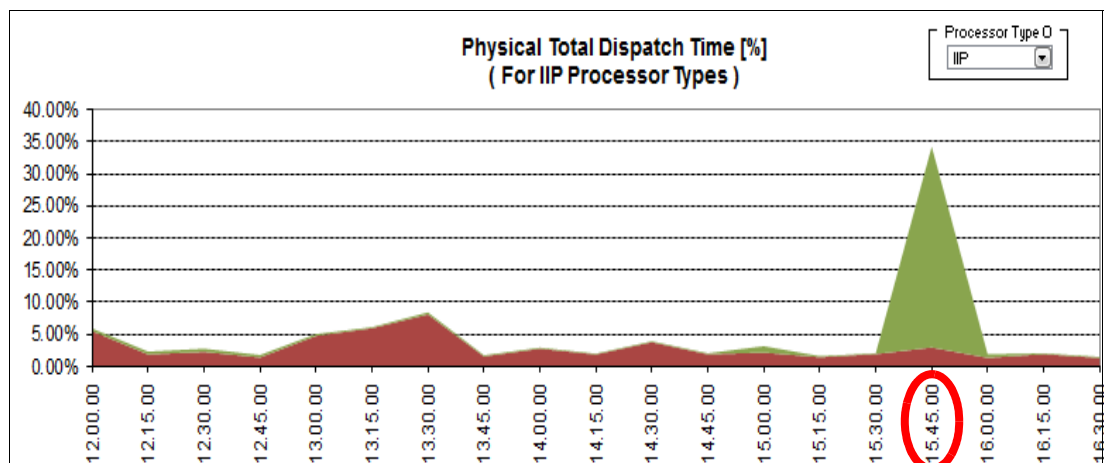


Figure 9-6 LPAR trend of zIIP usage

You have to understand what went on behind the scenes of these two graphics, as well as knowing a little about z/OS dispatching.

During this 1 RMF interval, where we see the APPL% IIPCP spike, there was a sudden flood of 4x the number of distributed threads than normal. Since 60% of these threads were zIIP eligible, the single zIIP on this LPAR was overwhelmed. If all of the eligible DDF threads had queued up behind the single zIIP engine and waited to get dispatched, there would have been a significant impact to the response time and throughput.

What we see in the graph is that the dispatcher noticed this queueing on the zIIP after the sudden spike in utilization, and dispatched some of the eligible work back to a GCP. The dispatcher looks at the pool of zIIP engines and tries to send zIIP eligible work there first, and when the pool is overrun with work, it then tries to send some of the work to the pool of GCPs so that the work will not miss its WLM service class goals. There is more on what parameters affect this situation in 9.4, “Avoiding self-imposed bottlenecks” on page 119.

So why did zIIP eligible threads flow back to the GCP below 100% utilization?

Of course, with a 15 minute RMF interval, much of the granularity is lost, and the zIIP was certainly more than 35% utilized at this point. The answer goes back to queueing theory.

If you have a single zIIP on the box and many GCPs, then you cannot simply compare the two utilization numbers. For instance, in the customer case just shown, there were 12 GCPs which in total were 65% utilized looking at the RMF report. That means each CP is 0.5% (0.65 busy ^12 CPs) instantaneously busy. This is simply queue theory since with a fixed utilization rate, the more processors that are involved, the lower utilization for each processor.

This is a mathematical equation, not something you see in the report. On the other hand, this system only had 1 zIIP that is 35% busy, so it is instantaneously really 35% busy. Queueing theory shows us that if a single processor is 50% busy, the response times will double. Going from 10% to 35% will increase the elapsed time by 50%, a noticeable delay. If, however, you add just one more zIIP, then you can see from Figure 9-7 that the response time will not double until the processor is over 70% utilized.

Tip: The point here is that if you have multiple GCPs and only 1 zIIP, there will be times where zIIP eligible work runs on a GCP even at low zIIP utilization and there could be performance implications.

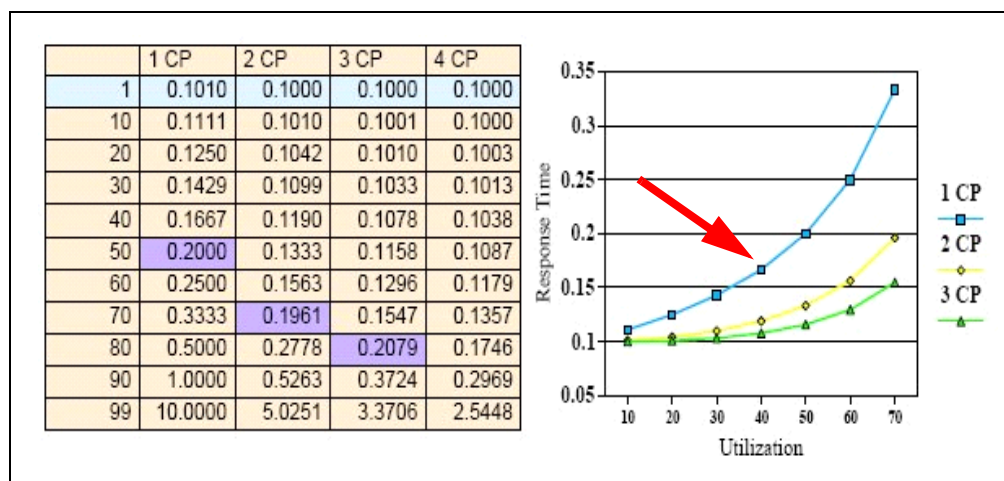


Figure 9-7 Response time based on CPU utilization

The red arrow shows where delay time begins to increase substantially (Markov's equation).

9.4 Avoiding self-imposed bottlenecks

In this section, we examine PARMLIB settings and their impact on zIIP capacity.

9.4.1 IEAOPTxx parameters

There are two PARMLIB IEAOPTxx parameters in particular that can affect zIIP offload that DB2 professionals should be aware of, as they could affect the DB2 workload:

- ▶ IIPHONORPRIORITY
- ▶ ZIIPAWMT

IIPHONORPRIORITY

This parameter determines if, when the zIIP processors become unable to execute all zIIP eligible work, standard logical processors are allowed to help execute zIIP load:

- ▶ IIPHONORPRIORITY= YES (the default)

The dispatcher is free to send excess workload back to a GCP if the zIIP pool becomes overwhelmed. It does this proactively based on the amount of work coming in and how long it is taking to be dispatched

- ▶ IIPHONORPRIORITY= NO

NO means that zIIP eligible work will queue until zIIP processing cycles are available. This could create unpredictable response times and system hangs, especially in DB2 10 and DB2 11.

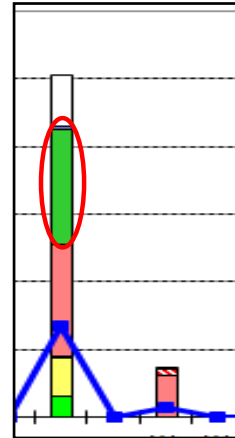
In DB2 11, if the subsystem comes up and detects that IIPHONORPRIORITY=NO, then the system agents will no longer be dispatched to the zIIP. This is to protect the overall system performance in the event of a zIIP capacity shortage.

Figure 9-8 shows the result of setting IIPHONORPRIORITY=NO. In this example, the customer was running a COBOL process for a data warehouse with a parallel degree of 4. The box was configured with 4 GCPs and 1 zIIP. With parallel queries, up to 80% of the child tasks are zIIP eligible. Since 3 out of the 4 child tasks would then be zIIP eligible (3/4 =75%), those 3 tasks queued up waiting for the 1 zIIP engine and ran serially. The result was almost an hour worth of not accounted for time in the application.

Looking at the WLM activity report and the reasons for the processing delays, we see that the number 1 delay (33%) was due to waiting for the zIIP engine. The design point is to never have tasks waiting for a zIIP as they should flow over to a GCP if the zIIP is saturated. With parallelism, the goal is to reduce elapsed time at the expense of CPU, or zIIP processing cycles in this case. But the elapsed time was not reduced at all due to the fact that 3 of the 4 tasks queued up against 1 processor.

zIIP capacity and parallelism

- **RMF Spreadsheet Reporter Response delay report**
 - Part of WLM Activity report
- **Lots of unaccounted for time**
 - OMPE accounting
 - This block does not show child task class 2 time
- **SYS1.PARMLIB (IEAOPTxx)**
 - IIPHONORPRIORITY = **NO**
 - 3 parallel tasks waiting for 1 zIIP
 - ZIIPAWMT also important here if delays seen in I/O in DB2 10
 - This becomes very important in DB2 10 and DB2 11



CPU delay at about 33%, and the zIIP suspense time at 34%.



CLASS 2 TIME DISTRIBUTION	
CPU	=====> 11%
SECPU	
NOTACC	=====> 37%
SUSP	=====> 19%

QUERY PARALLEL .	TOTAL
MAXIMUM MEMBERS	N/P
MAXIMUM DEGREE	4
GROUPS EXECUTED	10
RAN AS PLANNED	10
RAN REDUCED	0

Figure 9-8 zIIP constrained parallelism

To remedy this situation, the customer needed at least 2 zIIPs in order to have any type of parallelism, and based on the duration of the queries, 3 zIIPs would have been perfect for 3-way parallelism, that is, the 3 out of 4 tasks that were zIIP eligible. Or the customer could have changed to IIPHONORPRIORITY=YES and allowed some of the parallel queries to run against GCPs.

ZIIPAWMT

This parameter is the amount of time the zIIP engines will wait before asking for help from the GCPs. The alternate wait management setting is how z/OS encourages eligible work to run on a zIIP. If there was equal possibility of the zIIP eligible work being dispatched to a GCP then customers would need an equal number of GCPs and zIIPs to get even a 50% offload.

The minimum setting for this parameter is 1600 microseconds or 1.6 milliseconds. The lower the setting, the more responsive the dispatcher is in asking for help from the GCPs.

DO NOT raise this number in an attempt to get more work to flow to a zIIP as it can affect the dispatcher's algorithms, especially in hiper dispatch mode. Generally, under the guidance of z/OS L2, this number can be adjusted down to make the dispatcher more reactive towards zIIP queueing delay:

- ▶ The default for HIPERDISPATCH=YES is 3200.
- ▶ The default for HIPERDISPATCH=NO is 120000.

9.4.2 zIIP capacity in DB2 10 and DB2 11

Starting with DB2 10, we are allowing system agents to be zIIP eligible. Prefetch and deferred write are now zIIP eligible. The negative effects of a lack of zIIP capacity are much more noticeable with this. Even with IIPHONOPRIORITY=YES allowing zIIP eligible work to flow to a GCP, queued threads or system engines are affected by the ZIIPAWMT delay before they are re-dispatched to a GCP.

Since the system agents will be running at the priority of the DBM1 address space, they will be higher in dispatching priority and will be better off than the lower importance distributed agents.

But zIIP capacity should be closely monitored to avoid the queueing delay depicted in Figure 9-7 on page 118. A few customers who were severely constrained on zIIP capacity saw noticeable increases in Class 3: Other Read I/O. This was because the prefetch engines and asynchronous I/Os were waiting to be dispatched on a zIIP (usually seen if only 1 zIIP is installed).

A temporary solution is to lower ZIIPAWMT as much as possible, but the real answer is to have a minimum of 2 zIIPs. Since there are 600 prefetch engines, it is easy to see how deep the queue depth could get.

In DB2 11, log writes and log prefetch are also eligible (at the MSTR address space dispatching priority), so increases in LOG Write I/O, and filling up of the output log buffer might be seen with a shortage of zIIP cycles. This means that if the writing out of the log records is delayed, then the records will remain in the output buffer longer and possibly fill it up. Occurrences of “unavailable output buffers” seen in the statistics report should be treated very seriously and the OUTBUFF ZPARM should be increased immediately.

9.4.3 Conservative zIIP utilization

If you are trying to configure your system to minimize dispatching delay for zIIP eligible work when you have 1 zIIP, it should not be driven above 30% utilization; 2 zIIPs not above 60%, and 3 zIIPs not above 70%. You certainly can drive them higher if you choose, but realize that the dispatching delays for the zIIP, when there are 2x-4x more general purpose engines on the LPAR, can become noticeable. These numbers correspond with Markov’s equation and single processor delay times, where the queueing delay increases at a much higher rate than the processor utilization (i.e. above 30% for a single engine). Otherwise there could be performance implications for the work waiting to be dispatched to a zIIP.

How do I know if I am being affected by the ZIIPAWMT parameter or if I am exposed to zIIP capacity constraint? As long as IIPHONOPRIORITY=YES, then spikes in the occurrence of APPL% IIPCP indicate that the “needs help algorithm” is working. In fact, if you set IIPHONOPRIORITY= NO in DB2 11, DB2 will disable the system engines from being zIIP eligible. This is done in order to protect subsystem performance and data integrity due to the implications of the system tasks becoming non-dispatchable with large zIIP delays.

There are cases where APPL% IIPCP might show up in a steady state, single digits, due to local storage latches which cannot be offloaded to the zIIP, but these cases are not common. This refers to a local storage latch in z/OS, not DB2, where many concurrent threads in an address space are doing storage getmains.

9.5 Getting more zIIP eligibility

Increasing the amount of any of the workloads in Figure 9-2 on page 113 will obviously increase the amount of zIIP eligible cycles in the environment:

- **Parallelism:**

Roughly 80% of the child tasks are eligible for the zIIP, and up to 55-60% of the parent task is eligible if invoked remotely.

Notice that as true for parallelism in general, this will cost extra GCP time as well, but the zIIP offload should offset this cost.

- **Type 2 versus Type 4 connections to DB2:**

This can be a complex topic, and it really does depend. But in order for a workload to be zIIP eligible, it must be DRDA compliant and come through the DIST address space.

If you have JAVA processes running on the same LPAR as DB2 for z/OS, the most efficient choice is the T2 connection, which is defined in the data source, and uses the IBM Data Server Driver code that resides in a UNIX system services file structure. A T2 connection utilizes the Resource Recovery Services attach mechanism.

There is inherent overhead in using the T4 connection, as it must cross the DIST address space entering and leaving DB2. There is task switching to get onto an enclave, a WLM service class must be set up to properly manage it, and there should be enough information (such as client accounting information) in the thread to be able to identify and tune it if necessary.

Tip: You need to internally benchmark which type of attach is more efficient for a specific application; these measurements were done in our lab environment.

If the SQL is highly transactional in context, contains simple SQL, returns less than 50-100 rows, and follows good SQL/coding practices, then the T2 will actually consume less class 1 CPU time, starting with DB2 10. The administration is also easier as the AUTHID and correlation ID will likely match the process that made the request, and the WLM service class will be simpler to set up and maintain.

If the SQL is 4GL in nature, uses an SQL generator, cannot be controlled or tuned by the DBAs, is complex in nature, and has large result sets, then the amount of CPU cycles saved by using a Type 4 driver and a zIIP may win the class 1 chargeable CPU time competition. Also if there are high transaction rates of “frivolous” DB2 calls, for instance, polling a table every second to see if there are any new rows in it, the sheer volume they generate could sway the CPU numbers in favor of T4 as well. But if you add up the Class 1 TCB and zIIP time, the T2 driver will always win.

- **Moving to DRDA and TCP/IP:**

Many customers with inter-subsystem or inter-group communication are still utilizing SNA protocol. Since Private Protocol support was removed in DB2 10, that is one hurdle out of the way. But aside from the fact that simple SNA, or SNA over TCP/IP SQL requests are not zIIP eligible, customers using it have been incurring a CPU overhead since DB2 9. DB2 added the 64-bit shared addressing between DBM1 and DIST TCP/IP processing actually improved while SNA communication degraded. This is because SNA only operates in 31-bit mode (as well as being stabilized several years ago), so in order to work with the new shared storage DB2 needs to copy the information above the bar.

Tip: The accounting information is still reported under the CONNTYPE of DRDA and could be aggregated to estimate CPU savings. If you can uniquely identify these transactions coming from the other DB2s by LU NAME/CORRNAME, then you could add the CPU time used during a peak time and assume that around 60% of that time could be moved to a zIIP if those SNA calls were moved to use TCP/IP.



Buffer pools and group buffer pools

DB2 buffer pools are a key resource for ensuring good performance. This is becoming even more important as the difference between processor speed and disk response time for a random access I/O widens in each new generation of processor. With the decrease in cost of central storage, the opportunity to allocate more of this storage to buffer pools increases. General tuning goals include reducing the frequency of synchronous I/O suspensions, and lowering Other Read I/O suspension as well as Other Write I/O suspension time.

This chapter provides guidance on local and group buffer pool practices. Tuning based on statistics and basic object placement is critical for local buffer pools. Regarding group buffer pools, we describe what conditions to look out for as well as some insight on tuning and sizing based on different workloads.

This chapter contains the following topics:

- ▶ Local buffer pools
- ▶ Group buffer pools

10.1 Local buffer pools

Buffer pool tuning can be considered as much of an art as a science. It is based on the relative trade-off of real storage resources consumed versus the I/O savings return from enlarging the buffer pool. In recent years, the cost of central memory for the z196 machines and later models has dropped substantially from their predecessors, and this opportunity for performance enhancements should be investigated.

Tip: See *DB2 9 for z/OS: Buffer Pool Monitoring and Tuning*, REDP-4604 for details on the buffer pool residency times, as well as queries to run against the OMEGAMON PE performance database for further tuning.

10.1.1 Basic placement and settings

For a basic buffer pool strategy, the first key is to separate the catalog/directory table objects from the other application objects. Unfortunately, the default buffer pool choices for users is still based on BP0 and other buffer pools which the catalog and directory will end up in as well. Installation Panel DSNTIP1 still has the following defaults:

- ▶ **DEFAULT 4-KB BUFFER POOL FOR USER DATA ===> BP0**
- ▶ **DEFAULT 8-KB BUFFER POOL FOR USER DATA ===> BP8K0**
- ▶ **DEFAULT 16-KB BUFFER POOL FOR USER DATA ===> BP16K0**
- ▶ **DEFAULT 32-KB BUFFER POOL FOR USER DATA ===> BP32K**
- ▶ **DEFAULT BUFFER POOL FOR USER LOB DATA ===> BP0**
- ▶ **DEFAULT BUFFER POOL FOR USER XML DATA ===> BP16K0**
- ▶ **DEFAULT BUFFER POOL FOR USER INDEXES ===> BP0**

Change these defaults to other buffer pools to protect those pools defined for the catalog and directory. Many vendor applications, including IBM, create objects in BP0 as well, and this should be monitored by querying the catalog on a regular basis to see what objects end up there.

BP0 has a decent amount of I/O but does not need to be very large compared to other index and data pools, and it might as well be page fixed. The one time when the size of BP0 can affect system performance is during the DSNTIEN migration job. Temporarily increasing the size of BP0 here can help lower the elapsed time of CATENFM. The end user cannot control the 20 catalog and 3 directory LOB table spaces landing in BP0. Six of these have inline LOBs, so they should very rarely access their LOB data. The rest will likely have limited, specific access outside of Utility work. Here is a sample query to determine the mix of index and data objects in each pool as well as what might be assigned to BP0 other than the catalog and directory tables. See Example 10-1.

Example 10-1 Sample query to determine placement of objects in buffer pools

```
SELECT R.BP_SORT
      , R.BPOOL
      , R.PAGESET
      , R.TYPE
      , COUNT(*) AS COUNT
      , DECIMAL(SUM(R.PAGES_ACTIVE_LEAF), 15, 0)
        AS PAGES_ACTIVE_LEAF
FROM
```



```

( SELECT CASE
    WHEN SUBSTR(S.BPOOL CONCAT ' ', 4, 1)=' '
    THEN INT(SUBSTR(S.BPOOL, 3, 1))
    WHEN POSSTR(S.BPOOL,'K') = 0 THEN
    INT(SUBSTR(S.BPOOL CONCAT ' ', 3, 2))
    WHEN SUBSTR(S.BPOOL CONCAT ' ', 1, 5)
    = 'BP32K' THEN
    140+INT(SUBSTR(S.BPOOL CONCAT ' ', 6, 1)
    CONCAT '0')/10
    WHEN SUBSTR(S.BPOOL CONCAT ' ', 1, 4)
    = 'BP8K' THEN
    100+INT(SUBSTR(S.BPOOL, 5, 1))
    WHEN SUBSTR(S.BPOOL CONCAT ' ', 1, 5)
    = 'BP16K' THEN
    120+INT(SUBSTR(S.BPOOL, 6, 1))
    END AS BP_SORT
    , S.BPOOL AS BPOOL
    , 'TS' AS PAGESET
    , D.TYPE AS TYPE
    , ABS(S.NACTIVEF) AS PAGES_ACTIVE_LEAF
FROM SYSIBM.SYSTABLESPACE S
    , SYSIBM.SYSDATABASE D
WHERE D.NAME = S.DBNAME
UNION ALL
SELECT CASE
    WHEN SUBSTR(I.BPOOL CONCAT ' ', 4, 1)=' '
    THEN INT(SUBSTR(I.BPOOL, 3, 1))
    WHEN POSSTR(I.BPOOL,'K') = 0 THEN
    INT(SUBSTR(I.BPOOL CONCAT ' ', 3, 2))
    WHEN SUBSTR(I.BPOOL CONCAT ' ', 1, 5)
    = 'BP32K' THEN
    140+INT(SUBSTR(I.BPOOL CONCAT ' ', 6, 1)
    CONCAT '0')/10
    WHEN SUBSTR(I.BPOOL CONCAT ' ', 1, 4)
    = 'BP8K' THEN
    100+INT(SUBSTR(I.BPOOL, 5, 1))
    WHEN SUBSTR(I.BPOOL CONCAT ' ', 1, 5)
    = 'BP16K' THEN
    120+INT(SUBSTR(I.BPOOL, 6, 1))
    END AS BP_SORT
    , I.BPOOL AS BPOOL
    , 'IX' AS PAGESET
    , D.TYPE AS TYPE
    , ABS(I.NLEAF) AS PAGES_ACTIVE_LEAF
FROM SYSIBM.SYSINDEXES I
    , SYSIBM.SYSDATABASE D
WHERE D.NAME = I.DBNAME
) AS R
GROUP BY R.BP_SORT, R.BPOOL, R.PAGESET, R.TYPE
ORDER BY R.BP_SORT, R.BPOOL, R.PAGESET, R.TYPE
WITH UR;

```

Table 10-1 shows an example of a fairly simple buffer pool strategy to separate data, indexes, LOBs, catalog and directory objects, and workfiles.

Table 10-1 BP setup examples

BP #	VPSEQT	DWQT	VDWQT	Page steal algorithm
BP0 cat/dir	80	30	5	LRU
BP1 data	80	10	1	LRU
BP2 indexes	80 --> 50 (V11with reclassification will allow this)	30	5	LRU
BP3 large NPIs	80	30	0	LRU
BP7 workfile	98	50	30	LRU
BP8 in-memory	0	50	50	NONE
BP8k0 cat/dir	80	30	5	LRU
BP16k0 cat/dir	80	30	5	LRU
BP32k cat/dir	80	30	5	LRU
BP32k1 LOBs	95	5	1	LRU
BP32k2 workfile	99	50	30	LRU

LOBs

LOBs represent a unique tuning opportunity in DB2. The way they are stored, in auxiliary tables, and the way they are accessed using list prefetch, if they occupy more than one page, necessitate different buffer pool thresholds. In V8 and prior, when a thread wanted to reuse space in a LOB object, it would get a LOB lock on the object and reuse the space as long as the previous delete was committed. This separate LOB lock that could add to IRLM CPU time, and by removing it with locking protocol 3, we saw performance improvements by avoiding lock escalations with LOBs.

But in DB2 9, we no longer use the LOB lock for this, and rely on the read LSN from the oldest reader in the buffer pool. So if a thread deletes a LOB at time X and commits at time X+1, the inserter cannot reuse the space if the oldest read claim is earlier than time X. It becomes more important to make sure that the oldest read claim keeps moving up, by splitting LOBs into separate buffer pools. If you have applications that do not commit on a regular basis, it will necessitate separating catalog directory LOBs from application data LOBs.

The effects of this were seen in the growth of the SPT01 catalog table in DB2 10 due to the package statement being stored in a LOB (non-inline piece) when user objects with long commit scopes were resident in BP0. These LOB buffer pools should have a relatively high VPSEQT (95%) and be sized to take advantage of the improved prefetch quantities in DB2 9 and up. The DWQT and VDWQT are set low again to trickle the writes out, as normally when a LOB row (maximum of 1 row per page) is updated, it is not re-referenced, so we should free that page up for other work by writing the changed pages out quickly. Of course, this can affect write efficiency in some cases if the levels are very low, which will be discussed later.

VDWQT and DWQT

In general, the reason we prefer that VDWQT is hit prior to DWQT, and the default is always lower, is an attempt to lower the number of I/Os done and avoid a sudden flood of writes. If the number of dirty pages are spread out over multiple data sets, then there will likely be more I/Os if we rely on DWQT.

The optimal (maximum) deferred write amount is 128 pages, which will be done in 4 I/Os. Each I/O can have a max of 32 pages in it. This is much more likely to occur if the dirty pages are in the same page-set; whereas if DWQT is hit, then there will likely be multiple I/Os of less than 32 pages. If you have specified something such as 0,[# of pages] DB2 will wait until that number is reached before writing them out. There must be at least 40 pages eligible to be written before DB2 will start writing them out. So there is no point in specifying (0,20) for example, as we will wait until there are at least 40 dirty pages.

If DWQT is hit, then DB2 will write out enough pages to get below 10% of the DWQT threshold (in increments of 128 pages). If your buffer pool is 2M pages, then 20,000 pages could be quite disruptive to the DASD subsystem, depending on how your devices are spread out. If the VDWQT is triggered, DB2 writes up to 128 pages of the data set and continues to do so until the number of dirty pages drops below VDWQT. This trickle effect is less disruptive to the disk back-end as well as to the group buffer pools, since the changed pages would go there if the object was group buffer pool dependent.

Tip: Starting in DB2 10, the root pages of the indexes are fixed in the buffer pools. These pages, since they are not stealable, would count against the VDWQT threshold for that object as well as the DWQT threshold for that buffer pool. One page does not seem that drastic unless you cohabitate thousands of indexes in the same “index buffer pool.” This may be especially relevant in a test or development system with much smaller buffer pools or deferred write thresholds. A simple query against the catalog to determine the number of index objects in their respective buffer pools can give you an estimate of how many more “dirty” pages you will have in the buffer pool once in DB2 10 Conversion Mode.

Setting VDWQT and DWQT too low may result in poor write caching, writing the same page out many times causing more OTHER WRITE I/O, short deferred write I/Os, and increased DBM1 SRB CPU resource consumption. If you want to set VDWQT in pages, do not specify anything below 128 in order to keep the best write and I/O efficiency.

10.1.2 Thresholds

There are several critical thresholds that are reported in the DB2 statistics and/or the **DISPLAY BUFFER POOL** command.

Tip: These three thresholds in particular should be avoided even during times of system stress and high workload volume.

- ▶ Sequential Prefetch Threshold (although it applies to *any prefetch*) should be minimized, and if it was seen due to intentional tuning down of VPSEQT, then the offending objects or access paths should be investigated. This may be non-zero if you have set VPSEQT=0 as well:
 - Reported in statistics field PREF.DISABLED-NO BUFFER >0
- ▶ Data Management Threshold should be kept at '0':
 - Reported in statistics field DM THRESHOLD > 0
- ▶ Immediate write threshold is not reported directly, but if the Data Management threshold was hit, you only have 2.5% more of the buffers available before this is hit. If immediate write is triggered, you will see the DM threshold at a non-zero number AND very elevated synchronous writes:
 - SYNCHRONOUS WRITES > 'normal'
During typical operation, this will only occur if the changed page has been in the buffer pool more than 2 system checkpoints, or you are out of deferred write engines:
 - DM THRESHOLD > 0

Starting from the least severe and working our way up to the most severe, the prefetch threshold disables DB2 from using *any type* of prefetch if over 90% of the pages in the buffer pool are "dirty."

There are some legitimate reasons for seeing PRE-DISABLED NO BUFFER in the statistics report. For instance, you may have disabled prefetch using VPSEQT=0, or you have run up against whatever VPSEQT setting you currently have. These would be user controlled, but hitting the threshold of 90% dirty pages is not where you want to be.

Next is the Data Manager Threshold, which is 95% of VPSIZE. If 95% of the buffer pool becomes non-stealable, sequential threads that process multiple rows in the same page will start to do one getpage per row. This causes an obvious CPU increase, and it would severely impact applications using sequential access as well as elongate lock and latch times seen for updaters.

The last and highest threshold is the Immediate Write threshold at which point prefetch has been disabled, you are doing a getpage and release page for every row accessed, and to top it off, you are doing synchronous writes. The Immediate Write threshold is triggered when 97.5% of the buffers are non-stealable. So if the application is updated multiple rows in a page, it must do a getpage for each one, and risk having another application jump in and get the lock it needed depending on the lock size, as well as waiting for a synchronous I/O for each row it updates. The fallout from just one buffer pool hitting this threshold will be felt subsystem or group wide. See Figure 10-1.

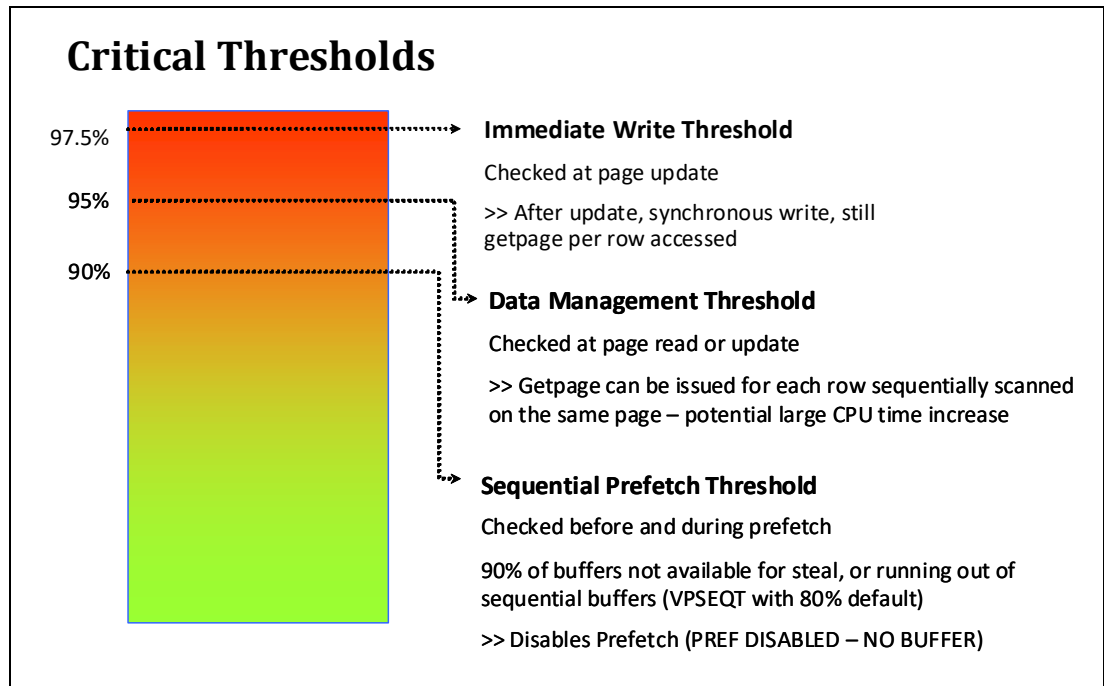


Figure 10-1 Buffer pool critical thresholds

10.1.3 Tuning

The raw size of the buffer pool is probably the most important aspect, and can even mask poor choices in VPSEQT and object placement. But getting an exact number as a starting point, with no data, is next to impossible. Buffer pool performance is for the most part incremental and relative. You have to choose a baseline to begin from, and then look to improve items such as residency time or hit ratio on an iterative basis.

Adding more memory has its cost, but it allows defining a bigger buffer pool and it could reduce the number of sync I/Os. If the number of I/Os is reduced, the CPU time will be less, and hence the cost of software usage fees is less.

Comparing across different buffer pools may be unrealistic for the most part, as the different objects and access paths may limit the hit ratio for instance. Unless you are aiming at something tangible such as an in-memory buffer pool with '0' I/Os, the job is never finished. The result will be incremental improvement, or simply maintaining the metrics you currently have by defending against access path changes, new applications, and new objects. Incremental improvement may be attained by moving objects around and or increasing the size of the buffer pool. As you increase the size, synchronous I/Os will lessen to a point and can save CPU and elapsed time in the queries. The law of diminishing returns applies here, and eventually the cost of storage will outweigh the next I/O saved.

In order to get the most efficient prefetch possible (especially for table spaces), you will need to reach a minimum buffer pool size.

Minimum size of buffer pool for larger prefetch starting with DB2 9:

- ▶ Large buffer pool defined as VPSEQT * VPSIZE > 160MB (320MB for Utilities):
 - VPSIZE is done in # of buffers, so you need to multiply that by the page size (4,096 for 4k page) to get the storage size.
- ▶ Max sequential prefetch of 128KB in V8 → 256KB in DB2 9 for a table space scan:
 - 256KB in V8 → 512KB in DB2 9 for utility operations.

If the buffer pools are smaller than 900 KB, then the prefetch for SQL is 32 KB; if less than 4 MB, the prefetch for SQL will be 64 KB; and if it is at least 4 MB, then the prefetch quantity is 128 KB. The relative prefetch for utilities is double the amounts.

Other than monitoring for and avoiding the conditions mentioned Figure 10-1 on page 131, the buffer pool tuning strategy varies considerably across customers. There are basically two metrics that help rate the efficiency of a buffer pool. One is the hit ratio, which measures how many times the page the application was looking for was in the pool versus having to go out to disc. The other is the page residency time, meaning that after a page is brought in, how long does it reside there before being stolen or written out. These metrics can be derived using a Statistics report and based on the interval the report covers. See Example 10-2.

Example 10-2 OMPE buffer pool statistics

SYNCHRONOUS READS	1183.1K
SYNCHRON. READS-SEQUENTIAL	3421.00
SYNCHRON. READS-RANDOM	1179.6K
GETPAGE PER SYN.READ-RANDOM	451.18
SEQUENTIAL PREFETCH REQUEST	2958.00
SEQUENTIAL PREFETCH READS	1515.00
PAGES READ VIA SEQ.PREFETCH	36068.00
S.PRF.PAGES READ/S.PRF.READ	23.81
LIST PREFETCH REQUESTS	0.00
LIST PREFETCH READS	0.00
PAGES READ VIA LIST PREFETCH	0.00
L.PRF.PAGES READ/L.PRF.READ	N/C
DYNAMIC PREFETCH REQUESTED	2162.9K
DYNAMIC PREFETCH READS	346.3K
PAGES READ VIA DYN.PREFETCH	9256.2K
D.PRF.PAGES READ/D.PRF.READ	26.73
PREF.DISABLED-NO BUFFER	0.00
PREF.DISABLED-NO READ ENG	0.00
PAGE-INS REQUIRED FOR READ	0.00

Buffer pool equations:

Hit Ratios = percentage of times the page was found in the BP:

- ▶ System HR = (Total getpages – Total pages read) / Total getpages * 100
 - Total getpages = random getpages + sequential getpages
 - Total pages read = synchronous reads for random getpages + synchronous reads for sequential getpages + pages read via sequential prefetch + pages read via list prefetch + pages read via dynamic prefetch
- ▶ Application HR = (Total getpages – Synchronous reads) / Total getpages * 100:
 - Total getpages = random getpages + sequential getpages
 - Synchronous reads = synchronous reads for random getpages + synchronous reads for sequential getpages

An overall hit ratio *over 90%* is a good goal, but varies across intervals and workload dependent.

Residency Time = average time that a page is resident in the buffer pool:

- ▶ System RT (seconds) = VPSIZE / Total pages read per second
Total pages read = synchronous reads for random getpages + synchronous reads for sequential getpages + pages read via sequential prefetch + pages read via list prefetch + pages read via dynamic prefetch.
A system residency time of *over 60* seconds is a good goal.
- ▶ Random page residency time (secs) = Max(System residency time, (buffer pool size * (1-VPSEQT/100) / sync pages read per second)
- ▶ Sequential page residency time (secs) = Min(System residency time, (buffer pool size * (VPSEQT/100) / async pages read per second)

These may sound like obscure goals, but unless you have segregated your buffer pools by application, then neither the hit ratio nor the absolute value of pages read in can predict the affect on application elapsed time by improving it. There is also no direct correlation between the amount of VPSIZE increase and hit ratio due to the LRU algorithm. Here the most frequently hit pages are already resident, so there is the law of diminishing returns working against you, as you may increase the buffer pool by 25% with only a 2% increase in hit ratio (for example).

However, trending the effect as you add buffers to the pool can be very effective. Using increments of 50 MB or so, when the next chunk of memory you add does not reduce the number of I/Os, you may have reached the top of the curve. Each time you lessen the number of I/Os (pages read), you are affecting the hit ratio (although maybe not by an entire percentage point). Increasing VPSEQT can increase the sequential page residency, where as decreasing it could help the random page residency time. Increasing the VPSIZE could help both. If you have the objects segregated by access type, then monitoring the random and sequential residency times would offer quantifiable metrics for you to judge improvement.

One example of a logical sizing that can be done for index buffer pools would be to ensure that the non-leaf pages remain resident in the buffer pool. See Example 10-3.

DB2 10 supports parallel read I/Os when 3 or more indexes are affected by an insert statement. In most cases, to determine the candidate page for a new row, DB2 probes the cluster index to look the new key or the next lower key (prior to DB2 10, it would have searched for the next higher key.) That search is done synchronously prior to probing the secondary indexes. Then DB2 searches secondary indexes and if Insert I/O Parallelism is used, DB2 schedules a prefetch I/O to read the leaf pages. Hence, if there are three or more indexes, DB2 uses prefetch to achieve I/O parallelism.

However, in the case of Append, Member Cluster, or a hashed table, even the cluster index is treated like a secondary index, so I/O parallelism could be used with just 2 indexes. Therefore, it is very important to avoid non-leaf page I/O. If there are a manageable number of indexes in the pool, or some very large important NPIs, then you could use this rough estimate to know the minimum number of buffers that need to be present for the frequently referenced non-leaf pages.

Example 10-3 Calculating size of index buffer pool

Assume 1M rows and 100 index entries per page,
1M/100 = 100,000 leaf pages
100,000/100 = 1,000 L2 pages
1,000/100 = 10 L3 pages
10/100 = 1 L4 page
So the number of non-leaf pages, and dedicated buffers would be = 1,000+10+1 = 1,011 for this 4-level index.

Further separation

When you get serious about segregating objects beyond just the table versus index level, some IFCID traces can be quite useful. First you may want to try and segregate them based on access type, whether they are sequential or randomly accessed. IFCID 198 shows down to the individual page level if it was accessed randomly or sequentially and if it was refreshed by pulling the page from the GBP (XI-data found, NF-data found statistics) or from DASD.

If you simply want to be able to see which objects are causing the sync I/O and the related delays in that buffer pool then IFCID 199 would be necessary. See Figure 10-2. This could be very useful for determining if you were able to pin a particular object in a buffer pool, or which objects are causing the most I/O delay for you from a buffer pool level.

If there is a troublesome object, such as a large NPI on a journal table, the application may access it skip-sequentially for updates at the end of the day. This object will never be resident as its pages are not re-referenced enough, and it just eats up useful sequential pages in the buffer pool since prefetch will likely be triggered frequently. This could be a candidate to split out of the general index buffer pool, and in with other objects that cycle through the buffer pool without being re-referenced.

The other reason for separation is to pin an object or objects in a buffer pool because they are highly re-referenced. Aside from the I/O savings that then equate to elapsed time savings, there are also associated CPU savings: disabling prefetch [VPSEQT(0)] and the LRU algorithm [PGSTEAL(NONE)] and save latch class 14, latch class 24, and prefetch scheduling. See “VPSEQT sequential steal threshold”.

Dataset Statistics for I/O Tuning

- Statistics class 8 (IFCID 199)

BPOOL	DATABASE SPACENAM PART	TYPE GBP	SYNCH I/O AVG ASYNC I/O AVG ASY I/O PGS AVG	SYN I/O AVG DELAY SYN I/O MAX DELAY	CURRENT PAGES (VP) CHANGED PAGES (VP) NUMBER OF GETPAGES
BP10	DB1 EMP 30	TSP N	23.35 0.01 32.00	8 78	3433 0 N/A 2868
BP11	DB2 EMPIX 36	IDX N	102.59 4.04 5.98	1 35	18991 74 N/A 245586

Count of
Sync I/O
per second

Average
Sync I/O
(ms)

Figure 10-2 IFCID 199

VPSEQT sequential steal threshold

The sequential steal threshold sets the number of buffers that can be held by sequentially accessed buffers before the least recently used algorithm starts going after them ahead of the random pages. If the threshold is not hit, then both randomly accessed and sequentially accessed pages are treated equally. This is also the case if VPSEQT is set to 100%, then all pages are treated equally on the LRU chain. We do not talk about first-in-first-out here because it is considered a special case, used mainly when the objects are pinned in a buffer pool, in which case PGSTEAL(NONE) in DB2 10 and DB2 11 is an even better option.

The purpose of a lower VPSEQT was to protect random pages, as we want to avoid synchronous I/O as much as possible. There was a significant change in DB2 9 though, that affected the number of pages that ended up being classified as sequential. Prior to DB2 9, if a page was read in by list prefetch or dynamic prefetch and later touched by a random getpage request, then it was re-classified as random.

In DB2 9 and 10, these buffers remained classified as sequential, because that is how they were brought in. In short, this meant that pages in your pool that were re-classified prior to DB2 V8 were not subject to the sequential steal threshold. If you had tuned down your VPSEQT setting to protect the random pages, you may have seen an increase in sync I/O in DB2 9 and up.

DB2 10 introduced row level sequential detection (see *DB2 10 for z/OS Technical Overview*, SG24-7892) which caused dynamic prefetch to be triggered earlier in many cases, since now in just 2 getpages, we might trigger dynamic prefetch. This is because after getting 5 out of 8 sequential rows, if the next row was on a separate page, Dynamic Prefetch would kick in. If VDWQT was tuned down to protect randomly accessed pages, this could also increase synchronous reads for sequential pages. For the most part, customers increased VPSEQT or enlarged their buffer pools to deal with this. DB2 11 will not trigger dynamic prefetch until the at least the 3rd getpage.

DB2 11 basically goes back to the old re-classification strategy, where if a page brought in by dynamic or list prefetch is requested by a random getpage at a later point, we will pull that buffer off of the sequential LRU chain, and classify it as random. This means the VPSEQT(80) setting could be reduced, especially for data buffer pools with good residency time, once you get to DB2 11.

Disabling prefetch via VPSEQT(0) may be useful for objects completely resident in the BP. It avoids unnecessary prefetch SRB scheduling. Remember that DB2 may schedule a prefetch engine even if all the pages end up being resident in the buffer pool, because we know the pages are sequential. Sequential prefetch is always avoided until the first buffer pool miss, but not Dynamic Prefetch. Up to a 10% overall CPU time reduction could be possible. Latch class 24 in Buffer Manager can also be decreased. But parallelism will also be disabled when VPSEQT(0), so you need to ensure that it will not affect the access paths that have been chosen. Since the objects here are assumed to be in buffer pool in entirety, the use of FIFO (First In First Out) or PGSTEAL(NONE) buffer steal algorithm can reduce CPU time for LRU chain maintenance as well as latch class 14 (LRU latch) contention.

Important: DB2 uses a most recently used (MRU) page steal algorithm for sequential buffers in the buffer pool. This means that if the pages are not 100% resident in memory, the pages touched the most may be stolen for the small percent of pages not held in memory. So only set VPSEQT(0) after you have changed the page steal algorithm to NONE or FIFO and make absolutely sure there is no read I/O. In DB2 10 and 11, NONE performs as if VPSEQT(0).

In DB2 11, there are some new counters in the statistics report that can help you determine the amount of sequential pages in the buffer pool as well as how close to the VPSEQT threshold it stands. See Example 10-4.

MAX BUFFERS ON SLRU would tell you the high water mark for sequentially accessed pages in that pool, so that number divided by the VPSIZE would let you know the theoretical VPSEQT percentage that would treat all of the pages equally in the buffer pool. If there is a non-zero number in the SLRU LENGTH EQUALS VPSEQT, then that buffer pool has hit VPSEQT and at that point the most recently used sequential pages will be stolen before any pages classified as random. If this occurs too quickly, then prefetched pages read in may be stolen before the getpage occurs for them.

Example 10-4 DB2 11 sequential page counters

MIN BUFFERS ON SLRU	12735.68	N/A	N/A	N/A
MAX BUFFERS ON SLRU	12736.02	N/A	N/A	N/A
SLRU LENGTH EQUALS VPSEQT	0.00	0.00	0.00	0.00
GETPAGE REQU RANDOM ON SLRU	14.00	14.00	0.56	0.08

Tip: If there are a large number of synchronously read sequential pages, meaning after a page was prefetched it was stolen before the getpage was issued for it, then there may be reason to increase VPSEQT. Seeing SYNCHRON. READS-SEQUENTIAL and VPSEQT being reached in the same interval means it is time to increase VPSEQT or the VPSIZE.

Since there are 600 prefetch engines within DB2, and their largest prefetch amount is 512 KB with Utilities, then as long as the $VPSIZE \times VPSEQT = 320MB$ [$512k \times 600$] you should be able to avoid most of the SYNCHRON. READS-SEQUENTIAL. Check and ensure that there is enough room for the improved prefetch size. If you are not worried about Utilities, just SQL then you would need 256 KB per prefetch engine and 160 MB for $VPSIZE \times VPSEQT$.

Tuning VDWQT

Increase the value for these settings:

- ▶ In-memory index or data buffer pools
- ▶ Fewer pages written per unit of time
- ▶ Reduced frequency of same set of pages written multiple times
- ▶ Reduced page latch contention (when page is being written, first update requestor waits on “other write I/O”, subsequent updaters wait on page latch). The page latch suspension time is reported in the field PAGE LATCH in the accounting report.
- ▶ Reduced forced log writes (log records must be written ahead of updated pages)
- ▶ Random insert, update, or delete operations

Decrease the value for these settings:

- ▶ Reduced total of updated pages in pool so more buffers are available for stealing
- ▶ Reduced burst of write activity at checkpoint intervals
- ▶ More uniform and consistent response time
- ▶ Better performance of the buffer pool as a whole when writes are predominantly for sequential or skip sequential insert of data
- ▶ For data sharing, free page p-lock sooner, especially NPI index page p-lock

Page-fix

Page fixing of buffers was introduced in DB2 V8. Its value comes from the CPU savings seen by not having to fix and unfix the real 4 KB frames in memory during an I/O. Simplistically, we need to ensure that there is a real frame to put the 4 KB page (buffer) of virtual storage before the getpage. If you are short on real storage and the frame was paged out to AUX, then it really does represent 2 sync I/Os. This is a case you should try to avoid, and indicates that the buffer pools may be oversized for the environment. The statistics report will show PAGE-INS REQUIRED FOR READ/WRITE if this scenario occurs.

Alternatively, if there is ample real storage to back the buffer pools, then page fixing them should be strongly considered. The lab saw up to 4% performance improvement in the IRWW workload by page fixing all of the buffer pools (depending on I/O intensity), which was documented in the *DB2 V8 Performance Topics*, SG24-6465. When DB2 10 was released, the baseline savings reported “out of the box” included page fixed buffer pools backed by 1 MB frames (available with the IBM z10™ and up). See “Real storage” on page 140.

In order to take advantage of this performance improvement, the buffer pools must be completely backed by real storage because if we are even one buffer short, then the performance benefit is lost.

So how do you decide, based on the available real storage, if you cannot page fix all of them, which buffer pools are the best candidates for page fixing? I/O intensity is the answer. Take the number of pages read (sync and all prefetches) plus the number of pages written, and divide by the number of buffers (Figure 10-3). The reason I/O intensity is so important goes back to the page fix and unfix DB2 needs to do for each I/O. In Figure 10-3, BP32k is the obvious choice with the highest I/O intensity. The actual number here does not matter it is just for relevance. Later we will discuss the benefit of the 1 MB frames in z/OS which for the most part have PGFIX(YES) as a prerequisite.

Long-Term Page Fix for BPs with Frequent I/Os								
Recommended for BPs with high I/O intensity								
— I/O intensity = [pages read + pages written] / [number of buffers]								
— Relative values across all BPs								
BPID	VPSIZE	Read Sync	Read SPF	Read LPF	Read DPF	Read - Total	Written	I/O Intensity
BP0	40000	2960	0	0	6174	9134	107	0.2
BP1	110000	12411	5185	0	1855	19451	6719	0.2
BP2	110000	40482	19833	11256	9380	80951	5763	0.8
BP3	75000	23972	6065	0	14828	44865	7136	0.7
BP4	80000	22873	45933	3926	50261	122993	3713	1.6
BP5	200000	0	0	0	0	0	0	0.0
BP8K0	32000	9	11	0	11	31	169	0.0
BP32K	2000	693	873	0	6415	7981	38	4.0
<i>In this example: Best candidates would be BP32K, BP4, BP2, BP3 No benefit for BP5 (data in-memory)</i>								

Figure 10-3 PGFIX(YES) candidates

So how can you convince the MVS sysprog and capacity team that the DB2 buffer pools deserve this storage, and that there are actual CPU savings to be had from it? The reason the PGFIX parameter was derived was to avoid the CPU cost of page fix and page unfix during an I/O. The cost of page fix/unfix for prefetch or deferred writes is reported in the DBM1 address space SRB CPU time. For sync I/Os, it is reported in Class 2 CPU time of the application which caused the sync I/O. In the IRWW workloads there was a 37% decrease in DBM1 address space SRB time and a 4% decrease in class 2 for the applications.

Figure 10-4 and Figure 10-5 represent an actual customer system on DB2 9 with no page-fixed buffer pools. So this would be the most dramatic picture where savings could be had. There are peaks of 300,000-350,000 pages read and written per minute. The page fix and unfix process is estimated to consume about 1,000 instructions. So with $300,000 \times 1,000 = 300$ MIPs. So you could take this rough MIP calculation to your sysprog and determine how many CPU seconds these activities are consuming. In Figure 10-5, we can see that the DBM1 address space is consuming around 70 seconds of CPU time per minute and page fixing will constitute a portion of that time, although prefetch would consume the majority of it.

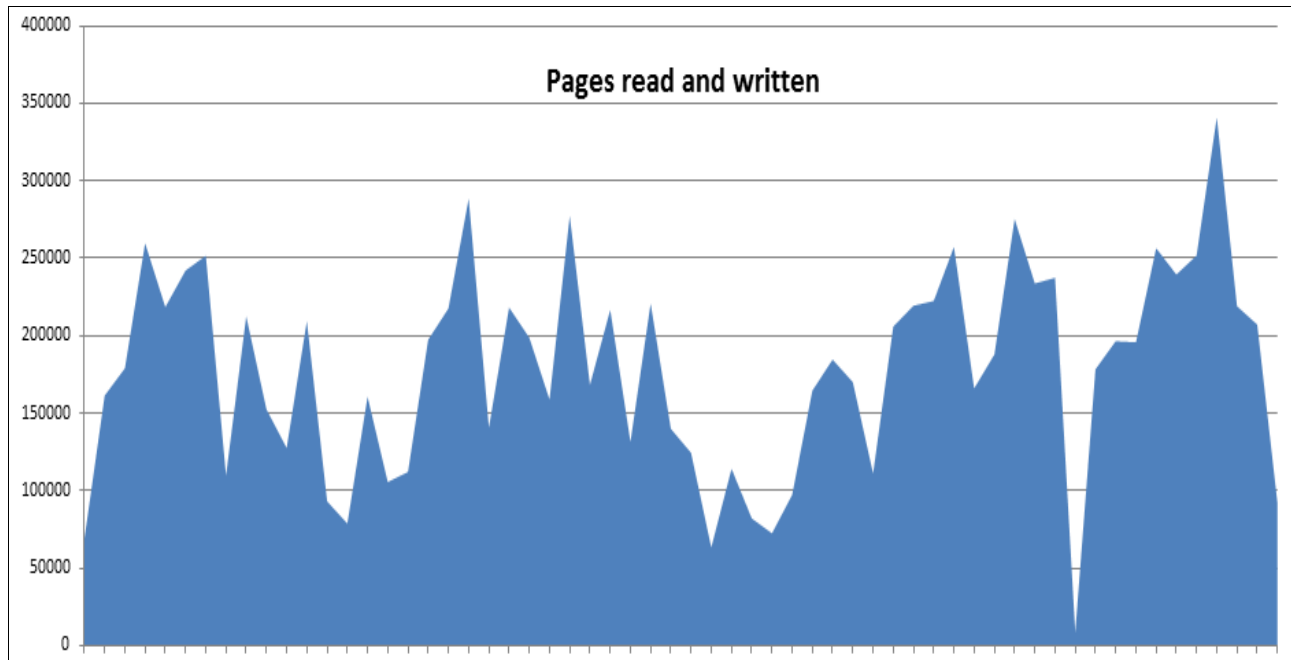


Figure 10-4 Total pages read and written

You can see the correlate in the peaks and valleys between the two charts where prefetch as well as reads and writes drove the DBM1 SRB time.

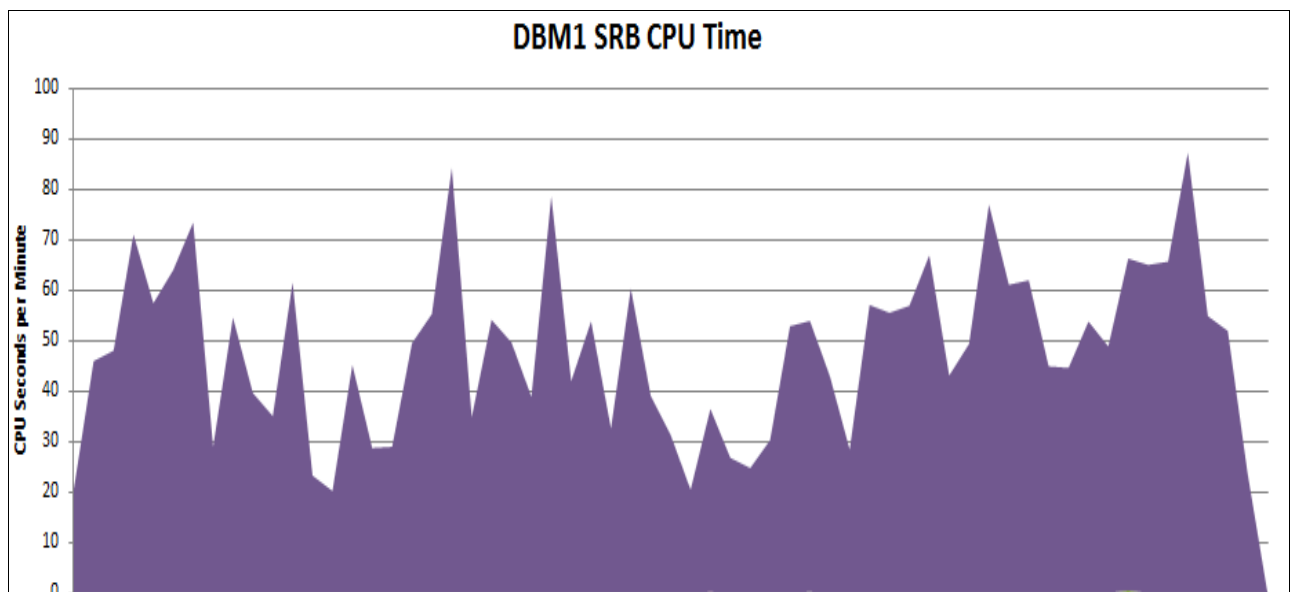


Figure 10-5 DBM1 SRB time in seconds per minute

Now when this customer moves to DB2 10, about 70% of this SRB time will be zIIP eligible because the vast majority of the time graphed is related to prefetch and deferred write activity. By page fixing the portion of the 300 MIP estimate due to fix and unfix related to prefetch, time will be saved on the zIIP processor. Those for sync I/O will be saved in the class 2 CPU for the application. But why waste extra instructions? Whether they are zIIP eligible or not, it makes more sense to avoid these CPU cycles.

DB2 10 increases the opportunity for prefetch to occur with the introduction of row level sequential detection, list prefetch for disorganized indexes, and parallel index I/O (prefetch engine). Some of this detail can be found in the *DB2 10 for z/OS Technical Overview*, SG24-7892. The impact on the zIIP can be found in 9.2.2, “System agents” on page 112.

Remember that the ALTER with PGFIX(YES) only becomes active at the buffer pool’s next allocation, so you can alter the buffer pool VPIZE down to ‘0’, and then alter the PGFIX parameter to yes, or wait until DB2 is recycled possibly. The **-DISPLAY BUFFERPOOL(*) SERVICE=4** Example on page 141 offers information on the buffer pool regarding what page size (4 KB, 1 MB [DB2 10], or 2 GB [DB2 11]) DB2 would like to use to allocate the pool. It will also show how many of the 1 MB frames are actually in use when the buffer pool is allocated.

Real storage

It has always been strongly advised to back the buffer pools 100% with real storage. Why have a buffer pool with data-in-memory if you page to AUX and need to do 2 sync I/Os for each page you are short. That is 1 I/O for the real frame of memory, and 1 I/O for the page of DB2 virtual storage both coming from DASD. In DB2 10, real storage was estimated to increase from 10-30% depending on what percentage of DB2 storage the buffer pools constituted, since they would not increase based on a release level. Section 13.2, “Monitoring real storage usage” on page 181 goes into how to look at this from the IFCID 225 perspective.

From the statistics report, you can see if the buffer pools required pages to be read-in from AUX storage in order to back their read or write requests. After the system is warmed up and the buffer pools are allocated, these page-ins should be close to ‘0’, otherwise there is not enough real storage to back the buffer pool. This also would not occur if the buffer pools were page fixed.

Large frames

Once you have your buffer pools (or at least some of them) page fixed and backed by real storage, then you can look at using the 1 MB large frame support that came with the z10 processors. Here is another CPU savings opportunity. The improvement comes in the TLB or translation look-aside buffer processing, which maps the virtual to real storage. Consolidating 256 individual 4 KB frames into 1 MB frame means a greater likelihood of getting a hit in the TLB. This can be difficult to measure due to the cost savings being spread across the workload in the buffer pool. Customers on DB2 10 saw between a 2-4% CPU improvement at a workload level.

Utilizing these large frames is a consolidated effort across the z/OS team. Support for it should be available on all processors and supported versions of z/OS, but it is not turned on by default. Effectively the MVS team needs to manually carve out a portion of 1 MB frames from the total available storage on the LPAR. In essence, you would sum up the VPSIZE of all page fixed buffer pools, and add another 15-20% cushion for growth. Since defining this LFAREA requires an IPL, you will want to plan as far out as possible for it.

APAR OA31116 in z/OS adds fields to the **DISPLAY VIRTSTOR,LFAREA** command so that users can see the high water mark for 2 GB, 1 MB, and 4 KB frames. See Example 10-5. So when this number gets within 10-15% of the LFAREA defined for 1 MB frames, capacity planning needs to be initiated to ensure that this limit is not overshot.

Example 10-5 MVS DISPLAY VIRTSTOR,LFAREA

```

RESPONSE=SC63
IAR019I 16.11.10 DISPLAY VIRTSTOR 559
SOURCE = DEFAULT
TOTAL LFAREA = 0M , 0G
LFAREA AVAILABLE = 0M , 0G
LFAREA ALLOCATED (1M) = 0M
LFAREA ALLOCATED (4K) = 0M
MAX LFAREA ALLOCATED (1M) = 0M
MAX LFAREA ALLOCATED (4K) = 0M
LFAREA ALLOCATED (PAGEABLE1M) = 0M
MAX LFAREA ALLOCATED (PAGEABLE1M) = 0M
LFAREA ALLOCATED NUMBER OF 2G PAGES = 0
MAX LFAREA ALLOCATED NUMBER OF 2G PAGES = 0

```

Once the PGFIX(YES) (or PGFIX(NO) in DB2 11) and LFAREA parameters have been set, you will want to ensure that your buffer pools are being backed by these 1 MB frames. The DSNB546I message displays the preferred framesize and what is allocated. The preferred field is controlled in the ALTER BUFFERPOOL option described in 10.1.4, “DB2 11” on page 142, so if you wanted a 2 GB frame but there were none available, then the preferred size is 2 GB, but it is actually using 1 MB frames. You can also see how many buffers are backed by 1 MB frames under each individual buffer pool.

Example 10-6 shows a buffer pool that was altered to use 1 MB frames but is not yet allocated so there are 0 1 MB frames being used.

Example 10-6 -DIS BUFFERPOOL () SERVICE=4*

```

DSNB401I -DB1A BUFFERPOOL NAME BP32K1, BUFFERPOOL ID 81, USE COUNT 0
DSNB402I -DB1A BUFFER POOL SIZE = 2000 BUFFERS AUTOSIZE = NO 390
          VPSIZE MINIMUM = 0 VPSIZE MAXIMUM = 0
          ALLOCATED      =      0 TO BE DELETED =      0
          IN-USE/UPDATED =      0
DSNB406I -DB1A PGFIX ATTRIBUTE - 391
          CURRENT = YES
          PENDING = YES
          PAGE STEALING METHOD = LRU
DSNB404I -DB1A THRESHOLDS - 392
          VP SEQUENTIAL   = 80
          DEFERRED WRITE  = 30 VERTICAL DEFERRED WRT = 5, 0
          PARALLEL SEQUENTIAL =50 ASSISTING PARALLEL SEQT= 0
DSNB546I -DB1A PREFERRED FRAME SIZE 1M 393
          0 BUFFERS USING 1M FRAME SIZE ALLOCATED

```

10.1.4 DB2 11

Other DB2 11 buffer pool parameters are examined in this section.

AUTOSIZE enhancements

The AUTOSIZE option on ALTER BUFFERPOOL allows WLM to determine if buffer pool I/O is the predominant delay for a random getpage. If so, it will use histograms to determine whether increasing the buffer pool size could help achieve the performance goal. Depending on the amount of storage available, WLM may instruct DB2 to increase the buffer pool or first decrease another buffer pool and then increase the buffer pool in question. Buffer pool sizes could vary +/- 25% during the auto sizing.

The function was initially shipped as disabled regardless of the AUTOSIZE setting. Then a later PTF from z/OS enabled the feature and some customers who had AUTOSIZE(YES) were caught by surprise. Some customers saw the increases in buffer pool sizes that were page fixed, and thus an increase in the real storage consumed. In order for WLM to be able to shrink the size of one buffer pool to give to another, you need to be on z/OS 2.1.

With DB2 11, we introduce the ability to set a low and high threshold for it with the VPSIZEMIN and VPSIZEMAX parameters. With them you can control how wide a variance WLM can apply, since when DB2 is restarted, the last recorded VPSIZE becomes the new base line and AUTOSIZE could move it another +/-25%. In DB2 11, if you specify AUTOSIZE and leave out VPSIZEMIN/VPSIZEMAX, then the buffer pool may still be altered up to the default +/-25%.

FRAMESIZE

Another enhancement is the ability to choose the FRAMESIZE for the buffer pools. With this option, the user can decide if the buffer pool will be backed with 4 KB, 1 MB, or 2 GB frames as long as they are supported on that z/OS and hardware level. If you specify 2 GB, the buffer pool must be over 2 GB to begin with, and it will then allocate the remainder in 1 MB frames so as not to waste excess 2 GB ones.

If you specify FRAMESIZE(1 MB) and the buffer pool is PGFIX(NO), then DB2 will still use 4 KB frames to back it. Some customers wanted the benefit of the page fix/unfix CPU savings for the buffer pools with PGFIX(YES), but did not want them to automatically use LFAREA and 1 MB frames in memory [FRAMESIZE(4KB)]. This can be done in DB2 11.

Tip: If you are in DB2 11 and are forced to limit your use of 1 MB frames, but PGFIX(YES) is acceptable, then those buffer pools with the highest GETPAGE rate should use those large frames first. Page fixing offers the most benefit to those pools with the highest I/O intensity. This case should be rare since page fixing has already reserved that much space, but maybe you are waiting for an IPL to increase the size of the LFAREA.

The use of large frames began in DB2 10 with PGFIX(YES) buffer pools. DB2 11 extends this support to allow you to have buffer pools that are PGFIX(NO) to take advantage of the TLB hit improvement. In our testing, we have not seen CPU improvement using the 2 GB frames, but as real storage footprints grow, we anticipate some savings in the future. You also cannot page fix a buffer pool with a 2 GB frame size. Since it is not page fixed, you will not benefit from the CPU savings during the fix and unfix instructions on an I/O boundary (discussed in “Large frames” on page 140). These pageable 1 MB frames are also what backs up the DB2 base code, which is discussed in 13.2.4, “New in DB2 11” on page 189. The various processing benefits are depicted in Figure 10-6.

Bufferpool - large size page frames, long term page fix

Frame size	Page fix	Supported DB2	H/W Requirement	Benefit
4 KB	NO	All	N/A	Most flexible configuration
4 KB	YES	All	N/A	CPU reduction during I/O
1 MB	NO	DB2 10 with PM85944, or DB2 11	zEC12 and Flash Express Backed by real or LFAREA	CPU reduction from TLB hit
1 MB	YES	DB2 10 above	z10 above LFAREA 1M=xx	CPU reduction during I/O, CPU reduction from TLB hit
2 GB	YES	DB2 11	zEC12 LFAREA 2G=xx	CPU reduction during I/O, CPU reduction from TLB hit

Figure 10-6 Buffer pool use of large frames

In DB2 11, we will allocate the buffer pool in its entirety at the first reference, but will not back the pages until we need them regardless of whether it is page fixed. This is a change from DB2 10, which used a “lazy load” approach. So the virtual storage footprint may look larger when DB2 comes up, but we will only back what we need with real storage. If the buffer pool is a 1 MB frame size [PGFIX(YES)] then the buffers will be backed with real storage when they are allocated.

10.2 Group buffer pools

Within the coupling facility, there are basically 3 components used by DB2:

- Lock structure: Maintains P-Lock and L-Lock status, types of locks against each object.
- SCA (list) structure: LPL, DBET status, BSDS names, LRSN delta, checkpoint data, and so on.
- Cache: Storage for cached GBP pages, as well as storage for the directory entries, which track individual pages that are part of intersystem read/write interest.

This chapter focuses on only the CACHE portion of structure, and what to monitor through statistics and the various display commands to protect the performance profile.

For a sizing exercise that involves the entire structure as well as the group buffer pools, see *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322. There are instructions and a reference for the CFSizer, which is a free use tool provided by IBM to aid with sizing the coupling facility due to an upgrade, workload increase, or other changes within the environment.

<http://www-947.ibm.com/systems/support/z/cfsizer/>

10.2.1 Best practices

The following practices are some standard suggestions regarding the entire Coupling Facility Resource Management Policy, which is not commonly maintained by the DBA group, so it will require some teaming with the system programmers. All of these parameters are options within the STRUCTURE statement in the CFRM policy. A quick way to verify these settings is to ask for the policy from your MVS counterpart, or issue the **-D XCF,STR,STRNM=groupname** command (see Example 10-7), and compare them to the “CFRM Policy.” on page 145.

*Example 10-7 -D XCF,STR,STRNM=groupname**

```
STRNAME: DB21_GBP2
STATUS: REASON SPECIFIED WITH REBUILD START:
POLICY-INITIATED
DUPLEXING REBUILD
METHOD: USER-MANAGED
PHASE: DUPLEX ESTABLISHED
EVENT MANAGEMENT: POLICY-BASED
TYPE: CACHE
POLICY INFORMATION:
POLICY SIZE      : 13200000 K
POLICY INITSIZE: 7700000 K
POLICY MINSIZE : 7700000 K
FULLTHRESHOLD   : 90
ALLOWAUTOALT   : YES
REBUILD PERCENT: 1
DUPLEX          : ENABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF00 CF00
ENFORCEORDER    : NO
EXCLUSION LIST IS EMPTY
DUPLEXING REBUILD OLD STRUCTURE
-----
ALLOCATION TIME: 10/21/2012 00:49:52
CFNAME          : CF00
COUPLING FACILITY: 002827.IBM.82.00000002E1A7
PARTITION: 01   CPCID: 00
ACTUAL SIZE      : 7520 M
STORAGE INCREMENT SIZE: 1 M
USAGE INFO      TOTAL    CHANGED    %    TOT INUSE    %
ENTRIES:        5154341    37995     0    3592151    69
ELEMENTS:       1419959    37995     2    1419063    99
PHYSICAL VERSION: C99E9D E740C01
LOGICAL  VERSION: CA9E9D E70C01
SYSTEM-MANAGED PROCESS LEVEL: 17
ACCESS TIME     : 0
MAX CONNECTIONS: 32
# CONNECTIONS   : 14
```

CFRM policy settings:

- ▶ Recommend use of XES Auto Alter (autonomic):
<http://www-01.ibm.com/support/docview.wss?uid=tss1prs1956&aid=1>
 - Exception with large NPIs: Batch or Utility jobs can flood the GBP with changed pages.
 - This necessitates either tuning the local and GBP for this scenario or possibly issuing a -ACCESS DATABASE() SPACENAM() MODE(NGBPDEP) to remove the object from GBP dependency.
<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/Webindex/Flash10669>
 - Tries to avoid Structure Full condition
 - Tries to avoid Directory Entry Reclaim condition
- ▶ CFRM Policy:
 - ALLOWAUTOALT(YES)
 - Set MINSIZE to INITSIZE
 - Set FULLTHRESHOLD = 80-90%
 - Set SIZE to 1.3-2x INITSIZE
- ▶ Periodic review and update to CFRM based on actual allocation and ratio:
Especially applies when the allocated size reaches SIZE.

- ▶ ALLOWAUTOALT allows XES to utilize algorithms to calculate appropriate size changes within a group buffer pool. It was designed for gradual increases in workload, not sudden floods of pages. These alterations are applied directly to the CF without any manual intervention, and more importantly without needing a rebuild of the structure. When either the directory entries or data pages exceed the FULLTHRESHOLD, XES will increase the component in short supply while decreasing the other one, and only when both thresholds are hit will the size of the GBP be increased. Message IXC588I, shown in Example 10-8, will accompany the altering of the structure.

Example 10-8 IXC588I

```
IXC588I AUTOMATIC ALTER PROCESSING INITIATED 989
FOR STRUCTURE GBP2
CURRENT SIZE:      250112 K
TARGET SIZE:      250112 K
TARGET ENTRY TO ELEMENT RATIO:      8881 :      31207
```

Note that if there is general storage stress on the CF (less than 10% free storage), XES can decrease those structures with ALLOWAUTOALTER(YES). XES will never decrease them below MINSIZE (defaulted to 75% of INITSIZE). The DB2 **-DISPLAY GBPPOOL** command will reflect the current directory to data ratios. The total numbers of directory entries and data elements are found in the CF usage summary section of the RMF CF activity report.

- ▶ Set MINSIZE=INITSIZE since Auto Alter can in-fact decrease the GBP sizes if necessary. We want to ensure that the GBP will not shrink, possibly degrade performance, then be enlarged again. The assumption is that your initsize is accurate based on previous experience. Generally when you go in to reevaluate a CFRM policy, you take the previous ACTUAL SIZE (translate it into 4 KB pages) and make it the new INIT size.
- ▶ Set FULLTHRESHOLD = 80-90% to ensure that Auto Alter can react before you actually run out of space in the structure. The penalties for running out of data pages and/or directory entries are discussed later.

- Set SIZE to 1.3-2x INITSIZE to allow Auto Alter to work its magic within a reasonable confinement of size. Obviously you need to ensure that the CF itself has enough storage available to accommodate all the Auto Alter structures reaching their max size, so this may be part of the capacity planning perspective. Do not assume that the max size will not be reached.
- Periodically review and update the CFRM, and take action when ACTUAL SIZE= SIZE. When the structure has reached size, Auto Alter is out of options regarding what conditions it can avoid. In order to accommodate fluctuations in storage for changed pages and directory entries Auto Alter will have to swap storage back and forth between the list and cache structures. This lack of equilibrium is not an efficient use of resources.

Some statistics tuning

When DB2 does a getpage, DB2 first looks for the page in the local buffer pool, then the group buffer pool, and finally does an I/O to DASD. In data sharing, if the page in the GBP has been updated by one DB2 member, and the page exists in other DB2's local buffer pools, then the corresponding pages in those buffer pools will be cross invalidated so that the stale data is not read. When a getpage discovers that its local buffer was invalidated by another member, it next reads the page from the GBP if it is there. If it is not found in the GBP, it will do a synchronous read I/O from DASD.

Example 10-9 shows statistics details for synchronous read. SYN.READ(XI) -DATA RETURNED means the data page was still in the GBP. However, -NO DATA RETURN means that the changed page is no longer in the GBP, so the requesting DB2 must do an I/O. With good page residency time in the GBP, we would like to see that 90% of the time the invalidated page is still in the GBP.

Example 10-9 GBP statistics for synchronous read

GROUP BP14	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
...				
SYN.READ(XI)-DATA RETURNED	1932.00	0.09	0.01	0.00
SYN.READ(XI)-NO DATA RETURN	39281.6K	1823.66	236.31	79.22
SYN.READ(NF)-DATA RETURNED	22837.00	1.06	0.14	0.05
SYN.READ(NF)-NO DATA RETURN	6955.8K	322.93	41.85	14.03

Tip: The sync read cross invalidation (XI) ratio should be < 10%.

Sync.Read(XI) miss ratio = SYN.READ(XI)-NO DATA RETURN / TOTAL SYN.READ(XI)

TOTAL SYN.READ(XI) = SYN.READ(XI)-DATA RETURNED + SYN.READ(XI)-NO DATA RETURN

Starting in DB2 10, when a page set goes out of group buffer pool dependency instead of scanning the local buffers of the members to ensure that the pages are consistent during the transition, we will cross invalidate the pages. This enhancement was used to reduce application timeouts due to P-LOCK negotiations as well as the DBM1CPU time consumed while the scan took place. If your objects are going in and out of GBP dependency and the sync I/Os in the application are increasing dramatically, then look at the number of pseudo closes occurring in relation to when applications using object scans occur, and possibly increase the PCLOSEN and PCLOSET values. See 6.2.1, "Monitoring open and close activity" on page 80 for how to monitor pseudo close activity.

If the ratio listed is much over 10% then the GBP could be too small or there could be cross invalidation due to directory reclaims going on Example 10-9. In that example, there were many cross-invalidations so the ratio is skewed drastically toward the page not being found in the GBP.

10.2.2 What to look out for

There are several statistics to be monitored as well as scenarios that should be avoided.

Recoverability

In DB2 9, the ability to alter a base table to NOT LOGGED was introduced. Customers have used this option to reduce logging volumes as well as Utility job CPU and elapsed times.

There is an inherent danger in using this with LOB objects though. If that NOT LOGGED LOB object becomes GBP dependent and an event (loss of connection to CF) occurs to place the object on the GRECP/LPL list, the object becomes unrecoverable. Automatic GRECP in DB2 9 and up will not help as the object will be put in AUXW or refresh pending. Hence you cannot recover the object using image copies as there are no log records to know if the object has been modified since the last copy.

Structure sizes

The **DISPLAY GROUPBUFFERPOOL** command (see Example 10-10) gives you a high level view to look for initial issues. Cross invalidations DUE TO DIRECTORY RECLAIMS. These cross invalidations are due to a shortage of directory entries, and hence one that was currently in use was stolen in order to track a new page that has come into one of the local buffer pools.

So the old page that was referenced by that directory entry is cross invalidated, and when it is next referenced it will require an I/O to bring that page back in. If you take the number of XI due to directory reclaims over the course of a day, and multiply it by the local buffer pool hit ratio you can get an estimate of how many extra sync I/Os were caused in that buffer pool by the lack of directory entries.

Tip: To avoid directory reclaims, enlarge the GBP, or increase the ratio of directory entries to data pages and rebuild the structure.

Example 10-10 -DIS GBPOOL() GDETAIL(*) TYPE(GCONN)*

DSNB787I	-DB1A	RECLAIMS	
		FOR DIRECTORY ENTRIES	= 46590253
		FOR DATA ENTRIES	= 76336383
		CASTOUTS	= 237620894
DSNB788I	-DB1A	CROSS INVALIDATIONS	
		DUE TO DIRECTORY RECLAIMS	= 3074104
		DUE TO WRITES	= 216105431
		EXPLICIT	= 0
DSNB762I	-PB1A	DUPLEXING STATISTICS FOR GBP16-SEC	
		WRITES	
		CHANGED PAGES	= 474956841
		FAILED DUE TO LACK OF STORAGE	= 57
		CHANGED PAGES SNAPSHOT VALUE	= 13421

Example 10-11 shows the corresponding failed writes in the statistics report.

Example 10-11 Failed writes from statistics report

GROUP BP14	QUANTITY	/SECOND	/THREAD	/COMMIT
CASTOUT ENGINE NOT AVAIL.	N/A	N/A	N/A	N/A
WRITE ENGINE NOT AVAILABLE	N/A	N/A	N/A	N/A
READ FAILED-NO STORAGE	N/A	N/A	N/A	N/A
WRITE FAILED-NO STORAGE	57.00	0.00	0.00	0.00

FAILED DUE TO LACK OF STORAGE means that the portion of the GBP designed to hold changed pages is full, and there is no page left to steal. The DB2 member who is updating the page will continue to attempt to write the page to the GBP for up to 8 minutes, meanwhile “emergency castout” is scheduled in the GBP. If GBP castout is not enough to free up space for the page, then the page will be put on the LPL (logical page list) and that page will have to be recovered. The solution to this problem once again harkens back to sizing.

Tip: The rule of thumb is to have the following result:

FAILED DUE TO LACK OF STORAGE <1% of total changed pages written [CHANGED PGS SYNC.WRTN + CHANGED PGS ASYNC.WRTN]

Ideally, this should be ‘0’ as you are burning CPU cycles trying to write the page multiple times (up to 8 minutes). The application must also wait for this process to complete if it is during COMMIT processing, and each iteration is 1 second long. Enlarging the GBP, or smaller castout thresholds, or more frequent GBP checkpoints could help alleviate this condition. Another tuning option is to manually reduce the RATIO in favor of data pages and rebuilding the CF using SETXCF. Assuming a directory entry size of 208 bytes, it may take a considerable decrease to accommodate the number of changed pages coming in.

The failures due to lack of storage is also accompanied by messages in the DB2 MSTR joblog. See Example 10-12. The DSNB319A message talks about a SHORTAGE of space as the changed pages are consuming 75% of the available space for data pages in the coupling facility. Then when the DSNB325A, CRITICAL SHORTAGE message appears, it means that 90% of the available pages have been consumed. Ideally we never want to see CRITICAL SHORTAGE messages.

Example 10-12 MSTR JOBLOG message for short on storage

*DSNB319A	-DB2A DSNB1CNE THERE IS A SHORTAGE OF SPACE	829
	829 IN GROUP BUFFER POOL GBP20	
*DSNB325A	-DB2A DSNB1CNE THERE IS A CRITICAL SHORTAGE	542
	542 OF SPACE IN GROUP BUFFER POOL GBP20	

10.2.3 Rough sizings

The CFSizer tool is a very good place to start from based on microcode or hardware upgrades, but every once in a while you need a sanity check. Also the CFSizer is not based on the workload occurring in the system.

The amount of real data sharing read/write interest, and the ratio of pages read versus pages written, and even the CLOSE YES/NO option for the data set can change the necessary page to directory entry ratio. With sudden workload spikes, as seen in some batch jobs, the Auto Alter code may cause violent swings in the directory to data ratio or the number of data pages in the GBP; this has to swing the other way to accommodate the peak online day workload.

The max ratio of directory entries to pages was recently increased from 40:1 up to 255:1. This allows for much larger local buffer pools among data sharing members who do not have much read/write interest in the pages.

The number of changed pages per second can be pulled from the display GBP output. See Example 10-13.

Example 10-13 -DIS GROUPBUFFERPOOL() GDETAIL(INTERVAL)*

```

DSNB782I  DB1S INCREMENTAL GROUP DETAIL STATISTICS SINCE 16:11:11 SEP
          9, 2013
DSNB784I  DB1S GROUP DETAIL STATISTICS
          READS
              DATA RETURNED                      = 0
DSNB785I  DB1S      DATA NOT RETURNED
              DIRECTORY ENTRY EXISTED              = 8
              DIRECTORY ENTRY CREATED              = 11
              DIRECTORY ENTRY NOT CREATED          = 1, 0
DSNB786I  DB1S  WRITES
              CHANGED PAGES                      = 2390
              CLEAN PAGES                        = 0
              FAILED DUE TO LACK OF STORAGE        = 0
              CHANGED PAGES SNAPSHOT VALUE        = 7
DSNB787I  DB1S  RECLAIMS
              FOR DIRECTORY ENTRIES                = 0
              FOR DATA ENTRIES                   = 0
              CASTOUTS                             = 1
DSNB788I  DB1S  CROSS INVALIDATIONS
              DUE TO DIRECTORY RECLAIMS          = 0
              DUE TO WRITES                       = 6
              EXPLICIT                             = 0
DSNB790I  DB1S DISPLAY FOR GROUP BUFFER POOL GBPO IS COMPLETE

```

Alternatively, changed pages per second from a statistics report are shown in Example 10-14.

Example 10-14 Statistics report to get changed pages per second

GROUP BP3	CONTINUED	QUANTITY	/SECOND	/THREAD	/COMMIT
WRITE AND REGISTER		6101.2K	77.04	75.00	0.34
WRITE AND REGISTER MULT		746.7K	9.43	9.18	0.04
CHANGED PGS SYNC.WRTN		7417.3K	93.65	91.18	0.41
CHANGED PGS ASYNC.WRTN		384.1K	4.85		

Building a small spreadsheet for each GBP can really help with the accuracy and planning of the total CF size as well as trying to eliminate cross-invalidations due to directory reclaims, SYN.READ(XI)-NO DATA RETURN, and even write failures due to storage and LPL conditions. Gathering the statistics for the changed pages during peak and also during batch could help determine how to avoid violent swings in the ratios as well. If your site has different buffer pool settings during batch and peak, then this certainly needs to be taken into account.

Figure 10-7 shows a sample GBP spreadsheet.

	A	B
2	Data Page Size (KB)	4
3	Directory Size (Bytes) - CF Level Dependant	432
4	# Members	4
5		Rough Calculation
6		BP2
7	VP Size	1000
8	Change Page Residency Time (secs)	30
9	Peak Changed Pages Written / s	99
10	Changed Page Rate + 20%	118.8
11	# Changed Pages (assuming residency Time)	3564
12	Data Entries (KB)	14256
13	# Directory Entries	7564
14	Directory Entries (KB)	3192
15	Structure Init Size(MB)	18
16	Structure Size (Init Size + 30%)(MB)	23
17	Ratio	2

Figure 10-7 Example of rough sizing of GBP

Here are the list of fields in the spreadsheet and their meanings using the actual values from this example; the values themselves are not important:

- ▶ B2: Data page size is the size of the page in kilobytes, 4, 8, 16, or 32:
 - 4 in this example
- ▶ B3: A directory entry is roughly 432 bytes for a 4 KB page BP and 530 bytes for 32 KB pages, assuming that the coupling facility level is 18 (CFLEVEL=18):
 - 432 in this example for a 4 KB buffer pool
- ▶ B4: # of members represents the number of members in the data sharing group:
 - 4 members in this example
- ▶ B7: VPSIZE is the size of the corresponding local buffer pool (assuming each is the same size, otherwise adjust accordingly):
 - 1000 buffers to keep it simple
- ▶ B8: Changed page residency time goal should be 30-180 seconds ideally (not calculated, just a best practice) and affects the calculations later so you can adjust it to see what kind of residency time you can aim for with the given storage constraints:
 - 30 seconds is on the low end, but saves CF storage if we are short-handed
- ▶ B9: Peak changed pages written per second can be attained from the **DISPLAY** command or the statistics interval:
 - 99 achieved by adding **CHANGED PGS SYNC.WRTN** + **CHANGED PGS ASYNC.WRTN** per second

- ▶ B10: This is a 20% padding of the changed pages in order to account for unforeseen peaks:
 - $=B9*1.2$
- ▶ B11: Multiply the number of changed pages per second by the time you want them resident in the GBP:
 - $=B10*B8$
- ▶ B12: Storage for Data Entries in rough Kilobytes:
 - $=ROUNDUP(B11*B2,0)$
- ▶ B13: # of directory entries, simply total number of possible unique pages open at once, so it includes the number of buffers in the local pools as well as the GBP:
 - $=(B7*B4)+B11$
- ▶ B14: directory entries storage in Kilobytes:
 - $=ROUNDUP((B13*B3)/1024,0)$
- ▶ B15: structure init size in Megabytes:
 - $=ROUNDUP((B14+B12)/1024,0)$
- ▶ B16: structure size plus 30% cushion in Megabytes:
 - $=ROUND(B15*1.3,0)$
- ▶ B17: ratio of directory entries to pages:
 - $=ROUND(B13/B11,0)$

10.2.4 Group buffer pools and DB2 11

For several years, customers have been asking for a way to ensure that large batch processes or Utility operations do not flood the group buffer pools with changed pages. This will cause the highly referenced, useful pages, to be castout and fill the GBP with pages that only that batch job was referencing. It will also force you to implement an artificially low CLASST threshold to ensure that the castout process can keep up with the updates.

The solution to this issue is what is called GBP write-around. With regards to asynchronous writes, if the page is already cached in the GBP, it will be written there, otherwise the page will be written “around” the CF directly to DASD. After it is written to DASD, the XI signals will be sent. The pages written around the CF will be those written out due to hitting VDWQT/DWQT in the local buffer pools.

Two thresholds are used to determine whether GBP write-around is invoked for all objects in a GBP or for a page set/partition: the GBP castout threshold; and the class castout threshold. When the GBP castout threshold hits 50%, meaning that 50% of the GBP is occupied by changed pages, then write-around is invoked for writing pages for all objects. When the class castout threshold hits 20%, meaning that 20% of a class castout queue is occupied by changed pages, then DB2 will employ the write-around protocol for the page set/partition. The write-around process at the GBP level will continue until the GBP castout threshold drops to 40%. The write-around process at the page set/partition level will continue until the class castout threshold drops to 10%. The thresholds at which the write-around stops cannot be changed.

As an example, let us say that we use the default CLASST=5%. When the number of changed pages in a class queue reaches 5%, DB2 will begin to write the pages to DASD. This is the castout process. However, at the same time, applications could be writing more changed pages to the GBP. If the transactions write pages into the GBP faster than DB2 can cast out the pages to DASD, the percentage of changed pages could well exceed 5%. With DB2 11, when that percentage reaches 20%, DB2 will stop writing more changed pages to the GBP and will begin to write pages directly from the local buffer pool to DASD.

Group buffer pool threshold CLASST in DB2 11 is now allowed to be configured much like VDWQT for local buffer pools. You can specify 0,[0-32767] to ensure that the threshold is hit at less than 1% of the group buffer pool size, which could be quite large. This could be especially useful for buffer pools with NPIs, or those that experience large write bursts from utility or batch jobs.



DDF activity

DB2 for z/OS distributed data facility (DDF) is a built-in component of DB2 that provides the connectivity to and from other databases over the network. DDF implements a full DRDA application server and application requester protocol.

This chapter contains the following topics:

- ▶ DDF and distributed environments
- ▶ Monitoring DDF activity

11.1 DDF and distributed environments

DDF is DB2's transaction manager for distributed database connections. DDF has developed mature thread management strategies to handle thousands of connections that can come from anywhere within the bounds of the network that DB2 is operating in. Figure 11-1 illustrates various DRDA applications coming to DB2 via DDF.

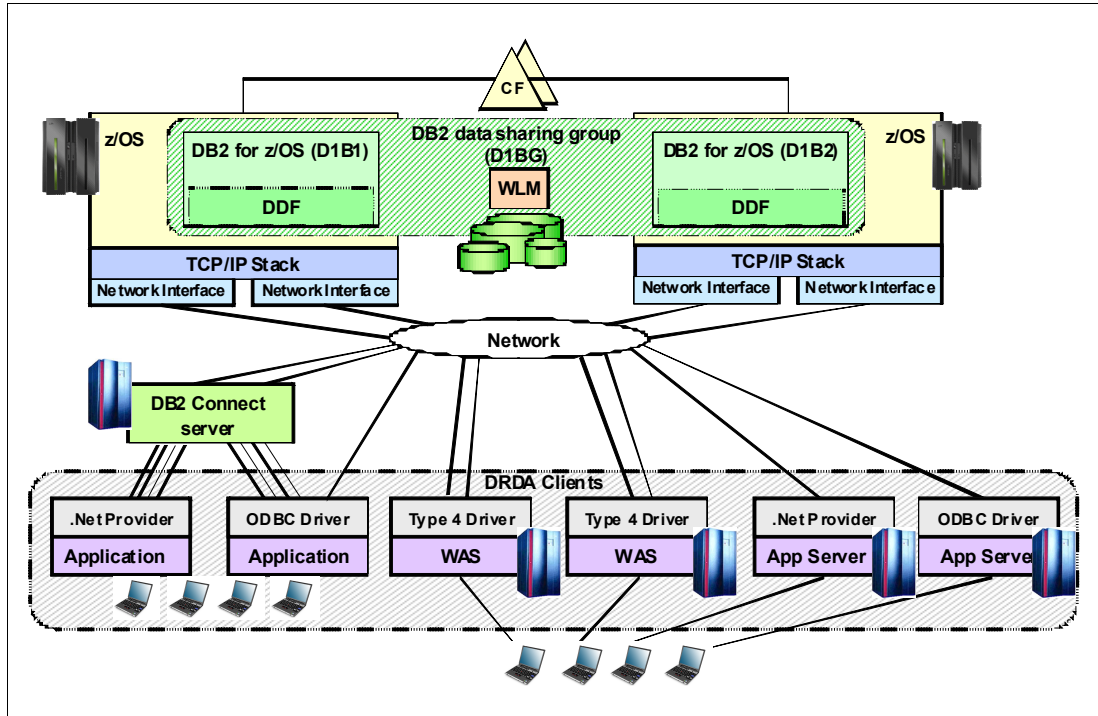


Figure 11-1 DB2 and distributed applications

DDF runs as an additional address space in the DB2 subsystem. The address space name is ssidDIST, where ssid is the DB2 subsystem name. DDF is an efficient connection handler. It uses SRBs instead of TCBs, which reduces CPU time. z/OS enclaves are used in exchanging data across address spaces. This enables proper management by Workload Manager (WLM) of the work coming into DB2 through DDF and the possibility of routing DDF work to the zIIP specialty engine.

DDF is the key component for handling connections from requesters, managing DBATs, and DRDA transactions. Using statistics traces, you can monitor how those resources are used or managed against DB2 resources set via subsystem parameters. You need to increase or limit those resources depending on how DDF working. Another important factor is that a lot of DRDA transactions such as JDBC, ODBC, .NET providers request SQL as a dynamic SQL, which can consume more storage. With V10 or later, a virtual storage may not be a constraint anymore, but you still need to monitor storage consumption to avoid having paging with important resources, which can impact your DB2 and/or overall system performance.

Figure 11-2 shows an overview of DDF related resources and processing. Because connections are made based on client requests, settings for application servers may be required to be changed in order to manage your resources efficiently.

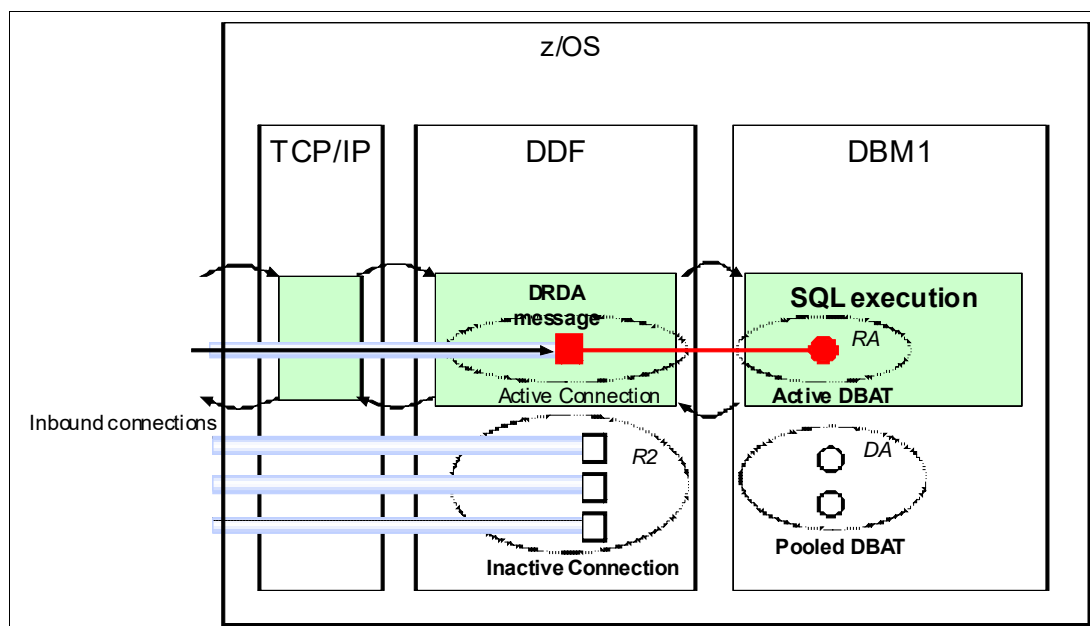


Figure 11-2 Processing within DDF

In the following sections, we describe some of the key resources and functionalities.

11.1.1 Database access thread

The transaction, or unit-of-work, from a remote system is received by the DDF, a thread is created in the DBM1 address space. This DB2 thread created for DRDA transactions are called a Database Access Threads or DBATs. Otherwise, and most likely, if a thread for a remote connection already exists, but is idle (pooled DBAT), it is reused for the request. The DBAT is implemented through an z/OS enclave, performing work through preemptive SRBs running in the DDF.

Because DBATs impact on performance and accounting, the general recommendation is to use INACTIVE MODE threads instead of ACTIVE MODE threads. The number of concurrent transaction on DB2 means number of DBATs needed to efficiently handle those transactions.

Table 11-1 lists the most relevant DSNZPARM parameters affecting DBATs.

Table 11-1 Summary of DSNZPARM parameters affecting DBATs

Parameter	Description
CMTSTAT	ACTIVE or INACTIVE. It governs whether DBATs/connections remain active across commits.
MAXDBAT	Maximum number of concurrent DBATs (<=19999) or connections if CMTSTAT=ACTIVE
CONDBAT	Maximum number of concurrent connections (<=150000)
POOLINAC	The approximate time, in seconds that a DBAT can remain idle in the pool before it is terminated
IDTHTOIN	Idle thread timeout interval
MAXCONQN	The maximum number of inactive or new connections that can be queued waiting for a DBAT to process the request.
MAXCONQW	The maximum length of time that a client connection waits for a DBAT to process the next unit-of-work or new connection request.

11.1.2 Modes of DBATs and inactive connections

DB2 has two modes of a DBAT after a successful commit or rollback operation, when certain conditions are met, such as if the DBAT holds no cursors. Modes of DBATs are controlled by CMTSTAT DSNZPARM parameter.

If you specify ACTIVE, the DBAT remains active and continues to consume system resources for life of a connection. This restricts the number of connections you can support for client applications. If you must support a large number of connections, specify INACTIVE, which is the default.

If you specify INACTIVE, if the DBAT holds no cursors, has no declared global temporary tables defined, no LOB locators being held, and has not accessed KEEP DYNAMIC YES packages, then DB2 will disassociate the DBAT from the connection, mark the connection inactive, and return the DBAT to a pool for use by another connection. This concept is also called inactive connection. Because the DBAT is pooled to be reused by any connections, it is more efficient in support a large number of connections.

While having inactive connection support, having a new unit-of-work means it will be dispatched to one of pooled DBATs, and if there were no pooled DBATs to dispatch new DBAT will be created. If MAXDBAT is reached while requesting new DBAT, those requests will be queued in DDF. After DBAT put into pool for 200 times, DBAT will be purged.

From the monitoring point of view, you should know what mode of DBATs you are using your DB2 with and how DBAT and connection being handled. We show you how to check in a running DB2 in 11.2, “Monitoring DDF activity” on page 161.

11.1.3 High Performance DBATs

Back in DB2 V6, all packages that accessed at server through DRDA were forced to work as RELEASE(COMMIT) even if they were bound as RELEASE(DEALLOCATE).

DB2 10 re-introduced the function so you can honor package options, where the package bound as RELEASE(DEALLOCATE) to run against DBATs, this is called the high performance DBAT. The function helps reduces CPU cost for package allocation and deallocation processing. Performance results can vary but the benefits are more pronounced for short transactions.

All the interaction with the clients will be the same, so if the client is part of a sysplex workload balancing, it will still receive indications that the connection can be multiplexed among many client connections.

The high performance DBAT re-introduced a restriction of DDL or BIND not able to break in while RELEASE(DEALLOCATE) packages are being active.

DB2 11 introduces an option to handle this, by setting a new system parameter PKGREL_COMMIT to YES, a persistent DB2 thread, at COMMIT or ROLLBACK, to implicitly release a package which is bound with RELEASE(DEALLOCATE) and active on that thread if there is a BIND REPLACE/REBIND PACKAGE, online schema change operation (such as DDL), or Online REORG with deferred ALTER operation that needs to quiesce/invalidate the package.

With the high performance DBAT, enclave will become inactive after successful commits or rollbacks, and accounting records will be cut. After units-of-work, the high performance DBAT will stay active associated with a connection, waiting for a new transaction to come in using the same connection.

When a high performance DBAT hits a timer for POOLINAC without getting the next transaction, it will be terminated and the connection becomes inactive. After the high performance DBAT is terminated by POOLINAC, DB2 creates pooled DBAT and they can be reused by any other inactive connections. The pooled DBATs will be terminated when they reach POOLINAC again without being assigned to a connection.

The high performance DBAT should be used selectively by separating packages for those applications who use them. You can do so by creating sets of packages with different RELEASE bind option with different collection identifiers and having those collection identifiers specified at the data source level using the following method depending on application environment. Be aware that the clients matching DB2 10, DB2 Connect™ V9.7 FP3a, and later, now use RELEASE(DEALLOCATE) as a default option:

- ▶ CurrentPackageSet property or for JDBC application
- ▶ CurrentPackageSet parameter in the db2dsdriver.cfg configuration file for .NET and ODBC / CLI applications.

To enable the function, in addition to having your packages bound with RELEASE(DEALLOCATE) and set it to be used from application, issue the **-MODIFY DDF** command with **PKGREL** option set to **BNDOPT**, as shown in Example 11-1. By changing this, DB2 will honor the package RELEASE option.

Example 11-1 Enable High Performance DBAT

```
-DB1A MODIFY DDF PKGREL(BNDOPT)
DSNL300I -DB1A DSNLTMDf MODIFY DDF REPORT FOLLOWS: 271
DSNL302I PKGREL IS SET TO BNDOPT
DSNL301I DSNLTMDf MODIFY DDF REPORT COMPLETE
```

To disable the function, issue the **-MODIFY DDF** command with **PKGREL** option set to **COMMIT**, as shown in Example 11-2. This will overlay the BIND option and forces it to work as RELEASE(COMMIT), which is the same behavior as DB2 9 and before. You might want to choose to disable the function when you want to allow BIND or DDL to run concurrently with your distributed transactions in DB2 10.

Example 11-2 Disable High Performance DBAT

```
-DB1A MODIFY DDF PKGREL(COMMIT)
DSNL300I -DB1A DSNLTMDf MODIFY DDF REPORT FOLLOWS: 273
DSNL302I PKGREL IS SET TO COMMIT
DSNL301I DSNLTMDf MODIFY DDF REPORT COMPLETE
```

As an existing DBAT comes to a commit point, after the PKGREL(COMMIT) has been set, regardless of the number of commits so far, the thread is terminated and the next request from the connection that had been using it is a new DBAT which continues RELEASE(COMMIT) processing. Any thread that has been sitting idle, waiting for a new unit-of-work to be started, is told to terminate by the DDF error monitor task which cycles every 2 minutes, such that after at most two DDF error monitor executions, all DBATs will be running as RELEASE(COMMIT) threads.

For DB2 11, the difference is that once in RELEASE(COMMIT) mode, it will take effect on the next COMMIT if there is a waiter and not just for new DBATs.

DB2 checks to see the PKGREL option when about to make the DBAT wait for its next transaction while in high-perf mode. If this thread is a candidate for staying active due to RELEASE(DEALLOCATE) packages, DB2 terminates the DBAT at the next commit.

However, if there are poorly behaved applications (held cursors past commit, DGTTs still allocated, and so on, DB2 leaves them active, which may include RELEASE(DEALLOCATE) resources still be allocated. However, with DB2 11 break in, these errant threads can be made to be RELEASE(COMMIT) at commit boundaries for break-in purposes.

The OMEGAMON PE statistics trace gives you fields for use of high performance DBATs. Because the high performance DBATs hold of the DBATs is much longer than regular DBATs, you are expected to use up your DBATs much faster than regular DBATs. If you are changing POOLINAC values for some reason, such as gaining reusability of regular DBATs, you might want to carefully monitor high performance DBATs also, for POOLINAC also affects against idle high performance DBATs as previously described. See Example 11-3.

Tip: When using high performance DBATs, you end up having many more active DBATs created. It can significantly increase your storage based on the range of your applications or application servers you apply high performance DBATs against. Monitor the number of DBATs and the storage resources usage when applying high performance DBATs.

Example 11-3 High performance DBAT activity

GLOBAL DDF ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
DBAT/CONN QUEUED-MAX ACTIVE	0	0.00	N/C	N/A
CONN REJECTED-MAX CONNECTED	0	0.00	N/C	N/A
CONN CLOSED - MAX QUEUED	0	0.00	N/C	N/A
CONN CLOSED - MAX WAIT	0	0.00	N/C	N/A
COLD START CONNECTIONS	0	0.00	N/C	0.00
WARM START CONNECTIONS	0	0.00	N/C	0.00
RESYNCHRONIZATION ATTEMPTED	0	0.00	N/C	0.00
RESYNCHRONIZATION SUCCEEDED	0	0.00	N/C	0.00
CUR TYPE 1 INACTIVE DBATS	0	N/A	N/A	N/A
HWM TYPE 1 INACTIVE DBATS	1	N/A	N/A	N/A
TYPE 1 CONNECTIONS TERMINAT	0	0.00	N/A	N/A
CUR INACTIVE CONNS (TYPE 2)	1	N/A	N/A	N/A
HWM INACTIVE CONNS (TYPE 2)	31	N/A	N/A	N/A
ACC QU INACT CONNS (TYPE 2)	30	0.50	N/A	N/A
CUR QU INACT CONNS (TYPE 2)	0	N/A	N/A	N/A
MIN QUEUE TIME	0.000073	N/A	N/A	N/A
MAX QUEUE TIME	0.000228	N/A	N/A	N/A
AVG QUEUE TIME	0.000110	N/A	N/A	N/A
HWM QU INACT CONNS (TYPE 2)	3	N/A	N/A	N/A
CUR ACTIVE AND DISCON DBATS	30	N/A	N/A	N/A
HWM ACTIVE AND DISCON DBATS	30	N/A	N/A	N/A
HWM TOTL REMOTE CONNECTIONS	31	N/A	N/A	N/A
CUR DISCON DBATS NOT IN USE	0	N/A	N/A	N/A
HWM DISCON DBATS NOT IN USE	3	N/A	N/A	N/A
DBATS CREATED	0	N/A	N/A	N/A
DISCON (POOL) DBATS REUSED	30	N/A	N/A	N/A
CUR ACTIVE DBATS-BND DEALLC	4	N/A	N/A	N/A
HWM ACTIVE DBATS-BND DEALLC	16	N/A	N/A	N/A

11.1.4 Sysplex workload balancing and connection queue redirect

For the transactions connecting to a DB2 data sharing group, the connections are established exactly the same way as connections to a non-data sharing DB2 subsystem. The information for the chosen DB2 data sharing member must be catalogued to the appropriate configuration. After a successful initial connection, the following transactions will automatically make connections to either members based on weight, which is information returned by z/OS WLM.

The DRDA workload balancing (WLB) work seamlessly interacting with DB2, z/OS WLM, and clients. In case you have an issue, the information relating to WLB can be collected from server and clients, such as RMF, DB2 traces, DB2 command output, client trace, and so on.

For more information, see *DB2 for z/OS and WebSphere Integration for Enterprise Java Applications*, SG24-8074 and *DB2 9 for z/OS: Distributed Functions*, SG24-6952.

Connection queue redirect

When MAXDBAT is reached at a member of a data sharing group, requests will wait in a queue until either CONDBAT has been reached or the number of requests waiting for a DBAT exceeds the MAXDBAT value. They can be either requests via inactive connection or a requests establishing a new connection. When a DBAT is available, DDF will then associate the DBAT to the next queued connection request so the work can be processed. When a DBAT does not becomes available, the queue of connection requests will continue to grow until either CONDBAT has been reached or the depth of the queue has exceeded the MAXDBAT value:

- ▶ When CONDBAT is reached, new connection requests will be rejected and message DSNL030I is issued with reason code 00D31034, as shown in Example 11-4.
- ▶ If the depth of queue has been exceeded before CONDBAT is reached, then DB2 closes the TCP/IP socket and marks the connection for eventual clean up, but a DBAT is still needed to fully complete this connection clean up.

Example 11-4 DSNL030I

```
DSNL030I  -D1B1 DSNLQCTP DDF PROCESSING FAILURE FOR
LUWID=G944FF9C.C9DF.000000000000
REASON=00D31053
THREAD-INFO=*.~*.~*.~*.~*.~*
```

In the end, DDF ends up reaching a point where no new work can be accepted. And because the client does not get instantly notified with the DB2 unavailability waiting for DBATs, this condition can continue on the member until the DBAT unavailability state is resolved.

To handle this, when requests come from applications using the IBM Data Server Driver and/or the DB2 Connect product family with a sysplex workload balancing configured, those requests, if given the chance, could have been seamlessly redirected to another DB2 member in a group where execution resources may be more abundant. This behavior is controlled by MAXCONQN and MAXCONQW subsystem parameter. Where MAXCONQN set to ON, it means that the depth of the connection queue is limited by the value of the MAXDBAT subsystem parameter. And MAXCONQW set to ON means that connections wait as long as the value specified by the IDHTOIN subsystem parameter.

In addition, to balance server resources, consider the number of connections reaching maximum allowed. When the current number of client connections exceeds either 80% or 90% of CONDBAT, DSNL074I will be issued. When this happens, the DDF system health will also be reduced to 50% or 25%, respectively, which will be reported to WLM immediately and reflected to weight information return to clients. A message is shown in Example 11-5.

Example 11-5 DSNL074I

```

DSNL074I  -D1B1 DSNLILNR NUMBER OF CLIENT CONNECTIONS
HAS EXCEEDED 80% OF THE MAXIMUM ALLOWED
DSNL074I  -D1B1 DSNLILNR NUMBER OF CLIENT CONNECTIONS
HAS EXCEEDED 90% OF THE MAXIMUM ALLOWED

```

When the number of client connections no longer exceeds 80% or 90% of CONDBAT, DSNL075I is issued. When this happens the DDF system health is going to be set back to either 50% or 100%, respectively. The message is shown in Example 11-6.

Example 11-6 DSNL075I

```

DSNL075I  -D1B1 DSNLEDDA NUMBER OF CLIENT CONNECTIONS
NO LONGER EXCEEDS 80% OF THE MAXIMUM ALLOWED

```

The **-DISPLAY DDF DETAIL** command reports the current configured values for the MAXCONQN and MAXCONQW subsystem parameters with MCONQN and MCONQW of DSNL091I. The number of closed connections due to MAXCONQN and MAXCONQW with CLSDCONQN and CLSDCONQW of DSNL094I, shown in Example 11-7. These messages will only be displayed when the subsystem is a member of a data sharing group. Also, regardless of the values specified for MAXCONQN and MAXCONQW, a value of zero (OFF) will be assumed if DB2 is configured with CMTSTAT=ACTIVE.

Example 11-7 -DISPLAY DDF DET displaying connection queue

```

-D1B1 DIS DDF DET
DSNL080I  -D1B1 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I  STATUS=STARTD
DSNL082I  LOCATION              LUNAME              GENERICCLU
DSNL083I  DB1B                  USIBMSC.SCPD1B1  -NONE
DSNL084I  TCPPORT=39400  SECPORT=39401  RESPORT=39402  IPNAME=-NONE
DSNL085I  IPADDR=::9.12.6.70
DSNL086I  SQL      DOMAIN=wtsc63.itso.ibm.com
DSNL086I  RESYNC  DOMAIN=wtsc63.itso.ibm.com
DSNL089I  MEMBER  IPADDR=::9.12.6.70
DSNL090I  DT=I    CONDBAT= 10000  MDBAT= 200
DSNL091I  MCONQN= 200 MCONQW= 120
DSNL092I  ADBAT= 0  QUEDBAT= 0  INADBAT= 0  CONQUED= 0
DSNL093I  DSCDBAT= 0  INACONN= 0
DSNL094I  WLMHEALTH=100 CLSDCONQN= 0 CLSDCONQW= 0
DSNL100I  LOCATION SERVER LIST:
DSNL101I  WT IPADDR              IPADDR
DSNL102I  17 ::9.12.6.9
DSNL102I  5  ::9.12.6.70
DSNL105I  CURRENT DDF OPTIONS ARE:
DSNL106I  PKGREL = COMMIT
DSNL099I  DSNLTDDF DISPLAY DDF REPORT COMPLETE

```

11.2 Monitoring DDF activity

In this section, we describe key fields for monitoring DDF using command and statistics traces. There are several traces related to monitoring DDF activity. We concentrate on monitoring how DDF is working:

- ▶ DDF activity report
- ▶ Statistics Global DDF Activity
- ▶ Statistics DRDA remote locations
- ▶ Monitoring using system profiling
- ▶ Client information and workload management

11.2.1 DDF activity report

The **-DISPLAY DDF DETAIL** command gives you active settings of the CMTSTAT, CONDBAT, and MAXDBAT subsystem parameters for running DB2 via DSNL090I, which CONDBAT and MAXDBAT can be updated online:

- ▶ DT gives the DDF thread values, which shows you settings for CMTSTAT values, where I means INACTIVE, and A means ACTIVE.
- ▶ CONDBAT gives the maximum number of inbound connections, which is also set with the same CONDBAT subsystem parameter.
- ▶ MDBAT gives the maximum number of DBATs, which is set via the MAXDBAT subsystem parameter.

The command output also shows various other DDF activity information that relates to DBATs and inactive connections when the command was issued:

- ▶ ADBAT shows current number of DBATs which includes pooled DBATs.
- ▶ DSCDBAT shows current number of pooled DBATs.
- ▶ INACONN shows current number of inactive connections, when DT=I.

When the following counters show a non-zero value, you may want to investigate the cause of it and how it affected the performance of the transactions waiting DBATs to be serviced:

- ▶ QUEDBAT shows a cumulative counter that is incremented when the MAXDBAT limit is reached. The QUEDBAT counter is reset only when this DB2 subsystem is restarted.
- ▶ CONQUED shows current number of queued connection requests that are waiting to be serviced by a DBAT.

Tip: When CMTSTAT=INACTIVE, a large number of connections can be serviced by smaller numbers of DBATs; having QUEDBAT and/or CONQUED with a non-zero value may indicate having MAXDBAT set too small. The QUEDBAT counter is cumulative, so look at the trace to investigate, when it happened, how it affected the performance of transactions. Setting a larger MAXDBAT will affect the size of virtual and real storage, and you should also investigate the impact of increasing the size before making any changes.

Example 11-8 gives a sample **-DISPLAY DDF DETAIL** command output from a non-data-sharing DB2 subsystem.

Example 11-8 -DISPLAY DDF DETAIL

```
-DB1A DIS DDF DET
DSNL080I -DB1A DSNLTDDF DISPLAY DDF REPORT FOLLOWS: 140
DSNL081I STATUS=STARTD
DSNL082I LOCATION          LUNAME          GENERICLU
DSNL083I DB1A              USIBMSC.SCPDB1A  -NONE
DSNL084I TCPRT=38420  SECRT=38422  RESRT=38421  IPNAME=-NONE
DSNL085I IPADDR=:9.12.6.70
DSNL086I SQL      DOMAIN=wtsc63.itso.ibm.com
DSNL090I DT=I  CONDBAT= 10000  MDBAT= 200
DSNL092I ADBAT= 30  QUEDBAT= 0  INADBAT= 0  CONQUED= 0
DSNL093I DSCDBAT= 0  INACONN= 1
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = BNDOPT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

11.2.2 Statistics Global DDF Activity

OMEGAMON PE statistics report will show you how DDF is working managing DBATs and inactive connections.

The following two fields show efficiency of DBAT reuse for DRDA transactions. Creating a new DBAT can cost CPU. Having a small number of DBATs created to process as many transactions as possible helps gain CPU efficiency, especially during a peak workload period where it consumes a lot of CPU:

- ▶ 'DBATS CREATED' shows the number of requests that required a database access thread (DBAT) to be created to process the request.
- ▶ 'DISCON (POOL) DBATS REUSED' shows the number of requests that were satisfied by assigning a pooled DBAT to process the request.

The following fields show a queued request waiting for DBATs to get assigned. When the current number for those fields gets high, you need to investigate the effects of having high numbers for those values:

- ▶ 'ACC QU INACT CONNS (TYPE 2)' shows the number of RECEIVE requests on type 2 inactive or new connections that are queued to be serviced by a pooled DBAT.
- ▶ 'CUR QU INACT CONNS (TYPE 2)' gives the current number of type 2 inactive or new connections that are queued waiting for a DBAT.

There are a few fields where you can monitor how DB2 is working against DDF related subsystem parameters, as previously explained for the **-DISPLAY DDF DETAIL** command in 11.2.1, "DDF activity report" on page 161. While the command gives snapshot or cumulative numbers, the trace output also shows you activity based on trace intervals. For V10 or later, the interval is fixed to 1 minute for basic statistics traces:

- ▶ 'DBAT/CONN QUEUED-MAX ACTIVE' gives the number of times a DBAT or connection was queued because it reached the MAXDBAT ZPARM value.
- ▶ 'CONN REJECTED-MAX CONNECTED' gives the number of connections that were rejected because the ZPARM limit for CONDBAT was reached.

- ▶ 'CONN CLOSED - MAX QUEUED' gives the number of queued client connections whose TCP/IP sockets were closed because the system parameter MAXCONQN was exceeded, when connection queue redirect was configured.
- ▶ 'CONN CLOSED - MAX WAIT' gives the number of queued client connections whose TCP/IP socket were closed due to system parameter MAXCONQW being exceeded, when connection queue redirect was configured.

Tip: Monitor your system periodically or when making changes to the system or applications. Validates your subsystem parameter settings using described fields.

Example 11-9 shows you example of global DDF activity report.

Example 11-9 Global DDF activity

GLOBAL DDF ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
DBAT/CONN QUEUED-MAX ACTIVE	0.00	0.00	0.00	N/A
CONN REJECTED-MAX CONNECTED	0.00	0.00	0.00	N/A
CONN CLOSED - MAX QUEUED	0.00	0.00	0.00	N/A
CONN CLOSED - MAX WAIT	0.00	0.00	0.00	N/A
 COLD START CONNECTIONS	0.00	0.00	0.00	0.00
WARM START CONNECTIONS	0.00	0.00	0.00	0.00
RESYNCHRONIZATION ATTEMPTED	0.00	0.00	0.00	0.00
RESYNCHRONIZATION SUCCEEDED	0.00	0.00	0.00	0.00
 CUR TYPE 1 INACTIVE DBATS	0.00	N/A	N/A	N/A
HWM TYPE 1 INACTIVE DBATS	1.00	N/A	N/A	N/A
TYPE 1 CONNECTIONS TERMINAT	0.00	0.00	N/A	N/A
 CUR INACTIVE CONNS (TYPE 2)	2.00	N/A	N/A	N/A
HWM INACTIVE CONNS (TYPE 2)	30.00	N/A	N/A	N/A
ACC QU INACT CONNS (TYPE 2)	60059.00	27.81	N/A	N/A
CUR QU INACT CONNS (TYPE 2)	0.03	N/A	N/A	N/A
MIN QUEUE TIME	0.000005	N/A	N/A	N/A
MAX QUEUE TIME	0.008089	N/A	N/A	N/A
AVG QUEUE TIME	0.000024	N/A	N/A	N/A
HWM QU INACT CONNS (TYPE 2)	15.00	N/A	N/A	N/A
 CUR ACTIVE AND DISCON DBATS	15.82	N/A	N/A	N/A
HWM ACTIVE AND DISCON DBATS	30.00	N/A	N/A	N/A
HWM TOTL REMOTE CONNECTIONS	30.00	N/A	N/A	N/A
 CUR DISCON DBATS NOT IN USE	3.67	N/A	N/A	N/A
HWM DISCON DBATS NOT IN USE	30.00	N/A	N/A	N/A
DBATS CREATED	30.00	N/A	N/A	N/A
DISCON (POOL) DBATS REUSED	60030.00	N/A	N/A	N/A
 CUR ACTIVE DBATS-BND DEALLC	0.00	N/A	N/A	N/A
HWM ACTIVE DBATS-BND DEALLC	0.00	N/A	N/A	N/A

11.2.3 Statistics DRDA remote locations

The OMEGAMON PE statistics report also shows conversation information between DDF and clients. To maintain your performance for DRDA transactions, it is important to determine how efficiently the SQL was processed. You can find such information relating to DRDA messaging and number of requests from this particular trace block.

There is a functionality called block fetch, which is important to significantly decrease the number of messages sent across the network. With block fetch, DB2 groups the rows that are retrieved by an SQL query into as large a “block” of rows as can fit in a message buffer. DB2 can transmit that large block over the network. To use block fetch, DB2 must determine that the cursor is not used for updating or deleting. It is important to make sure your applications that retrieve many rows will utilize block fetch to maintain good performance while retrieving many rows.

If a cursor is not explicitly defined read-only, DB2 can determine that some cursors are implicitly read only because of ORDER BY, GROUP BY, DISTINCT, UNION. For a static SQL application, you can BIND your packages with CURRENTDATA NO to encourage your applications to use block fetch even they include ambiguous cursors.

You can use DRDA remote locations block to see the efficiency of the block fetch. Because block fetch groups the rows, compare the ROWS field with the MESSAGES field to determine the efficiency. The more rows you get with fewer messages, the more efficient you get with your applications. Example 11-10 shows an example of tracing a workload where each SQL is returning small numbers of rows.

Example 11-10 DRDA remote locations

DRDA REMOTE LOCS	SENT	RECEIVED
-----	-----	-----
TRANSACTIONS	N/A	N/A
CONVERSATIONS	0.00	120.00
CONVERSATIONS QUEUED	0.00	
CONVERSATIONS DEALLOCATED	0.00	
SQL STATEMENTS	0.00	405.4K
(SINGLE PHASE) COMMITS	0.00	67516.00
(SINGLE PHASE) ROLLBACKS	0.00	0.00
ROWS	135.1K	0.00
MESSAGES	473.4K	473.4K
BYTES	62480.6K	69742.3K
BLOCKS	270.2K	0.00
MESSAGES IN BUFFER	N/A	
CONT->LIM.BLOCK FETCH SWTCH	N/A	
STATEMENTS BOUND AT SERVER	N/A	
PREPARE REQUEST	N/A	N/A
LAST AGENT REQUEST	N/A	N/A
TWO PHASE COMMIT REQUEST	N/A	N/A
TWO PHASE BACKOUT REQUEST	N/A	N/A
FORGET RESPONSES	N/A	N/A
COMMIT RESPONSES	N/A	N/A
BACKOUT RESPONSES	N/A	N/A
THREAD INDOUBT-REM.L.COORD.	0.00	
COMMITS DONE-REM.LOC.COORD.	N/A	
BACKOUTS DONE-REM.L.COORD.	N/A	

11.2.4 Monitoring using system profiling

Profile tables enable you to monitor the use of system resources and to apply more granularity to the performance-related subsystem parameters to better reflect the need of your distributed applications. The monitored environment is defined by a set of criteria called a profile.

Using system profiling, you can apply the limit values that were previously assigned at system level using DSNZPARMs such as CONDBAT, MAXDBAT, and IDTHTOIN, at a more granular level. The limits can be applied based on the following categories:

- ▶ IP Address (LOCATION)
- ▶ Product Identifier (PRDID)
- ▶ Role and Authorization Identifier (ROLE, AUTHID)
- ▶ Collection ID and Package Name (COLLID, PKGNAME)
- ▶ DB2 client information (CLIENT_APPLNAME, CLIENT_USERID,
- ▶ CLIENT_WORKSTNNAME)

System profiling defines the type of action to take after these thresholds are reached for each category. You can display a warning message or an exception message when CONDBAT, MAXDBAT, and IDTHTION have been exceeded. When you choose to display a warning message, a DSNT771I or DSNT772I message is issued, depending on your settings, and processing continues. In the case of exception processing, a message is displayed to the console and the same action as reaching CONDBAT, MAXDBAT, and IDTHTION will take place.

You can find more information on using system profiles in *DB2 for z/OS and WebSphere Integration for Enterprise Java Applications*, SG24-8074.

11.2.5 Client information and workload management

A common requirement for identifying a user is to determine who is having or causing a problem, for accounting. There are some good reasons to use a generic user ID for the application servers. However, a generic user ID makes it difficult to determine who is having problems using accounting trace data. If the accounting traces are populated with generic user ID and Java processing, it is impossible to determine which application had an issue.

DB2 provides special registers where to put client information, and DRDA provides a solution by providing a way to pass client information in the DRDA data flow that is also written out with application related traces.

DB2 11 enhances client information in DB2 for z/OS by expanding the lengths of the fields listed in Table 11-2. You need DB2 Connect V10.5 FP2 level of clients, such as IBM Data Server Drivers, to set the enhanced client information.

Table 11-2 Client information field length in DB2 11 for z/OS

Property name	Max length in DB2 10	Max length in DB2 11
Client Application Name	32 bytes	255 bytes
Client Accounting Information	200 bytes	255 bytes
Client Correlation Token	N/A	255 bytes
Client Workstation Name	18 bytes	255 bytes
Client User ID	16 bytes	128 bytes


DDF transactions are classified and defined to a WLM service class in the active WLM policy. Use the WLM administrative panels to define service classes, report classes, and classification rules for DDF transactions. Using Client information, the users can assign separate service class to each transaction.

Existing WLM classification attributes cannot handle enhanced client information.

In z/OS V2.1, WLM has been enhanced to handle enhanced client information by adding new attributes as shown in Table 11-3. Other classification attributes provided from previous versions can still be used to classify DRDA transaction using client information, but the length of information you can use is limited.

Table 11-3 DDF work classification attributes for enhanced client information

Attributes	Type	Description
The client accounting information	CAI	The value up to 512 bytes. The value is defined by QWDASUFIX in the DSNDQMDA mapping macro.
The client IP address	CIP	The value up to 39 bytes.
The client user ID	CUI	The value up to 128 bytes. The value is defined by QWHCEUID_Var in the DSNDQWHC mapping macro
The client workstation name	CWN	The value up to 255 bytes. The value is defined by QWHCEUWN_Var in the DSNDQWHC mapping macro.
The client transaction or application name	CTN	The value up to 255 bytes. The value is defined by QWHCEUTX_Var in the DSNDQWHC mapping macro.



Workfiles, RID, and sort pools

This chapter covers past and current changes to the different DB2 “pools.” Customers have struggled to bring their workfile database(s) into harmony between the DB2 releases as they were merged into one database. DB2 10 introduced the ability for various sorting and RID list processes to overflow to the workfile to avoid access path degradation. Various reports and rules of thumb are mentioned.

This chapter contains the following topics:

- ▶ The workfile evolution
- ▶ Sorting
- ▶ RID list processing

12.1 The workfile evolution

In DB2 9, we saw the demise of the TEMP database. The TEMP database was used for declared global temporary tables (DGTT) and declared temp tables for static scrollable cursors only. Declared tables were then moved from their own database to the workfile database. This opened up the possibility for the DGTTs to share the same table spaces as used in sort. The work file is used for sort, star join, trigger, created temporary tables, view materialization, nested table expression, merge join, non-correlated subquery, sparse index, DB2 9 DGTT, and so on. Figure 12-1 depicts the merging of TEMP in the workfile.

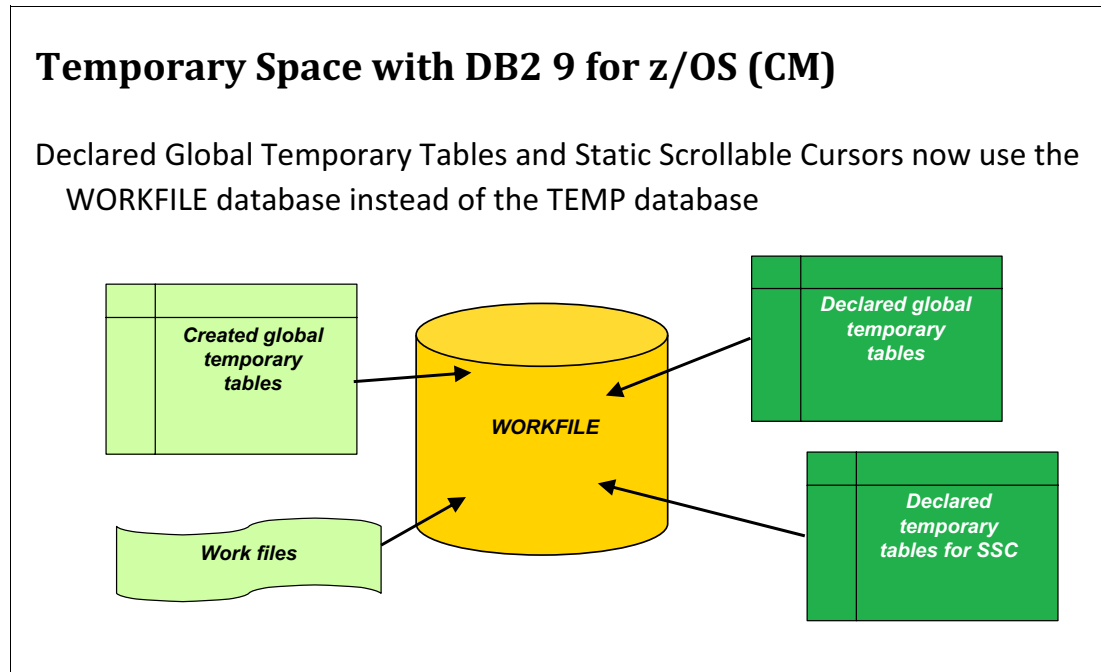


Figure 12-1 DB2 9 workfile and temp picture

This somewhat simplified the infrastructure (eliminating the need for 8k and 16k table spaces) but customers ran into problems where large DGTTs were landing in the same work file as sort work and running out of space, or very large ones were filling up entire workfile table spaces on their own. This is partly because a workfile can span multiple table spaces in the workfile database, while DGTTs cannot span multiple table spaces. So there was a soft separation introduced with PK70060 so that DB2 would *favor* the choice of DB2-managed (STOGROUP) workfile table spaces with SECQTY 0 or user-managed table spaces (regardless of their secondary allocation) for non-DGTT work tables and DB2-managed table spaces with SECQTY > 0 or -1 (or omitted) for DGTT work tables.

User-managed table spaces are treated the same as DB2-managed table spaces with SECQTY 0. Note that if a favored table space does not have room or is unavailable for any reason, DB2 will still try to use a non-favored table space, which could lead to sharing of space between non-DGTT and DGTT applications. This can be monitored by the statistics fields shown in Example 12-1.

Example 12-1 To see if DB2 used a non-favored table space

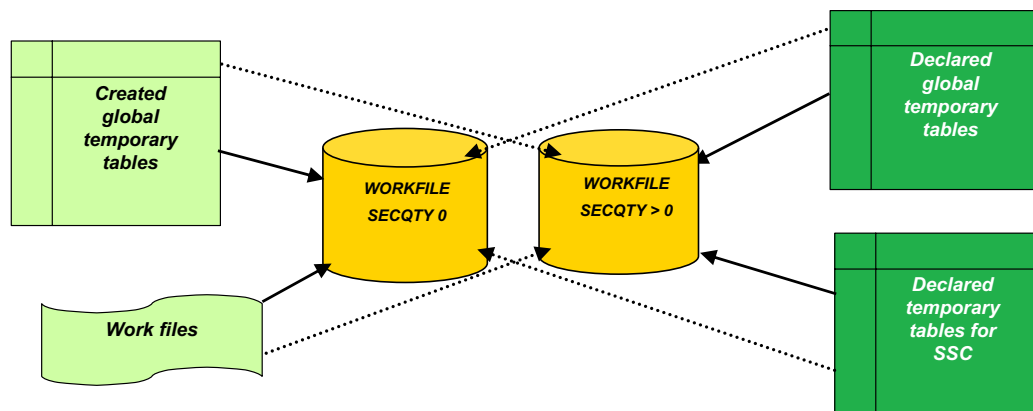
```
QISTWFP3/4= #times DGTT had to be allocated from 0 secqty 4K/32K work file table space
QISTWFP5/6= #times non DGTT had to be allocated from >0 secqty 4k/32K work file table space
```

From DB2 9, if you would prefer to fail the allocations with SQLCODE -904 rather than allowing the allocations to spill over into a non-favored table space, you can specify the DSNZPARM WFDBSEP=YES (PM02528). See Example 12-2.

Temporary Space since DB2 9 for z/OS (CM) ...

APAR PK70060 introduces a soft barrier between DGTT and non-DGTT space use in the workfile database

APAR PM02528 introduces a hard barrier with zPARM WFDBSEP



Recommendation: Define some table spaces with zero secondary quantity and some with non-zero secondary quantity

Figure 12-2 DB2 9 NFM and up view

One way to control the amount of work storage an agent consumes is with a DSNZPARM. Online DSNZPARM MAXTEMPS controls how much space in a workfile database an agent can use (or a family of parallel tasks in case of query CP parallelism). The usage can be for sort workfiles, created global temporary tables (CGTT), declared global temporary tables (DGTT), scrollable cursors result tables, trigger transition tables, and so on. The ZPARM is not granulated to any one type of these uses, instead, it tracks the overall use by the agent.

Still, some people want to limit the growth of DGTTs, while allowing large sorts, and MAXTEMPS is not good enough because it applies to both DGTTs and sort. The problem is compounded by the fact that a small secondary quantity does not necessarily limit the growth when using system managed storage, because the system tries to consolidate contiguous extents. For example, specifying SECQTY 1 does not limit the growth to 123 extents of 1K each, because if the system happens to consolidate the secondary extents, the data set could continue to grow beyond 123K.

A circumvention to that problem with DB2 9 was to avoid using system managed storage, since extent consolidation requires system managed storage. However, DB2 10 solves this problem by introducing the ability to use PBG for work files. Hence, now you can limit the size of a DGTT to as little as 1 GB by setting MAXPARTITIONS=1 and DSSIZE=1GB, and you can use system managed storage. Conversely, you can also overcome the old limit of 64 GB for segmented table spaces if you want to, and you can do so using a single large data set rather than chopping up the table space into lots of 2 GB data sets.

New in DB2 10, UTS PBG is available for DGTG use, as it can help accommodate very large tables. The UTS table spaces will be used only as a last resort for normal sort work if ZPARM WFDBSEP = NO. So DB2 will try to use a segmented table space where SECQTY=0, and if none are found, it will take one with a non-zero SECQTY, and if none of those are available, it will accept a PBG. But, if ZPARM WFDBSEP = YES, then sort work will not use the UTS table space and DB2 will issue a -904 SQL CODE. Hence classic segmented workfile table spaces must also be defined for sort workfile clients if WFDBSEP=YES.

The suggestion is to have some table spaces with a secondary quantity and some without unless you have no DGTGs or DTTs for static scrollable cursors. Do this because whether or not the hard separation is there, the code path is there to favor SECQTY 0 or not 0, and it would still be invoked even if the “less favored” option is not there.

12.1.1 Tuning the workfiles

Tip: Here is an info APAR regarding workfile best practices in DB2 9 and DB2 10:

<http://www-01.ibm.com/support/docview.wss?uid=isgl1114587&myns=apar&mynp=D0CTYP&component&mync=E>

Even with these separations, the primary quantity specification could affect performance to some extent. DB2 managed workfiles’ primary quantity can make a difference even when SECQTY= 0. DB2 may try to extend the table space even if the size is below 2 GB, which can lead to unnecessary space growth and CPU cycles. But at the same time, you want to make the primary as large as possible for efficient use of the table spaces.

Suggested primary and secondary space allocation for workfile table spaces used by workfiles:

4 KB pagesize - PRIQTY 2096640 (KB)	SECQTY 0
32 KB pagesize - PRIQTY 2096512 (KB)	SECQTY 0

Suggested primary and secondary space allocation for workfile table spaces used by DGTGs:

4 KB pagesize - any PRIQTY > 0 or -1 and any SECQTY > 0 or -1 (or omitted)
32 KB pagesize - any PRIQTY > 0 or -1 and any SECQTY > 0 or -1 (or omitted)

Along with DB2 9 also came the preference of a 32 KB page over a 4 KB page for work files. Basically any workfile record less than 100 bytes will go to a 4 KB page workfile and anything larger to a 32 KB page workfile. This of course not only affects the number and size of the workfile table spaces you must create, but also the size and parameters in use for one of your 32k buffer pools.

The suggestion during migration to DB2 9, or skip release to DB2 10, was to have the same amount of physical space in the 32 K workfiles and workfile buffer pool as you did in the 4 K. Then after migration, you can assess the need for the 4 K pool and objects based on their use. IFCID 002 in statistics shows you the current storage in use broken down by page size, and this example is typical, where about 80% of the sorts end up in the 32 KB workfiles.

Example 12-2 shows an example of workfile page size distribution.

Example 12-2 Typical 4 KB versus 32 KB distribution

WORK FILE DATABASE	QUANTITY
-----	-----
MAX TOTAL STORAGE USED (KB)	1780.6K
MAX DGTT STOR USED (KB)	8704.00
MAX WF STOR USED (KB)	1780.6K
CUR TOTAL STORAGE USED (KB)	11584.00
CUR DGTT STOR USED (KB)	0.00
CUR WF STOR USED (KB)	11584.00
STORAGE IN 4K TS (KB)	22.93
STORAGE IN 32K TS (KB)	1548.15
 4K USED INSTEAD OF 32K TS	 0.00
32K USED INSTEAD OF 4K TS	0.00

Tip: One scenario you want to avoid is having the field related to 4 KB used instead of 32 KB populated. This means that DB2 wanted a 32 KB workfile, but it was not available, so you have lost the efficiency of fitting more work elements into 1 page, as well as the I/O savings of 2 16-KB unit control blocks (UCBs), instead of 8 4-KB UCBs. The solution to such a problem is simply more and larger 32 KB workfiles.

The reverse is even worse, because, for example, if the record size were only 50 bytes, DB2 would only use 12750 bytes out of each 32K page (since DB2 is limited to 255 rows per page), thereby wasting DASD space as well as making performance worse.

To allow more concurrent users of workfile table spaces and better management of the storage, assuming the number of rows in the logical workfile tables and DGTTs typically small, it is recommended that in NFM a SEGSIZE value smaller than default 16 be set, especially considering most are using a 32 KB workfile. Increasing the number of workfiles can help avoid space map page latch contention, and even show up as LC25.

Workfile physical layout should also be taken into account for concurrency, namely by spreading them across multiple DASD storage extent pools. If there are very large sorts going on and they are hitting the same volumes in the backend the disc cache hit ratio could be degraded and I/O response time could increase substantially. HyperPAV is recommended to avoid IOSQ time with your work files. If you observe IOSQ time for your workfiles even with HyperPAV (which is unlikely), then you need to spread your workfiles across Logical Control Units. However, if you see I/O disconnect time for your work files, then it means that you also need to spread the workfiles across physical disk extent pools.

12.2 Sorting

Sorting is part of the Relational Data Services (RDS) component of DB2. It is a tournament style sort consisting of two phases. DB2 scans the data for qualifying records and places qualifying records into workfiles. This process of scanning and populating the workfiles is the initialization phase of the sort. The second phase of the sort process is the merge phase. The term tournament comes in here as elements go head-to-head to determine who is next in line in the sorted results. DB2 uses what is called a binary sort tree here for efficiency.

The sorting of rows involves 3 separate but very much related DB2 resources, including the sort pool, workfile buffer pools, and physical workfiles on DASD. It is a bit of a balancing act between all 3, and rarely can you tune any 1 piece to alleviate any dependence on the others.

- The sort pool has a default of 2 MB until DB2 10 where it becomes 10 MB with a maximum of 128 MB.¹ Since the sort records and tree structure are in 2 different blocks of storage, each run can take up to 64 MB (as seen in Figure 12-3).

Figure 12-3 shows the sort phases.

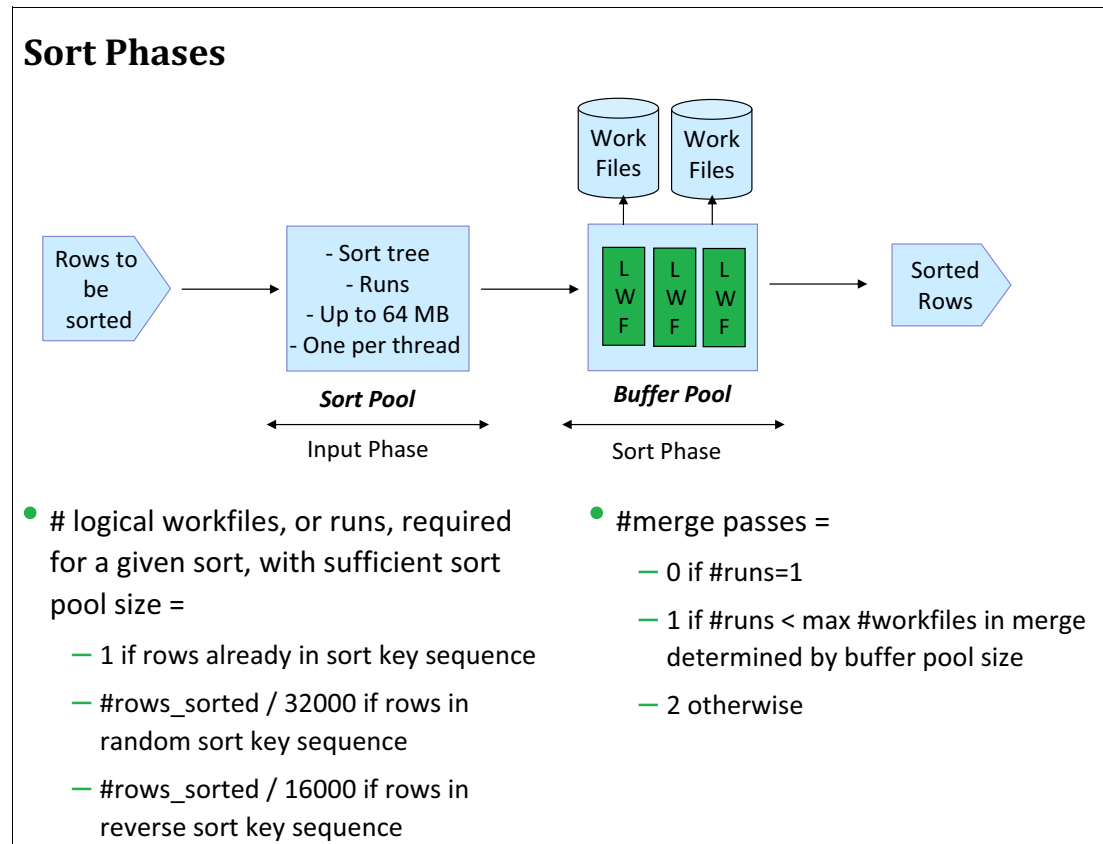


Figure 12-3 Sort phases

The first step of the sort, the initialization phase, passes the rows through the sort pool. Each of these runs becomes a logical workfile (LWF) in the buffer pool.

Then the second sort step, merge pass, takes these runs and combines them in the correct final order to pass back to the SQL requestor. If they are already in sequence then we can get away with 1 logical workfile in the buffer pool. There is a concept of a node within the sort phase, and each node houses 2 records which are competing for who is higher in the sequence. If the rows are in random sequence, we can do 32,000 nodes (64,000 records) at a time, so divide the number of rows by 32,000 to see how many runs, and then merge passes you will need in the buffer pool. Remember that SRTPOOL is the max size for each concurrent sort so look out for real storage usage. The default for SRTPOOL= 10 MB, max of 128 MB in DB2 10 and DB2 11. The sort pool grows in increments of 4 K blocks, and 128 MB is 32,000 4 K blocks.

¹ The MAXSORT_IN_MEMORY subsystem parameter, new with DB2 11, specifies the maximum allocation of storage in kilobytes for a query that contains an ORDER BY clause, a GROUP BY clause, or both. The storage is allocated only during the processing of the query. Increasing the value in this field can improve performance of such queries but might require large amounts of real storage when several such queries run simultaneously. Acceptable values: 1000 up to the value specified in SORT POOL SIZE.

In the DB2 installation guide, there are examples of estimating max sort size:

- ▶ 16 + sort key length + sort data length * the number of rows to be sorted
- Since DB2 can handle up to 32000 random rows in a run, then multiply this max sort size by 32000 to determine the minimum amount of runs necessary and use it to work on buffer pool sizing (Example 12-3). The only reason you would want the sort pool bigger is if you had a sort request for over 64000 rows and they were in sort order, in which case you are trying to force it through in just one run.

During the second sorting step, we are dependent on the number of logical workfiles we can create in the workfile buffer pool. Looking at Figure 12-3 on page 172, the number of runs from the first step relates to the number of LWFs. If we can do it all in one pass, meaning the rows were already in sorted sequence, then merge pass =0 in the statistics report. If we had more than enough workfiles in the buffer pool to do our runs, then merge pass = 1. Otherwise if we ran out of workfiles in the buffer pool and had to go to disc, then our merge pass = 2.

Rules of thumb for statistics for sorting and the sort buffer pool:

- ▶ Merge Passes Degraded < 1 to 5% of Merge Pass Requested:
MERGE PASSES DEGRADED = number of times merge pass > 1 the buffer pool could not support the max number of workfiles requested.
 - Max workfiles allowed = ((sequential steal threshold * buffer pool size - (2 * workfiles already allocated)) / 16
 - Each LWF requires a minimum of 16 sequential pages.
- ▶ Workfile Req. Rejected < 1 to 5% of Workfile Req. All Merge Passes:
WORKFILE REQUESTS REJECTED = number of workfile requests rejected because buffer pool is too small to support concurrent sort activity.
- ▶ Workfile Prefetch Not Scheduled < 1 to 5% of Seq Pref Reads:
WORKFILE PREFETCH NOT SCHEDULED = number of times sequential prefetch was not scheduled for work file because of buffer pool shortage.
 - Each workfile needs 16 dedicated buffers to perform efficient prefetch.

Take these recommendations and correlate them to what is seen in the statistics report for the work file buffer pool.

Example 12-3 Workfile BP sorting

BP7	SORT/MERGE	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----	-----
MAX WORKFILES CONCURR. USED		778.77	N/A	N/A	N/A
MERGE PASSES REQUESTED		52.00	0.00	0.30	0.08
MERGE PASS DEGRADED-LOW BUF		1.00	0.00	0.01	0.00
WORKFILE REQ.REJCTD-LOW BUF		1426.00	0.06	8.29	2.33
WORKFILE REQ-ALL MERGE PASS		20004.00	0.85	116.30	32.69
WORKFILE NOT CREATED-NO BUF		0.00	0.00	0.00	0.00
WORKFILE PRF NOT SCHEDULED		0.00	0.00	0.00	0.00

Performance trace Class 9 is strictly sort information with IFCIDs 26-28 (multiple distinct sort activity), 95-96(sort detail), and 106. See Example 12-4.

Example 12-4 IFCID 95-96 sorting

MEMBER : SAU85C	MAX RETURN CODE : 0	WORKFILES : 1.00	RECORDS : 77.00
TOTAL SORTS : 1	INITIAL WORKFILES : 1	RECORD SIZE : 832	SORT TYPE : ESA
SORT KEYS : 2.0	SORT COLUMNS : 50.00	KEY SIZE : 10.00	MERGE PASSES : 27.00

Much of this is buffer pool dependent as far as how many concurrent LWFs are allowed. This involves the size of the buffer pool and portion that is VPSEQT. In the rules of thumb above it is assumed that VDWQT=10 and DWQT=50, and VPSEQT=95% which is considered a defensive setting to protect against large and unpredictable sorts. Other than this, the option is to increase the size of the buffer pool, see 12.2.1, “Buffer pool settings” on page 174 for more discussion.

12.2.1 Buffer pool settings

Despite there being an entire chapter in this book on buffer pool tuning, it is appropriate to include a section here on the workfile buffer pool to alleviate confusion in the other sections.

There are two trains of thought regarding workfile buffer pool tuning:

- Very aggressive finite tuning with a well understood workload:

This is seen in very few customer shops these days, but the point is to ensure that all of the sorting and logical workfiles are contained in memory, in the sort pool and buffer pools, without spilling over to DASD workfiles. This is likely not realistic if there are DGTTs or static scrollable cursors in the environment. The settings would look something like this:

- VPSEQT= 100%
- DWQT = 70-90%
- VDWQT= 90%

The downside here is that, if you do see I/Os, then the strategy is not working and it is likely you will hit other buffer pool thresholds such as prefetch disabled, or even the data manager threshold as described in Figure 10-1 on page 131, thus destroying performance and affecting system stability.

- Conservative approach with a mix of sort and DGTT work, and unpredictable sort sizes (real life):

This is the common approach due to mixed workloads and dynamic SQL:

- VPSEQT= 90-98%
- DWQT= 10%
- VDWQT= 50%

With this approach, some performance is sacrificed for stability and versatility.

There are some items to look for the in the read operations of the buffer pool statistics shown in Example 12-5.

Rules of thumb for buffer pool read statistics:

- ▶ SYNCHRONOUS READS should be < 1-5% of pages read by prefetch
This symptom could point toward the buffer pool being too small, or having too few physical workfiles to write out to and we are overwriting the same page multiple times
- ▶ $x.PR.F.PAGES\ READ/x.PR.F.READ > 4(4k\ page)\ or\ 2\ (32k\ page)$
Here you look at the number of pages read per prefetch request, 'x' refers to 'S' or 'D' for sequential or dynamic prefetch. This also means the size of the buffer pool should be increased.

Example 12-5 Workfile BP stats

BP7	READ OPERATIONS	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----	-----
BPOOL	HIT RATIO (%)	63.90			
BPOOL	HIT RATIO (%) SEQU	63.84			
BPOOL	HIT RATIO (%) RANDOM	71.10			
GETPAGE	REQUEST	42229.6K	1804.68	245.5K	36.8K
GETPAGE	REQUEST-SEQUENTIAL	41902.9K	1790.72	243.6K	36.5K
GETPAGE	REQUEST-RANDOM	326.7K	13.96	1899.46	284.59
SYNCHRONOUS	READS	94434.00	4.04	549.03	82.26
SYNCHRON.	READS-SEQUENTIAL	81745.00	3.49	475.26	71.21
SYNCHRON.	READS-RANDOM	12689.00	0.54	73.77	11.05
GETPAGE	PER SYN.READ-RANDOM	25.75			
SEQUENTIAL	PREFETCH REQUEST	2421.0K	103.46	14.1K	2108.91
SEQUENTIAL	PREFETCH READS	1912.6K	81.73	11.1K	1666.02
PAGES READ VIA SEQ.PREFETCH		15152.4K	647.54	88.1K	13.2K
S.PR.F.PAGES READ/S.PR.F.READ		7.92			
LIST	PREFETCH REQUESTS	0.00	0.00	0.00	0.00
LIST	PREFETCH READS	0.00	0.00	0.00	0.00
PAGES READ VIA LIST PREFETCH		0.00	0.00	0.00	0.00
L.PR.F.PAGES	READ/L.PR.F.READ	N/C			
DYNAMIC	PREFETCH REQUESTED	0.00	0.00	0.00	0.00
DYNAMIC	PREFETCH READS	0.00	0.00	0.00	0.00
PAGES READ VIA DYN.PREFETCH		0.00	0.00	0.00	0.00
D.PR.F.PAGES READ/D.PR.F.READ		N/C			
PREF.DISABLED-NO	BUFFER	0.00	0.00	0.00	0.00
PREF.DISABLED-NO	READ ENG	0.00	0.00	0.00	0.00
PAGE-INS	REQUIRED FOR READ	0.00	0.00	0.00	0.00

In this example, the synchronous reads were 0.6% of the prefetched pages, so we are below the threshold for serious investigation, assuming the customer is not trying to keep the workfiles in memory.

The number of pages read in per prefetch operation is 7.9 so that is above our 4 page minimum in prefetch quantity for a 4 KB page.

12.2.2 DB2 11

DB2 11 brings in some new fields that could be useful in tracking overall system utilization by sorting and DGTTS, as well as individual offenders.

Having the ability for the DBAs to track the total storage configured versus the max used for DGTTS, sorts, and the grand total can help them in the following ways:

- ▶ Better plan for capacity upgrades
- ▶ Trend usage and growth over time and correlate buffer pool statistics to the overall utilization of workfiles
- ▶ Monitor for drastic changes during application or release migrations that warrant further investigation

There are also some extra preventative measures built into DB2 11. Two new DSNZPARMs for the system and thread level can issue warnings based on the percentage of the workfile or DGTTS storage that is consumed, instead of just abending the thread. If the WFDBSEP ZPARM is set to YES, then the threshold applies to both the DGTTS and workfile storage individually, so it kicks in if either category of storage is infringed upon. If WFDBSEP = NO, then the thresholds apply to the total storage configured. These ZPARMS are as follows:

- ▶ **WFSTGUSE_SYSTEM_THRESHOLD** - % of workfile storage consumed at the system level, when DB2 will issue the message:

```
DSNIO52I csect-name AN AGENT HAS EXCEEDED THE WARNING THRESHOLD FOR STORAGE USE  
IN WORK FILE DATABASE.....FOR DECLARED GLOBAL TEMP TABLES or WORK FILES or  
DECLARED GLOBAL TEMP TABLES AND WORK FILES with thread information
```
- ▶ **WFSTGUSE_AGENT_THRESHOLD** - % of workfile storage consumed by a single agent, when DB2 will issue the message:

```
DSNIO53I csect-name THE DB2 SUBSYSTEM HAS EXCEEDED THE THRESHOLD FOR STORAGE  
USE IN WORK FILE DATABASE...
```

The *DB2 11 for z/OS Technical Overview*, SG24-8180, has more details on these settings and the nuances of how the percentage of total storage is calculated (such as needing all of the workfile data sets to be open to calculate it), and frequency of the messages.

These two DSNZPARMs are a more gentle and proactive approach to monitoring out-of-bounds conditions than the previous MAXTEMPS DSNZPARM. This parameter would terminate any agent who used more storage than was specified in the installation parameter with an SQLCODE -904 and 00C90305 reason code. This is good as a last resort, but some customers were hesitant to set it for fear of disrupting service. In DB2 11 statistics, we can now track the number of times that MAXTEMPS parameter was hit by the 'NUMBER OF LIMIT EXCEEDED' field (QISTWFNE). See Example 12-6.

Example 12-6 OMEGAMON PE V5R2 and DB2 11 workfile stats

WORKFILE DATABASE		QUANTITY	/SECOND	/THREAD	/COMMIT
-----		-----	-----	-----	-----
TOTAL STORAGE CONFIG	(KB)	134.3M	N/A	N/A	N/A
TOT DGTTS STOR CONFIG	(KB)	134.2M	N/A	N/A	N/A
TOT WF STOR CONFIG	(KB)	40960.00	N/A	N/A	N/A
TOTAL STORAGE THRESHOLD	(%)	2.00	N/A	N/A	N/A
MAX TOTAL STORAGE USED	(KB)	1780.6K	N/A	N/A	N/A
MAX DGTTS STOR USED	(KB)	8704.00	N/A	N/A	N/A
MAX WF STOR USED	(KB)	1780.6K	N/A	N/A	N/A

CUR TOTAL STORAGE USED (KB)	11584.00	N/A	N/A	N/A
CUR DGT T STOR USED (KB)	0.00	N/A	N/A	N/A
CUR WF STOR USED (KB)	11584.00	N/A	N/A	N/A
STORAGE IN 4K TS (KB)	11584.00	N/A	N/A	N/A
STORAGE IN 32K TS (KB)	0.00	N/A	N/A	N/A
4K USED INSTEAD OF 32K TS	0.00	0.00	0.00	0.00
32K USED INSTEAD OF 4K TS	0.00	0.00	0.00	0.00
MAX ACTIVE (DM) IN-MEMORY	4.00	N/A	N/A	N/A
MAX ACT (NONSORT) IN-MEM	1.00	N/A	N/A	N/A
CUR ACTIVE (DM) IN-MEMORY	0.00	N/A	N/A	N/A
CUR ACT (NONSORT) IN-MEM	0.00	N/A	N/A	N/A
MAX STOR (DM) IN-MEM (KB)	128.00	N/A	N/A	N/A
CUR STOR (DM) IN-MEM (KB)	0.00	N/A	N/A	N/A
MAX ACTIVE (SORT) IN-MEMORY	1.00	N/A	N/A	N/A
CUR ACTIVE (SORT) IN-MEMORY	0.00	N/A	N/A	N/A
MAX STOR (SORT) IN-MEM (KB)	2.77	N/A	N/A	N/A
CUR STOR (SORT) IN-MEM (KB)	0.00	N/A	N/A	N/A
IN-MEM (NONSORT) OVERFLOWED	0.00	0.00	0.00	0.00
IN-MEM WORKF NOT CREATED	0.00	0.00	0.00	0.00
AGENT STORAGE CONFIG (KB)	0.00	N/A	N/A	N/A
NUMBER OF LIMIT EXCEEDED	0.00	0.00	0.00	0.00
AGENT STORAGE THRESHOLD (%)	10.00	N/A	N/A	N/A
MAX AGENT STORAGE USED (KB)	1780.6K	N/A	N/A	N/A

The following other fields in this report could be useful:

► **MAX STOR (SORT) IN-MEM (KB)**

The in-memory workfile could be up to 32k in size in DB2 9. DB2 10 allows for up to 1 MB in memory. There is nothing the customer can do to tune this number, but it may give you an idea of the size of some of your sorting activities.

12.3 RID list processing

RID list processing is used in List Prefetch, Multiple-Index Access, Hybrid join, Dynamic index ANDing, and in some unique key handling. Each RID block stored in the work file occupies 32 KB of work file storage and contains 6524 RIDs. The failure of a RID list was quite severe in DB2 9 and prior as it failed over to a relational scan of the table in most cases. In DB2 10, we allowed it to overflow to a workfile after filling the RID pool (default of 400,00KB in DB2 10), and it did not fall back to a relational scan until the MAXTEMPS_RID ZPARM limit was hit. Specifying a value of 10,000 for the MAXTEMPS_RID limits the number of RIDs that are allowed to be stored in the work file to 6,524,000 (about 312.5 MB). The rough size for space needed in the RID pool is the number of RIDs * 2 * 5.

The optimizer considers RID list access at bind or prepare time only when it assumes that no more than 50% of the RID pool would be consumed by this access path, and that it would not scan more than 25% of the rows (index entries) in the table. Since this decision is based on bind time statistics, and the RID list failures are runtime issues, obviously RUNSTATS can play a large role in the success of RID list processing.

With the enhancement to overflow to the workfile database (into 32k pages), the failures are not as degrading to the application, but many of these overflows could impact other work in the 32k workfile buffer pool supporting the workfiles, as well as the workfile utilization. So failures should still be monitored and application groups encouraged to tune out the failures. The field MAX RID BLOCKS OVERFLOWED in DB2 11 will tell you how many 32k blocks ended up in a workfile, and can then calculate the total storage used. There are several new fields in DB2 11, and those will be underlined as in Example 12-7. Those in **bold** are the fields that relate to tuning opportunities.

Example 12-7 RID list processing statistics

RID LIST PROCESSING	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
SUCCESSFUL	0.00	0.00	0.00	0.00
NOT USED-NO STORAGE	0.00	0.00	0.00	0.00
NOT USED-MAX LIMIT	0.00	0.00	0.00	0.00
MAX RID BLOCKS ALLOCATED	3.00	N/A	N/A	N/A
CURRENT RID BLOCKS ALLOCAT.	0.00	N/A	N/A	N/A
<u>MAX RID BLOCKS OVERFLOWED</u>	<u>0.00</u>	<u>N/A</u>	<u>N/A</u>	<u>N/A</u>
<u>CURRENT RID BLOCKS OVERFL.</u>	<u>0.00</u>	<u>N/A</u>	<u>N/A</u>	<u>N/A</u>
TERMINATED-NO STORAGE	0.00	0.00	0.00	0.00
TERMINATED-EXCEED RDS LIMIT	0.00	0.00	0.00	0.00
TERMINATED-EXCEED DM LIMIT	0.00	0.00	0.00	0.00
TERMINATED-EXCEED PROC.LIM.	0.00	0.00	0.00	0.00
<u>OVERFLOWED-NO STORAGE</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>
<u>OVERFLOWED-MAX LIMIT</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>
<u>INTERRUPTED-NO STORAGE</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>
<u>INTERRUPTED-MAX LIMIT</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>
<u>SKIPPED-INDEX KNOWN</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>	<u>0.00</u>

- ▶ **TERMINATED-NO STORAGE:** This means that there is not enough 31-bit storage to support the RID list. In DB2 9 the RID MAPs were the only portion below the bar. This moved above the bar starting in DB2 10, so this should not be an issue.
- ▶ **TERMINATED-EXCEED DM LIMIT:** This is the physical limit of the RID pool, regardless of the physical size of MAXRBLK, limited at just over 26 million RIDs
- ▶ **TERMINATED-EXCEED RDS LIMIT:** The matching index scan part of the RID list processing scanned more than 25% of the index.
This is a perfect example of where RUNSTATS at bind time differed from the current size of the table. As the objects grow over time, real-time stats should be used to monitor the need for RUNSTATS. If this does not solve the issue, the access path itself should be revisited due to the fluctuation in size.
- ▶ **TERMINATED-EXCEED PROC.LIM:** The DSNZPARM RID POOL SIZE has been hit, and in DB2 10 and DB2 11, the excess will spill to a workfile unless MAXTEMPS_RID constrains it. This could be due to its shear size, or more likely to concurrent RID list processing.
- ▶ **MAXTEMPS_RID threshold.** See 12.3, “RID list processing” on page 177.
If you have overflowed and filled up the amount of work file storage you allowed, then the statistics and/or access path should be investigated.



Virtual and real storage

This chapter discusses how to monitor both real and virtual storage in DB2. Prior to DB2 10 the emphasis was on virtual, but now it is shifting more to real storage. We consider what components you should be concerned with and how to look at them.

This chapter contains the following topics:

- ▶ Storage concerns for both real and virtual
- ▶ Monitoring real storage usage
- ▶ Monitoring virtual storage usage

13.1 Storage concerns

Real storage is not just the concern of the capacity planning and MVS team; nor is what DB2 consumes solely the responsibility of the DB2 systems programmer. Proper planning and sizing should be a joint venture.

13.1.1 Real storage concerns

The issues seen here are much more straightforward to explain, and measure in some cases. Prior to DB2 10, the discussions between DB2 and the z/OS capacity team revolved around ensuring that there was ample room for the DB2 buffer pools to be backed by real storage. If this was the case, then the next logical step was to page fix the pools in order of I/O intensity as shown in Chapter 10, “Buffer pools and group buffer pools” on page 125.

DB2 10 brings with it the possibility of consolidation of data sharing members due to the vertical scalability, so not only members but also LPARs may be consolidated. Not only would there be more threads, a larger subsystem footprint, and the supporting application towers coming in, but the local buffer pools would need to be adjusted as well. Even in DB2 9, customers complained that a 16 GB address space DUMP was disruptive, and few could set that much aside in case of a crash during a busy period.

Paging out to AUX affects class 3 unaccounted for time, transaction times, and z/OS resources. A DUMP occurring during a time of peak activity could cause work in the LPAR to become non-dispatchable until the DUMP completes, depending on the options chosen. Provisioning storage for the good as well as the bad times is key, and with DB2 10, there are many performance gains to be made by utilizing the real storage at your disposal.

13.1.2 Virtual storage concerns

Customers have run into many problems in the past due to a lack of real storage, and the proper monitoring of virtual storage. Virtual storage constraint became an issue if CTHREAD and MAXDBAT were set too high, allowing too many local or distributed threads into the system, respectively. This was further compounded based on the change in workload from local CICS attach threads that might be tens to hundreds of kilobytes in size to remote JAVA threads that could surpass 1 megabyte in thread footprint size.

A common misconception is that by raising these thread limits, you can increase throughput. This is only true to a point where, based on class 2 CPU time spent in DB2, the arrival rate begins to surpass the exit rate of transactions, and they begin to queue inside DB2. This buildup of threads within DB2 is acceptable if the thread limits are set correctly to protect from exhausting the virtual storage. At some point the threads need to queue outside of DB2, which is quite acceptable for remote threads who may hold a connection without an active database access thread in DB2, and there is a short queue time as a new request is associated with a DBAT. This may be unacceptable for CICS, as it appears as denial of service since each request represents a real thread.

Many times when application developers or end users complain that they need more concurrency, or if the DBA notices that a thread limit has been hit, the first reaction is to increase the limit, which then simply puts more burden on the LPARs processing power, and can slow everyone down. If threads pile up in DB2 without getting any real work done, then not only can the virtual storage be exhausted but lock and latch counts may also impede concurrency.

Another inhibitor to the number of concurrent threads are other address spaces using common storage below the bar, such as MQ, WebSphere, CICS, and IMS. The use of ECSA storage by these products eats into the EPVT (extended region size) where DB2 threads can allocate their storage. Some customers with many CICS regions on an LPAR with several DB2s were anxiously awaiting DB2 10 in order to move the thread storage above the 2 GB bar and then increase ECSA at the LPAR level in order to give the storage to these CICS regions.

13.2 Monitoring real storage usage

Sizing the available storage on the LPAR to fit a new maximum thread limit is not as easy as it sounds, but we will look at how to address this. In short, there is no magic bullet, and the thread footprint estimates we used in previous versions are vastly skewed now due to the 64-bit storage pool allocation. But we can also use the data in the IFCID 225 records, or the MEMU2 REXX (page 192) to map the real storage available versus what DB2 is using, as well as that storage that has been paged out to auxiliary storage.

Figure 13-1 shows several days worth of IFCID 225 data from a customer on DB2 10. The RMF Monitor III STORF (storage frames report) can show you in real time which address spaces are occupying AUX slots (*Resource Measurement Facility Report Analysis z/OS 1.13*, SC33-7991-19). RMF monitoring could be used for a snap shot or to track overall storage usage, but this graph makes it simple to compare the trends. The two parallel lines at the top of the chart show that the total LPAR storage usage and DB2 usage are tracking in unison. These lines are inversely proportional to the more jagged line below, which represents the storage that is available on the LPAR.

If the REALAVAIL goes to zero, then we will page out to AUX. Depending on who is requesting the storage and your WLM setting for Storage Critical workloads, DB2 or another address space may be pushed out. If all of your buffer pools are page fixed, then it will likely be a portion of one of the thread pools within DB2. If it is the case that you are tight on real storage, and DB2 storage continues to get pushed to AUX, the WLM Storage Critical can help improve DB2 performance.

The arrows in Figure 13-1 point out a 07:05 in the morning where the REAL available storage on the LPAR dropped to zero. We then see the AUX storage in use by DB2 go from roughly 2700MB up to 3500MB, and increase of 800MB paged out to AUX storage. Since the AUX storage will be attributed to the DB2 address space until it is stopped and started again it does not mean that there is 3500MB of storage that DB2 is using in AUX currently, just that at some point it has touched 3500MB worth of DASD storage.

However, this is still an unenviable place to be. If DB2 needs to utilize any of that storage it will cost CPU instructions to fix the number of frames it needs in memory as well as an I/O out to disc for each 4 KB of storage it is trying to retrieve. This would certainly result in an LPAR wide slow down if a sudden rush of threads came into DB2. And of course if the DBM1 address space dumped at that time and it needed 16GB of real memory to capture it in DB2 10, then the LPAR would see a dramatic performance impact. The impact would be from the DUMP utilizing all the storage currently, and then having to page back in all the work that was pushed out to DASD.

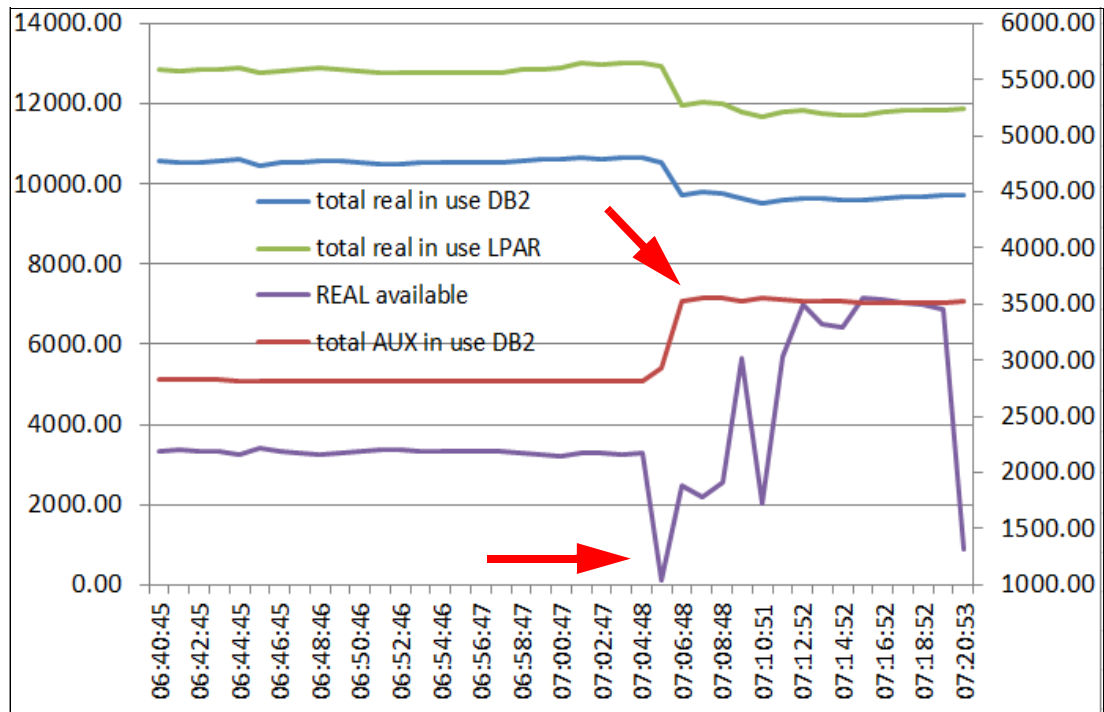


Figure 13-1 Aux storage

Looking at Figure 13-1, it is important in DB2 10 and up to be able to track the real in use on the LPAR versus the AUX and pinpoint the time of the trade-off (in DB2 and elsewhere). Here is the list of fields used to calculate the 4 lines plotted broken out by the IFCID field, the OMEGAMON Performance Expert field, the OMEGAMON PE performance database column name, and the name used in the column name of the MEMU2 REXX exec:

1. Total real storage used by DB2 (add them together for the first line), see Table 13-1.

Table 13-1 Total Real in use by DB2 fields to sum

IFCID field	PM field	PDB column name	MEMU description
QW0225RL	QW0225RL	REAL_STORAGE_FRAME	ASID DBM1 REAL in use for 31 and 64-bit priv
QW0225RL	QW0225RL	DIST_REAL_FRAME	ASID DIST REAL in use for 31 and 64-bit priv
QW0225ShrStg_Real	SW225SSR	A2GB_SHR_REALF_TS	REAL in use for 64-bit shared
QW0225ShrStkStg_Real	SW225KSR	A2GB_SHR_REALF_STK	REAL in use for 64-bit shared stack
QW0225ComStg_Real	SW225CSR	A2GB_COMMON_REALF	REAL in use for 64-bit common

2. Total Real in use by the LPAR (add them together for the second line), see Table 13-2.

Table 13-2 Total Real in use by LPAR fields to sum

IFCID field	PM field	PDB column name	MEMU description
QW0225RL	QW0225RL	REAL_STORAGE_FRAME	ASID DBM1 REAL in use for 31 and 64-bit priv
QW0225RL	QW0225RL	DIST_REAL_FRAME	ASID DIST REAL in use for 31 and 64-bit priv
QW0225SHRINREAL	SW225SRL	A2GB_SHR_PGS_BACKD	REAL in use for 64-bit shared - LPAR
QW0225ComStg_Rea l	SW225CSR	A2GB_COMMON_REALF	REAL in use for 64-bit common

3. Total Real available on the LPAR (this is the field for the third series in the graph), see Table 13-3.

Table 13-3 Total Real available on the LPAR field

IFCID field	PM field	PDB column name	MEMU description
QW0225_REALAVAIL	S225RLAV	QW0225_REALAVAIL	REALAVAIL (S)

4. Total AUX in use by DB2 (sum these to make the fourth line on the graph), see Table 13-4.

Table 13-4 Total AUX in use by DB2

IFCID field	PM field	PDB column name	MEMU description
QW0225AX	QW0225AX	AUX_STORAGE_SLOT	ASID DBM1 AUX in use for 31 and 64-bit priv
QW0225AX	QW0225AX	DIST_AUX_SLOT	ASID DIST AUX in use for 31 and 64-bit priv
QW0225ShrStg_Aux	SW225SSA	A2GB_SHR_AUXS_TS	AUX in use for 64-bit shared
QW0225ShrStkStg_A ux	SW225KSA	A2GB_SHR_AUXS_STK	AUX in use for 64-bit shared stack
QW0225ComStg_Aux	SW225CSA	A2GB_COMMON_AUXS	AUX in use for 64-bit common

- This field is not graphed. It might also be useful to see the total AUX storage on the LPAR because something else might have been paged out to AUX aside from DB2. See Table 13-5.

Table 13-5 Total AUX on the LPAR

IFCID field	PM field	PDB column name	MEMU description
QW0225AX	QW0225AX	AUX_STORAGE_SLOT	ASID DBM1 AUX in use for 31 and 64-bit priv
QW0225AX	QW0225AX	DIST_AUX_SLOT	ASID DIST AUX in use for 31 and 64-bit priv
QW0225SHRAUXSLO TS	SW225SAX	A2GB_SHR_AUX_SLOTS	AUX in use for 64-bit shared - LPAR

13.2.1 Maintaining control over the system use of real storage

There are settings in the DB2 DSNZPARMs as well as the sort products customers use in MVS that can affect the amount of real storage available to DB2 and the rest of the LPAR. It is important to be able to balance each MVS component's storage use in order to balance their performance and the relative impact to the other components.

► DB2 settings:

- REALSTORAGE_MANAGEMENT = AUTO (default):

This functionality was introduced in DB2 10 with ZPARM options of OFF/AUTO/ON, and set to AUTO by default. The goal was to monitor and react to how DB2 was managing cached storage based on central storage conditions within the environment. Returning storage back to z/OS ensures that DB2 did not continue, or appear to continue growing over time. If this was turned OFF, then we would only free off storage when the LPAR was under stress for real storage and paging to AUX by intercepting the ENF55 message from MVS. AUTO returns the real frames if there is heavy paging to AUX or REALSTORAGE_MAX is approached. If it was set ON, then we would go through what is called DISCARD MODE constantly, which would unback the storage and give it back to MVS so that the storage was no longer charged against DB2.

You will see the DSNV516I message followed by this message:

DSNVMON – BEGINNING STORAGE CONTRACTION MODE

There are two activities going on within DB2 here that we need to explain. One is that after thread deallocation or a certain number of commits, DB2 will “contract” the thread storage and give what is not needed back into the 64-bit pools so that other DB2 threads could use it (PM86952 helps in this case). DISCARD MODE gives the storage back to MVS as a free frame for use by someone else and is generally avoided until significant paging activity is seen on the LPAR.

Note: Not turning the real storage frames over to MVS can improve DB2 system performance due to a significant reduction in first reference page faults, RSM lock contention, and uncaptured CPU time; however, this causes the DB2 subsystem's storage footprint to grow.

PM99575, currently open, is meant to change the DISCARDATA logic for real frames based on the customer settings for REALSTORAGE_MANAGEMENT to a more uniform and predictable function based on allowing freed real frames to return to the 64-bit stack pools constantly, agent shared, private and other shared pools on a timer, but use KEEPREAL(YES). The KEEPREAL setting means the frame is put back in the pool for DB2 to use, but is also stealable by MVS if it is needed, thus avoiding the first reference page faults.

- REALSTORAGE_MAX = roughly 2x peak DB2 working set size (no default):

Amount of REAL and AUX in GB a given DB2 subsystem is allowed to consume. If this value is hit, DB2 will terminate. The trade off is potential LPAR loss (Note: default is 0 which means NO LIMIT). This setting is to stop 1 run-away DB2 from affecting other address spaces on the LPAR in the event of a storage leak or an uncontrolled influx of threads. If this value is approached a “sticky” DSNS003I is written to the console. Upon relief, a DSNS004I message is written and outstanding DSNS003I messages are DOM'ed:

DSNVMON – SUBSYSTEM WARNING FOR REAL STORAGE HAS BEEN DETECTED

This number should be carefully addressed at 2x the normal working set size, or 1.5x the working set if the buffer pools constitute the vast majority of the DB2 storage
Example 13-2 on page 187.

► MVS settings with suggestions:

- MAXSPACE= 16000 MB minimum (default is 8000 MB):

This PARMLIB setting for dump options specifies the amount of space that the DUMPSERV address space can occupy during a DUMP capture.

- AUXMGMT= ON (default=ON):

New SVC dumps are not allowed when auxiliary storage usage reaches 50%, and an SVC dump that is in the process of capturing data is truncated, resulting in a partial dump, when auxiliary storage usage reaches 68%. Once either limit is reached, data capture for new dump requests is not allowed again until auxiliary storage usage drops below 35%.

- MAXSNDSP= <10 seconds (default is 15):

This specifies the maximum amount of time an SVC dump is allowed to keep the system non-dispatchable. If the system is reset to be dispatchable because the system has been kept non-dispatchable longer than a certain number of seconds, a SNAPTRC is issued.

► DFSORT parameters affecting storage use (II13495):

- EXPOLD =% of storage allowed to be pushed to AUX:

To avoid impacting other work running on the system (such as DB2), this number should be 0 in the most desirable situation, because at the point it goes to AUX, it must have already pushed other address spaces out to AUX. Of course setting storage critical in the WLM definition for DB2 will avoid DB2 being pushed to AUX.

- EXPRES=% of storage to preserve:

In the case where MAXSPACE = 16GB with DB2 10, you may want to protect at least 16 GB worth of real storage if not more. This is simply a defensive measure telling MVS what % or actual amount of storage to hold in reserve in case of a DUMP or other event.

- EXPMAX=% of storage for memory object and hyperspace sorting, somewhat depends on EXPOLD and EXPRES:

How much storage can you spare for DFSORT operations? The previous 2 settings are defensive in nature by allowing DFSORT to take everything it can, except for what you specify in EXPOLD and EXPRES. EXPMAX, however, is a prescriptive number which will not allow DFSORT to take more than that % or hard number of storage. So this would need to be monitored on a regular basis to compare the trade-off of restricting storage given to DFSORT, with the performance impact on DFSORT.

Figure 13-2 shows the appearance of AUX storage on 08/01 (red line) when the available storage on the LPAR went to zero. EXPMAX could have been used as a cap to limit the sort job that was absorbing the last of the real storage on the box. As a general statement this customer was running very low on available real storage, and EXPMAX would have needed to be set at about 5GB to keep the sorting jobs from causing storage to be paged out to AUX, but it is a good demonstration of where the setting could have helped other workload remain in real memory.

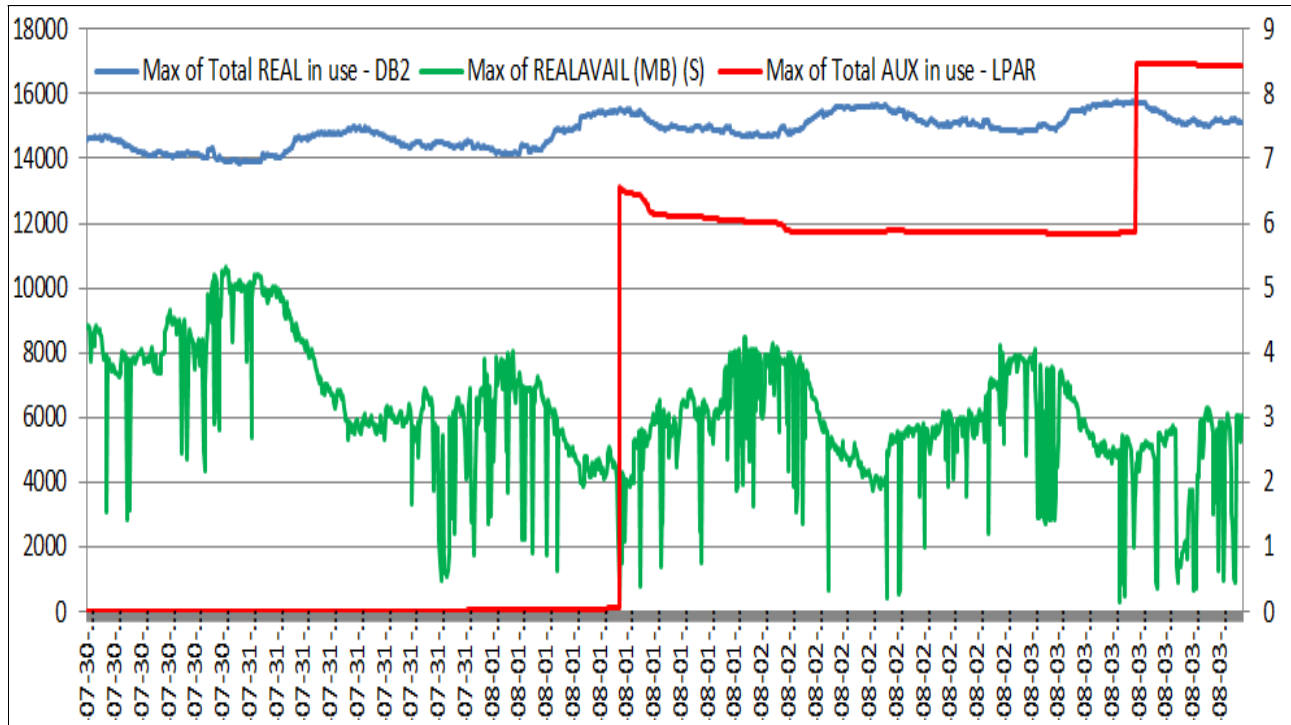


Figure 13-2 Customer real storage, axis in MB

Attention: We address the DFSORT parameters here, but there are similar settings in other vendor products that would have the same effect and need to be monitored.

Important: Provision enough real storage on the box/LPAR and ensure it is configured properly. There should be a buffer of real storage to accommodate sudden influxes of work as well as large sort jobs, and possible address space dumps. Plan upgrades and other activities that could increase the amount of storage used around maintenance windows that could allow you to provision more storage as well if needed. Even if you are certain you will not need it, order more memory on the box than you will provision, just in case.

Remember that trading real storage to buy back CPU cycles is a very cost effective trade-off.

In order to graph these series of data points refer to the IFCID fields listed after Figure 13-1 on page 182.

13.2.2 64-bit storage

Example 13-1 has the field names from OMEGAMON PE, the performance database column name, and the report label, to give you an idea of how to calculate the total storage used by the DB2 subsystem. The buffer pool column is important because anytime we talk about estimating storage increase or savings between releases or maintenance levels, the percentage given varies based on the size of the buffer pools.

So if SVPOOL1 is a high percentage of QW0225RL for DBM1, then the increase in storage between releases would have been on the lower side of the estimated 10-30% increase moving to DB2 10. It would be on the low end if you were coming from DB2 9 and had 10's of gigabytes of buffer pools allocated. The increase would be on the high side if you skipped from V8 and had only 1-2 gigabytes of buffer pool space because then the below the bar storage would constitute a larger percentage of the overall picture. The really helpful fields arrived with DB2 10 as we did not report on 64-bit SHARED private in DB2 9 and prior. Here is the example of the fields that sum up to show the total REAL storage used by DB2.

Example 13-1 List of IFCID 225 fields

Real storage total=

<u>field</u>	<u>OMPE perf. DB col.</u>	<u>Description of field</u>
QW0225RL	REAL_STORAGE_FRAME	ASID DBM1 REAL 4K frames for 31 and 64-bit priv +
QW0225RL	DIST_REAL_FRAME	ASID DIST REAL 4K frames for 31 and 64-bit priv +
SW225SSR	A2GB_SHR_REALF_TS	REAL 4K frames for 64-bit shared +
SW225KSR	A2GB_SHR_REALF_STK	REAL 4K frames for 64-bit shared stack +
SW225CSR	A2GB_COMMON_REALF	REAL 4K frames for 64-bit common

The buffer pools are included in the DBM1 64-bit private frames (QW0225RL) but we report on them separately as well if you want to break them out.

SVPOOL1 VIRTUAL_BUFFERPOOL Virtual buffer pools - Allocated

OMEGAMON PE offers a summary at the end of the storage information which covers private, shared, and common storage for the subsystem. Keep in mind that the AUX may not be accurate due to double accounting. This means that z/OS will continue to charge those AUX frames against the address space that spilled into them even if the address space is not currently using them. The AUX storage will remain "charged" against the last DB2 subsystem to use the frame until DB2 is bounced, or some other address space uses that page data set.

The report in Example 13-2 represents what Figure 13-1 on page 182 has depicted for total real storage in use by DB2 and can be found in the OMEGAMON PE statistics report.

Example 13-2 Real storage in use

REAL STORAGE IN USE - SUMMARY

-----	-----	-----
31/64-BIT PRIVATE (DBM1)	(MB)	3841.24
31/64-BIT PRIVATE (DIST)	(MB)	38.37
64-BIT SHARED THREAD AND SYSTEM	(MB)	1082.89
64-BIT SHARED STACK	(MB)	225.61
64-BIT COMMON	(MB)	8.27
TOTAL REAL STORAGE IN USE	(MB)	5196.38

Figure 13-3 depicts the breakdown of what pieces of DB2 storage live where. Starting at the top, 64-bit common is made up of distributed agents and package accounting and roll-up. Typically the use is in the range of 10's of megabytes per DB2 subsystem, but the subsystem will not come up if there is less than 6GB of Common defined per subsystem on the LPAR.

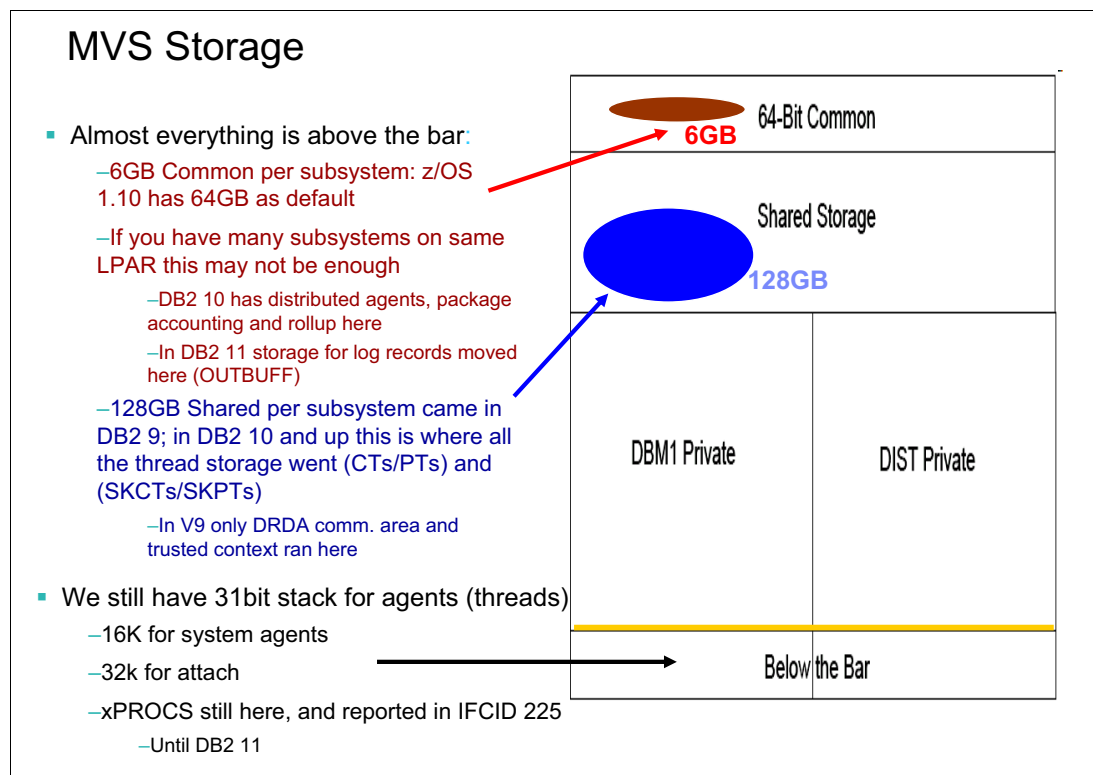


Figure 13-3 64-bit virtual

The 64-bit shared area is used to execute SQL; holds the dynamic statement caches; houses the cursor table/package table entries CT/PT and SKCT/SKPT. In DB2 9 the shared storage was required but only used by TCP/IP and trusted context. The 64-bit Stack storage in DB2 10 is much like the stack in DB2 9, which was used for active threads. The parameter REALSTORAGE_MANAGEMENT ZPARM now acts as the gate keeper to free up 64-bit storage from DB2 and give it back to MVS. Depending on what mode (AUTO or ON) REALSTORAGE_MANAGEMENT is in, it will contract the agent level shared pools, stack storage, and specific shared and private pools.

13.2.3 LFAREA

The large frame area (LFAREA) is used for the fixed 1 MB large pages and fixed 2 GB large pages. Using large pages can improve performance for some applications by reducing the overhead of dynamic address translation.

Do not overallocate LFAREA for 1 MB frames:

If the LPAR runs short on 4 KB frames, it will have to decompose the available 1 MB frames down into 4 KB frames. So not only are you missing out on the CPU savings, but you are burning extra cycles during decomposition. Then when the frames are freed up they are coalesced back into 1 MB frames. DB2 and JAVA (based on heap size) are currently the only users of 1 MB frame support:

<http://www-01.ibm.com/support/docview.wss?uid=isg10A34024>

However, the more advantage you can take of the large frames, the more possibility there is for performance improvement based on improving the translation lookaside buffer hit ratio.

Take the following settings into account when sizing LFAREA:

- ▶ How much PGFIX(YES) buffer pool storage there is
- ▶ Size of OUTBUFF ZPARM (only in DB2 11)

13.2.4 New in DB2 11

In DB2 11, some storage was shifted for performance reasons and the reporting has improved yet again.

DB2 11 introduces the ability to track IRLM virtual and real storage as well. This information was added to the IFCID 225 record and can be reported against via OMEGAMON PE. See Example 13-3.

Example 13-3 IRLM storage use

IRLM STORAGE BELOW AND ABOVE 2 GB		QUANTITY
-----	-----	-----
EXTENDED CSA SIZE IN USE	(MB)	5.15
HWM EXTENDED CSA SIZE IN USE	(MB)	10.00
31 BIT PRIVATE IN USE	(MB)	0.00
HWM 31 BIT PRIVATE IN USE	(MB)	0.00
THRESHOLD 31 BIT PRIVATE	(MB)	0.00
64 BIT PRIVATE IN USE	(MB)	0.00
HWM 64 BIT PRIVATE IN USE	(MB)	0.00
THRESHOLD 64 BIT PRIVATE	(MB)	0.00
64 BIT COMMON IN USE	(MB)	0.00
HWM 64 BIT COMMON IN USE	(MB)	0.00

As you can see, the storage footprint here is quite small and due to 31-bit addressing. As opposed to the DBM1 storage in DB2 10 going above the bar, the storage here will remain fairly consistent. This should be in the order of 10's of megabytes of real storage. This storage much like that of the buffer pools (considering they are not fluctuating) is a "given" when measuring storage usage or predicting increases, but it may become more important in a future release.

The remaining 31-bit storage related to threads is moved above the bar. The xPROC, or fast column processing procedures, were moved above the bar in DB2 11, and are now referred to as zPROC. These provide a shortened code path when dealing with queries accessing multiple columns. Moving them above the bar may save about 10% (10-20MB with around 300 threads) of the storage seen in DB2 10 31-bit storage. The storage is reported in IFCID 225 and there are separate storage areas for dynamic versus static SQL. See Example 13-4.

Example 13-4 xPROC storage

<u>IFCID</u>	<u>Perf. Database</u>	<u>Report label</u>
QW0225LS8	IN_USE_LOCAL_CACHE	xPROC for dynamic SQL - In use
QISEKSPA8	XPROC_STORAGE	xPROC for static SQL - In use

Finally, the storage area for OUTBUFF is moved from MSTR private storage to Common 64-bit storage. It has been page fixed since DB2 10, but was not eligible for 1 MB frames until DB2 11. So be sure to include it in the LFAREA calculation where you determine the sizing for the 1 MB frames you need in DB2 11 Conversion Mode. The 64-bit Common storage seen in DB2 10 was normally in the range of 10-20MB due to its limited use. It will obviously grow in DB2 11 as log buffer storage is moved over, but it should also help with logging performance as we avoid the cross memory calls between the MSTR and DBM1 address spaces. There is more on LFAREA discussed in “Real storage” on page 140.

Flash Express

Flash Express is a new technology designed to extend the central storage on the zEC12s (and zBC12s). With the combination of Flash Express installed on a zEC12 and the pageable 1 MB large page support in z/OS V1R13, DB2 11 can exploit the large page support by allocating internal control blocks (PMBs) using 1 MB pageable storage. This function has been retro-fitted to DB2 10 with APAR PM85944. Some of the DB2 code itself can also be backed by 1 MB pageable frames if Flash is installed. Although you want enough *real storage* to back these components, these large pages may be paged out to and from Flash Express. It is much faster and more efficient to access this storage in central storage, so you should not depend on pageable large frames.

With Flash Express, DB2 11 can gain the performance improvement from a reduction in translation lookaside buffer (TLB) misses and an increase in the TLB hit rate. Flash memory can also be used to improve SVC and Standalone dump data capture time, if you cannot back the dump with real storage. Internal test have shown the dump capture time was reduced by 25% and the system was able to resume steady state 5x faster than if the dump had gone to AUX storage. It also removes the requirement for PLPA and Common page data sets when used for cold start IPLs. This is discussed in the *DB2 11 for z/OS Overview*, SG24-8180. The reason to acquire Flash Express storage is tied to 3 factors:

- How much of the MAXSPACE in PARMLIB you cannot guarantee in central storage, which will then be forced to AUX storage.

This is the only real sizing requirement with Flash, as the other items in this list are more DB2 performance than a large user of storage.

- In DB2 11 and DB2 10 (with PM85944), the PMB storage used for buffer pool control blocks can be backed by 1 MB frames even if the buffer pool is not page fixed. You can estimate their storage footprint with the following equation:

$$((VPSIZE * pagesize / 1024 / 1024) + 6399) / 6400 = \# \text{ of MBs of storage used for the PMB}$$

- The DB2 11 31-bit Extended low-private storage usage:

The 31-bit low private storage reported in the statistics report field QW0225EL can take advantage of being backed by 1 MB pageable frames, which of course helps with the TLB hit rate. Storage here would be consumed by DB2 code and pageset storage (see Example 13-5).

Example 13-5 Extended low private storage

DBM1 AND MVS STORAGE BELOW 2 GB	CONTINUED	QUANTITY
24 BIT LOW PRIVATE	(MB)	0.21
24 BIT HIGH PRIVATE	(MB)	0.48
24 BIT PRIVATE CURRENT HIGH ADDRESS		000000000003D000
31 BIT EXTENDED LOW PRIVATE	(MB)	75.14
31 BIT EXTENDED HIGH PRIVATE	(MB)	47.62

13.3 Monitoring virtual storage usage

Virtual storage is not quite a thing of the past. Depending on which version you are on, and other products you have running on the LPAR, items such as ECSA may still require monitoring attention. Having many IMS and CICS regions on the same LPAR may necessitate the adjustment of ECSA/ESQA from time to time, and these can affect DB2 as well. If you are still on DB2 9 or have just migrated to DB2 10 or DB2 11 and have not rebound packages from DB2 9 yet, then you will still need to monitor and configure for 31-bit virtual storage constraint. See Figure 13-4 for a visual breakdown of virtual storage.

That is the reason the EDMPOOL ZPARM is still around even in DB2 11. Even after rebinding in DB2 10 or DB2 11 many customers are still in the 1,500-2,500 range for maximum number of threads based on virtual storage. Items such as RELEASE(DEALLOCATE) and thread parallelism can reduce this maximum number as well. The Getmained storage represents the vast majority of storage below the 2 GB bar and is allocated from the top down. The DB2 code, and items such as data set control blocks get allocated from the bottom up.

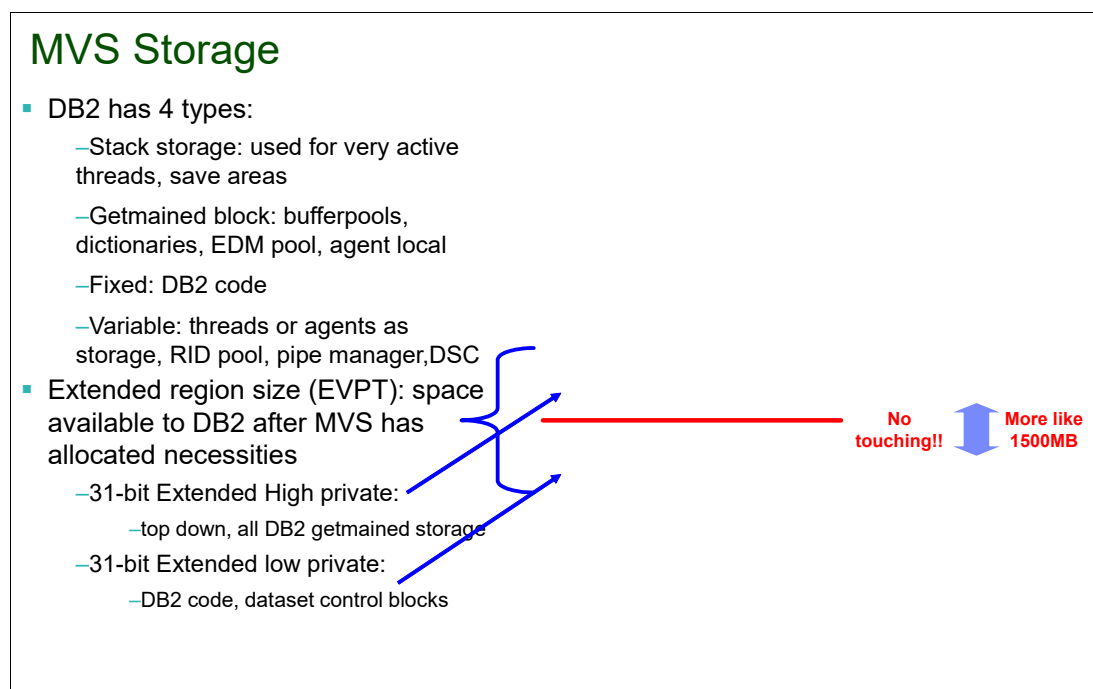


Figure 13-4 Virtual storage visual breakdown

If these storage allocations meet in the middle, then DB2 will enter storage contraction, and if storage cannot be freed off for must complete or abort processing, then the DB2 subsystem will come down. On rare occasions DB2 can bring the LPAR down with as well, depending on the amount of ECSA storage it has consumed. This was one of the primary reasons the MEMU2 REXX exec was created and distributed to customers.

The MEMU2 REXX exec, when it is run, will wake up every 5 minutes by default and snap the IFCID 225 data through IFI reads. With a more granular interval of 1 minute you can see thread count, and storage spikes thus giving you the ability to more accurately estimate the number of threads the system can handle. By running the exec from the time DB2 is started until it is bounced you can trend the storage and watch for the overall growth to slow when it reaches a steady state. The calculation for the 31-bit virtual storage limited number of threads is the same across multiple DB2 versions. As these thread footprints grow when they move above the bar though, real storage used by the subsystem based on a given number of threads can become important as well.

Tip: Granular monitoring of virtual storage (and threads) is very important - just remember if you extract your own IFCID 225 information you must remember to normalize the 4 KB frames/slots to 4,096 bytes at some point and then calculate the megabytes in use.

MEMU2 for DB2 9 or DB2 10 can be downloaded from the following site:

<https://www.ibm.com/developerworks/community/files/app?lang=en#/person/270000K6H5/file/e2736ed5-0c73-4c59-b291-9da08255b941>

The calculations for thread footprint and estimated number of supported threads remains the same based on virtual storage:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS3431>

The new OMEGAMON PE Spreadsheet input Data Generator can be used for this as well, and it has a function within it to allow the user to specify the units in frames or MB and minutes or seconds as they choose:

<http://www.ibm.com/support/docview.wss?uid=swg1PM73732>

<http://www.ibm.com/support/docview.wss?uid=swg27036083>

In an effort to simplify some of this, there is a list of the IFCIDs and the corresponding OMEGAMON PE performance database columns can be found in the appendix. Extracting these fields with the OMEGAMON PE Spreadsheet Generator will yield the same results as running MEMU2.

See Table B-4 on page 313 for details.

Figure 13-5 shows an example of an OMEGAMON PE statistics report that breaks down the 31-bit virtual storage. We can see the four main types of storage being GETMAINED, Variable, Fixed, and Stack. There is no need to go into detail on each of these right now, but do take note that when looking to establish a base line, or project for a storage increase, you need to ensure that these numbers are in a steady state.

For example, prior to DB2 10, the system agents such as prefetch and deferred write, could take over 100 MB of storage below the 2 GB bar. If the system had not been put under a substantial load, yet when you captured the IFCID 225 data to estimate how many threads the system could handle, then you could be overly aggressive in your estimates. Your estimate should include storage for all the system agents to be active (also seen in IFCID 225). Similarly, the more user threads that are present in the system during the statistics capture, the more accurate the estimate of the maximum threads and real storage usage will be.

DB2 DBM1 Virtual Storage

- 31-bit / 24-bit DB2 storage

- Getmained
- Variable
- Fixed Storage
- Stack storage

DBM1 AND MVS STORAGE BELOW 2 GB		QUANTITY
-----		-----
TOTAL DBM1 STORAGE BELOW 2 GB	(MB)	62.44
TOTAL GETMAINED STORAGE	(MB)	10.48
TOTAL VARIABLE STORAGE	(MB)	26.63
TOTAL AGENT LOCAL STORAGE	(MB)	13.56
TOTAL AGENT SYSTEM STORAGE	(MB)	11.99
NUMBER OF PREFETCH ENGINES		600.00
NUMBER OF DEFERRED WRITE ENGINES		300.00
NUMBER OF CASTOUT ENGINES		65.00
NUMBER OF GBP WRITE ENGINES		30.00
NUMBER OF P-LOCK/NOTIFY EXIT ENGINES		20.00
TOTAL AGENT NON-SYSTEM STORAGE	(MB)	1.57
TOTAL NUMBER OF ACTIVE USER THREADS		40.53
NUMBER OF ALLIED THREADS		2.73
NUMBER OF ACTIVE DBATS		2.58
NUMBER OF POOLED DBATS		35.22
NUMBER OF PARALLEL CHILD THREADS		7.17
THREAD COPIES OF CACHED SQL STMTS	(MB)	5.27
IN USE STORAGE	(MB)	0.01
HWM FOR ALLOCATED STATEMENTS	(MB)	0.06
THREAD COPIES OF STATIC SQL	(MB)	4.26
IN USE STORAGE	(MB)	0.01
THREAD PLAN AND PACKAGE STORAGE	(MB)	0.00
BUFFER MANAGER STORAGE CNTL BLKS	(MB)	1.75
TOTAL FIXED STORAGE	(MB)	0.40
TOTAL GETMAINED STACK STORAGE	(MB)	24.92
TOTAL STACK STORAGE IN USE	(MB)	21.12
SYSTEM AGENT STACK STORAGE IN USE	(MB)	19.65
STORAGE CUSHION	(MB)	306.09

Figure 13-5 DBM1 virtual storage

The DSNVMON message listed in Example 13-6 could show up if you do not defensively set the DSNZPARMs governing virtual storage in DB2, i.e. CTHREAD, MAXDBAT, EDMPOOL. These should be monitored and configured to still allow roughly 100 MB of extra cushion beyond the cushion that DB2 already sets aside on behalf of z/OS (max data sets and threads). Hopefully your storage cushion, that is, MAXDBAT and CTHREAD, are set realistically so that this message will come out just before entering the storage cushion. If the number is over-inflated (such as MAXDBAT+CTHREAD=2,000 prior to DB2 10), then you will be triggering full system contraction much sooner than entering the cushion, thus artificially limiting the number of threads the subsystem can handle.

Example 13-6 DSNVMON message

```

DSNV508I -SE20 DSNVMON - DB2 DBM1 BELOW-THE-BAR
STORAGE NOTIFICATION
  77% CONSUMED
  76% CONSUMED BY DB2
  352M AVAILABLE OUT OF REGION SIZE 1553M
  WITH A 274M STORAGE CUSHION

```

This means that you need to determine the average thread footprint during the time of peak activity, determine the number of threads the system can handle before hitting the cushion, then back the number of threads off to allow another 100MB below the cushion for safety. MEMU2 can help you determine this footprint.

OMEGAMON PE has some rough estimates based on both the 31-bit and 64-bit thread footprint. These fields are SW0225TF(31-bit) S225DTFR (31 and 64-bit), so the former is more aligned with the virtual storage limited number of total threads, and the latter shows approximately what all the working storage, divided by the number of threads yields.

The 64-bit footprint looks very large usually because of how DB2 10 allocates the storage pools above the bar. It is important to try and capture a statistics trace interval (not a report from 24 hours of data) where the peak number of threads entered the system and hence the thread footprint has leaned out due to most of the allocated storage being used. In particular a change in the size of the 64-bit shared stack storage would show that the subsystem had to react to the storage requests from incoming threads. However, the best approach for DB2 10 and up is to monitor the overall DB2 real storage usage and correlate growth to thread increase/decrease in order to get a working model for how large your thread limit could be based on a real storage budget as shown in Figure 13-2 on page 186.

You can get a snapshot of the amount of storage available to threads as well as the cushion at any point by using the **DISPLAY THREAD** command. See Example 13-7.

Example 13-7 -DIS THREAD() TYPE (SYSTEM)*

NAME	ST A	REQ ID	AUTHID	PLAN	ASID	TOKEN
VA1A	N *	0 002.VMON 01	SYSOPR		002A	0
V507-ACTIVE MONITOR, INTERVALS=8216, STG=77%, BOOSTS=0, HEALTH=100						
REGION=1553M, AVAIL=352M, CUSHION=274M						

If you do not heed these messages, and we suggest that automation be put in place to look for it, then you could easily encroach on the DB2 storage cushion. There are basically 4 elements control the size of the cushion. They included your settings for DSMAX, CTHREAD, MAXDBAT and the Extended Region Size.

Figure 13-6 shows the calculation for the storage cushion and the consequences for entering each layer of protection within DB2. Hitting full system contraction will stop any current thread from being allocated any more storage, as well as block new threads from coming into the system, and you will see a burn in DBM1 TCB time in the statistics as well.

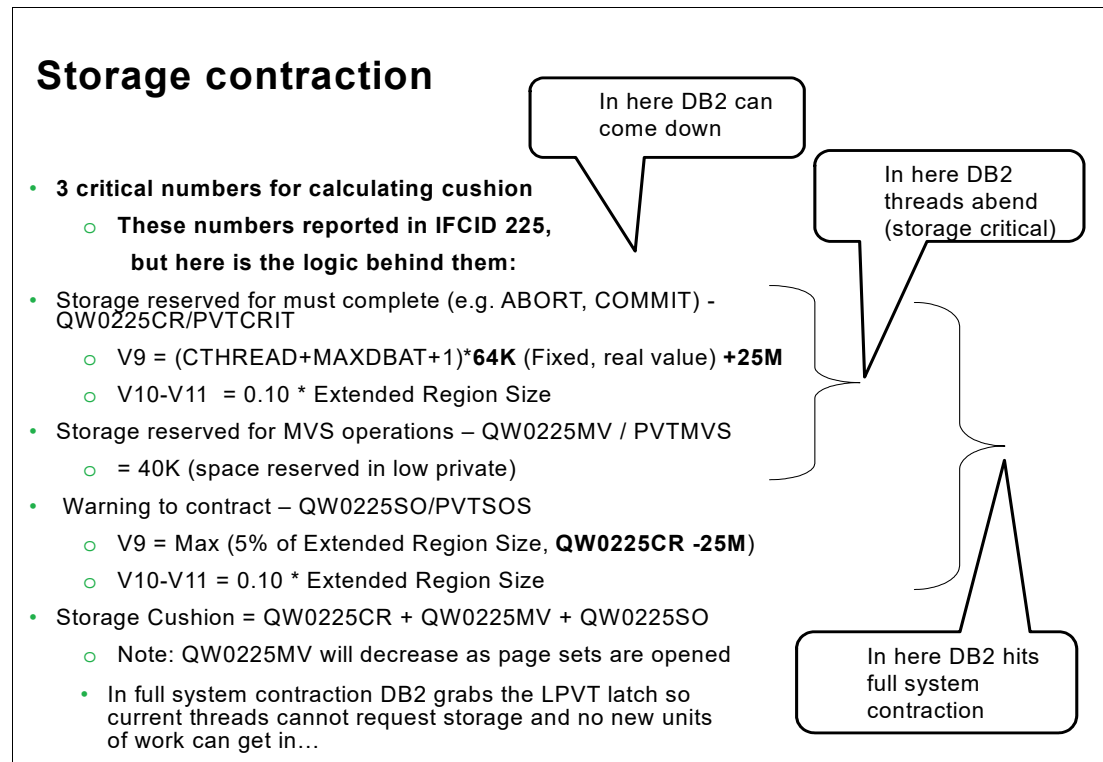


Figure 13-6 Storage contraction

So when you are looking to set up CTHREAD and MAXDBAT, you should graph the EPVT - Storage cushion = how much room you have left for 31-bit DBM1 storage (which includes thread storage). Figure 13-7 shows an example of the thread footprint calculation, which is also referred to in the TIP section on page 192.

Thread footprint calculations

Estimating Maximum Number of Threads...

- **Remember to use the MAX impact value across all available data, e.g. MAX system storage**
- **'Basic' storage cushion (BC)**
 - $(BC) = QW0225CR + QW0225MV + QW0225SO$
- **Calculate Max non-DB2 storage (ND)**
 - $(ND) = \text{MAX}(\text{MVS 31 BIT EXTENDED HIGH PRIVATE } QW0225EH - \text{TOTAL GETMAINED STORAGE } QW0225GM - \text{TOTAL GETMAINED STACK STORAGE } QW0225GS - \text{TOTAL FIXED STORAGE } QW0225FX - \text{TOTAL VARIABLE STORAGE } QW0225VR)$
- **Max. allowable storage (AS)**
 - $(AS) = QW0225RG - (BC) - (ND)$
- **Max. allowable storage for thread use (TS)**
 - $(TS) = (AS) - \text{MAX}(\text{TOTAL AGENT SYSTEM STORAGE } QW0225AS - \text{TOTAL FIXED STORAGE } QW0225FX - \text{TOTAL GETMAINED STORAGE } QW0225GM - \text{MVS 31 BIT EXTENDED LOW PRIVATE } QW0225EL)$
- **Average thread footprint (TF)**
 - $(TF) = (\text{TOTAL VARIABLE STORAGE } QW0225VR - \text{MAX}(\text{TOTAL AGENT SYSTEM STORAGE } QW0225AS) + \text{TOTAL GETMAINED STACK STORAGE } QW0225GS) / (\text{Allied threads } QW0225AT + \text{DBATs } QDSTCNAT)$
- **Max threads supported = $(TS) / (TF)$**

Figure 13-7 Thread footprint calculation

Another issue you can run into is what is NOT reported in OMEGAMON PE. The Max non-DB2 storage shown in Figure 13-7 is basically determining the total DBM1 31-bit storage and subtracting it from the high private, so another way to look at it is:

Tip: Calculation used to determine non-DB2 storage in the DBM1 address space:

TOTAL DBM1 STORAGE = TOTAL GETMAINED STORAGE QW0225GM + TOTAL GETMAINED STACK STORAGE QW0225GS + TOTAL FIXED STORAGE QW0225FX + TOTAL VARIABLE STORAGE QW0225VR

NON-DB2 STORAGE = MVS 31 BIT EXTENDED HIGH PRIVATE QW0225EH - TOTAL DB2 DBM1 STORAGE

This non-DB2 storage will certainly grow after DB2 is bounced and normally peaks out just over 100MB after several days. It is associated with MVS functions such as SMF. One PARMLIB option that can cause this number to continue to grow is the DETAIL parameter in SMFPRMxx. Specify NODETAIL and INTERVAL to exclude the EXCP sections from SMF type 30 subtype 4 and subtype 5 records collected for started tasks. Otherwise, the EXCP sections for SMF type 30 subtype 4 and subtype 5 records will be included.

For long running tasks, excluding EXCP (execute channel program for I/O) sections from SMF type 30 subtype 4 and subtype 5 records can greatly reduce these amounts:

- ▶ The amount of storage required to hold SMF records in memory
- ▶ The amount of DASD space required to write the records out to data sets
- ▶ The amount of CPU used at step end and job end

By graphing this storage along with the DBM1 usage, you can detect storage creeps, which could also be leaks below the bar caused by DB2, monitoring tools, or other products.

Obviously the non-DB2 storage is taking up precious room that DB2 threads could be using to execute in. Figure 13-8 illustrates how as non-DB2 storage increases the estimates for the maximum number of threads supported decreases.

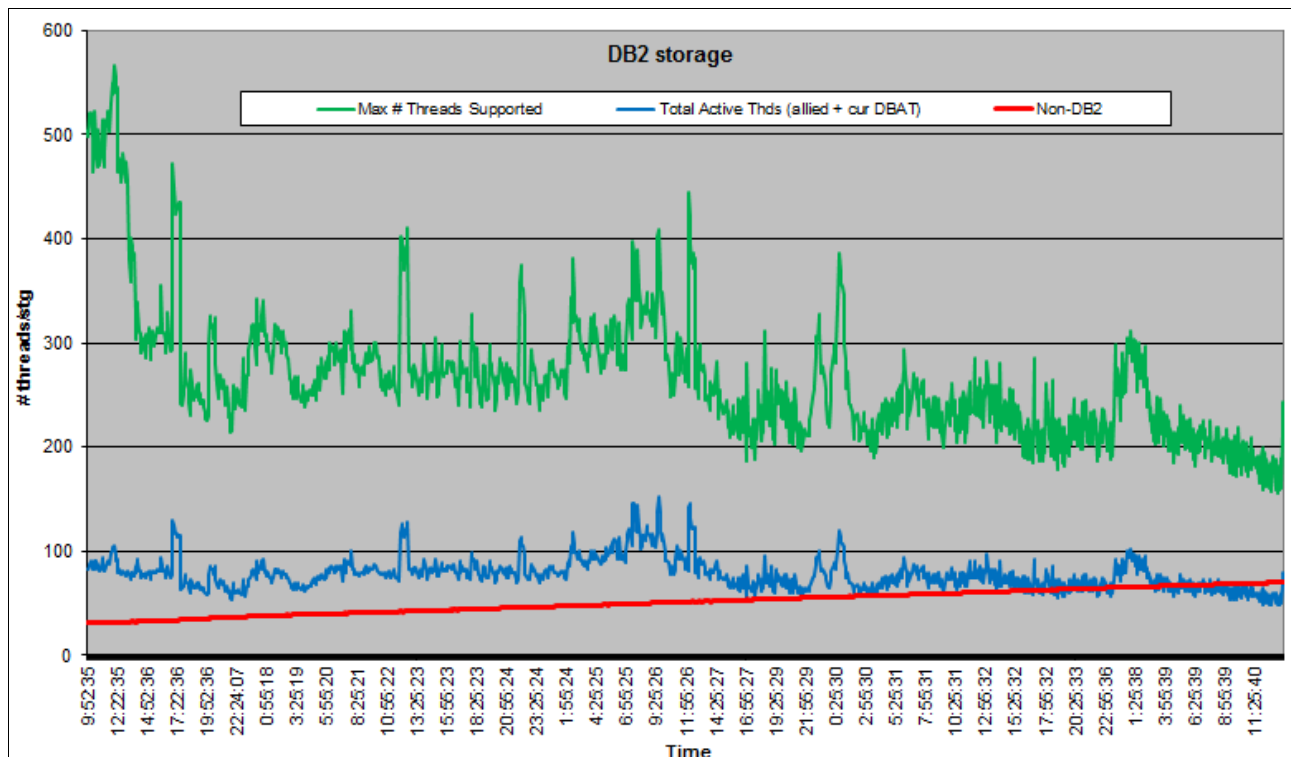


Figure 13-8 Storage creep in DBM1 address space



Part 3

Transaction monitoring

Depending on the transaction manager, you can use IMS or CICS monitoring facilities to determine throughput, in terms of transactions processed, and overall whether you need transaction response times.

At a certain point of your analysis, especially for distributed or local types of transactions, you will want to understand more about where DB2 spends its transit time and which actions can reduce it.

DB2 accounting provides plan and package level reporting. The data is collected by the DB2 accounting traces, which are the source of performance indicators in DB2 applications and the first step in application performance monitoring and tuning.

In this part of the book, we describe the information and the actions for transaction monitoring.

This part contains the following chapters:

- ▶ Chapter 14, “Accounting trace overview” on page 201
- ▶ Chapter 15, “Analyzing accounting data: CPU and suspension time” on page 217
- ▶ Chapter 16, “I/O suspensions” on page 231
- ▶ Chapter 17, “Locking, latching, and buffer pool accounting counters” on page 243
- ▶ Chapter 18, “Service task suspension” on page 257
- ▶ Chapter 19, “Stored procedures, user defined functions, and triggers” on page 265
- ▶ Chapter 20, “DRDA, parallelism, and statement cache” on page 277



Accounting trace overview

The best way to troubleshoot a poorly performing application is to understand where the time is actually spent and what DB2 resources are being used. DB2 traces provide these indicators, but you need to know how to interpret them and what the most efficient corrective actions are. If most of the time is spent using system CPU, what is the most likely culprit? If the problem is elapsed time, there are many types of suspensions and many of them have specific tuning steps to reduce the wait times.

This chapter discusses the following topics, including some rules of thumb of interest for the application performance troubleshooters:

- ▶ Response time factors:

- The complex layers of transaction execution that combine to make up the total elapsed and CPU times

- ▶ Response time scope:

- DB2 reporting capability of the application layers

- ▶ DB2 accounting data:

- The five cases that cause an accounting record to be written to help you evaluate the scope of each accounting record

14.1 Response time factors

Today's applications are often complex and involve numerous layers such as presentation, application logic, and database processing. These layers are typically implemented at different hardware and software tiers and include network as the major architectural element. These topologies make the task of performance monitoring and tuning very complex and challenging. Figure 14-1 shows an example of general transaction flow.

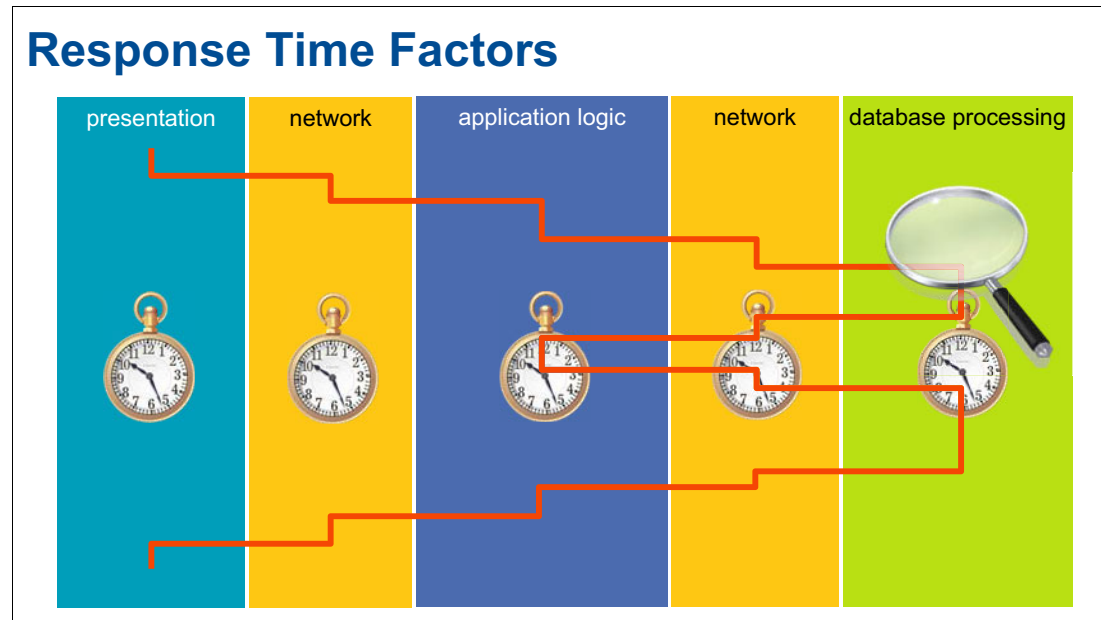


Figure 14-1 Response time factors

Depending on application architecture, some additional layers exist: for example, CICS or IMS applications often run at the same tier as the database.

This chapter focusses on performance inside the database engine. It discusses:

- ▶ Where to find the appropriate indicators
- ▶ Reasons for a particular time distribution
- ▶ Typical corrective actions

Note that database processing does not necessarily include only database access. It increasingly involves complexities such as these:

- ▶ Parts of application logic that is encapsulated in triggers, user-defined functions, and stored procedures.
- ▶ Time spent processing in communications and processing at different sites with distributed transactions.
- ▶ Factors involving query parallelism, where query execution is not limited to a single thread, but to a number of parallel tasks, all of which have their own time distribution.

14.2 Response time scope

DB2 and its monitoring tools provide timers at the plan, package, and statement level as shown in Figure 14-2.

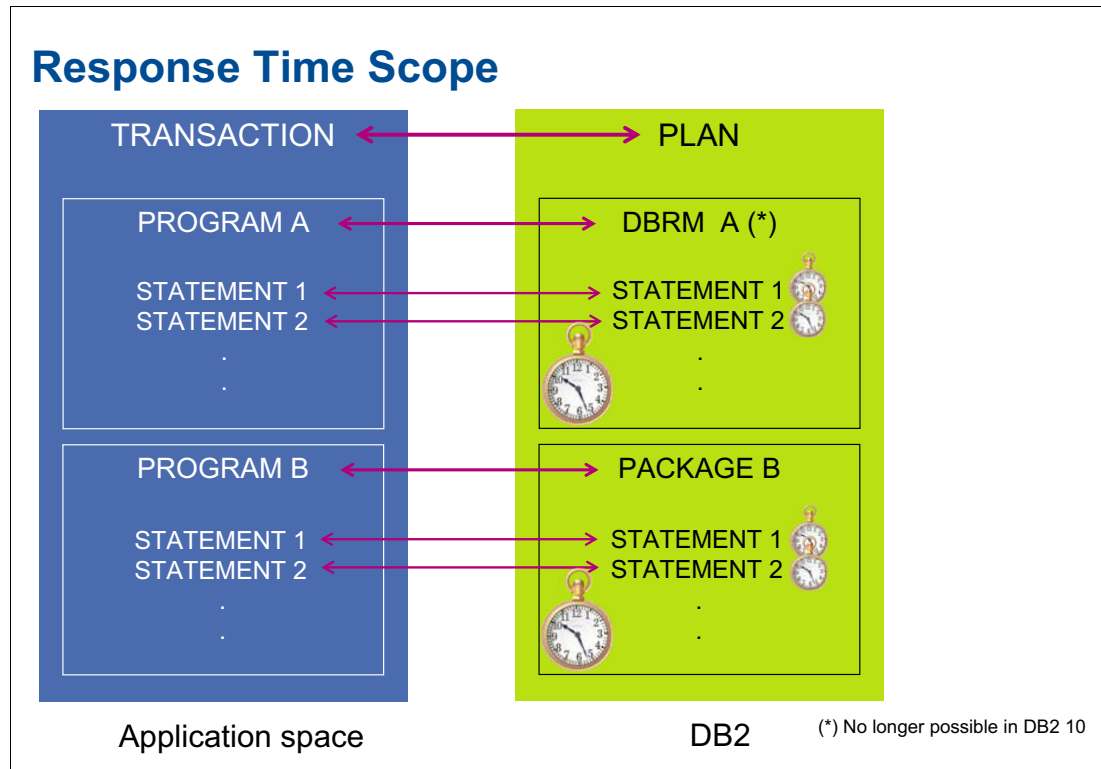


Figure 14-2 Response Time Scope

DB2 accounting traces collect relevant data at the plan and package level, and are the source of the most important performance indicators in DB2 applications. They provide a wealth of performance timers and counters that in most cases directly indicate the source and cause of the problem. Monitoring tools summarize and externalize accounting data on per-transaction, per-program basis, which allows a top-down response time analysis.

For dynamic SQL, bottom-up analysis is sometimes more appropriate. Dynamic SQL often requires SQL statement level timers, and DB2 provides statement-level accounting information from the dynamic statement cache via its instrumentation facility. Section 20.4, "Capturing performance data with the statement cache" on page 284 explains how to use the dynamic statement cache to get tuning data at the statement level.

DB2 does not know how much time was spent in the application before the first access to DB2, so DB2's time measurements begin with the first access to DB2, which is at thread allocation. After that, DB2 can collect data during the entire life of the thread, which includes time spent in the presentation layer, application logic, network and DB2 processing.

All of this is recorded in DB2 accounting class 1 elapsed time. See Figure 14-3.

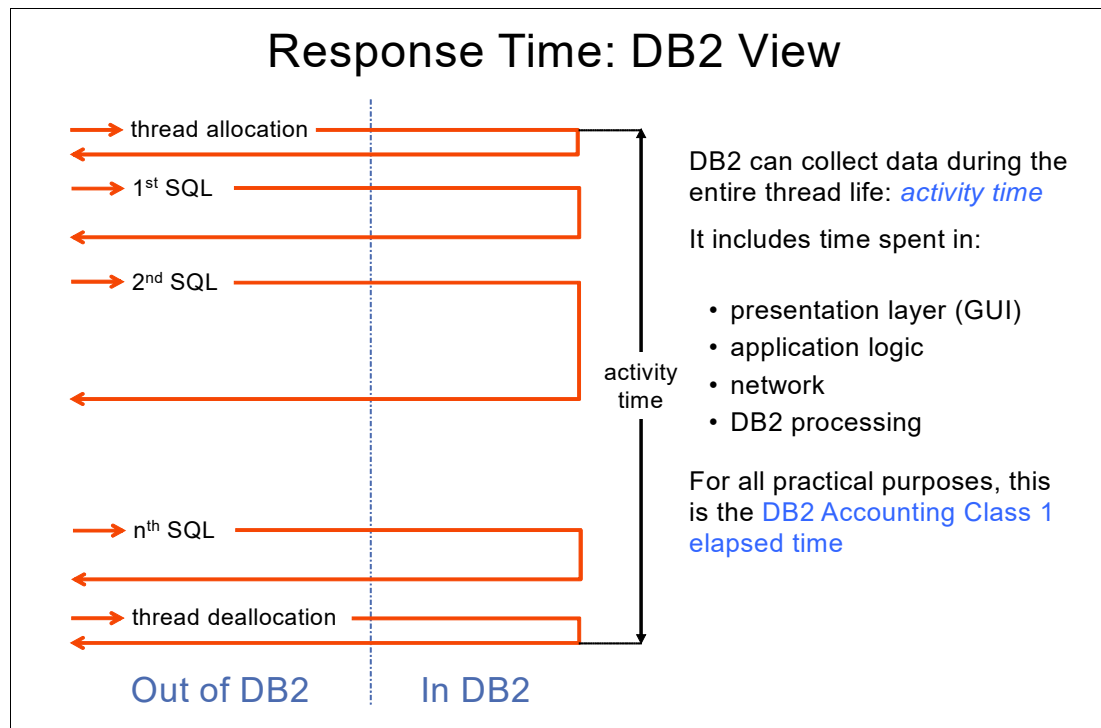


Figure 14-3 The DB2 view

In order to get plan or package level data, the DB2 accounting or monitor trace needs to be active.

14.3 DB2 accounting data

As we discussed in 2.3, “Managing DB2 traces” on page 18, DB2 Accounting trace is written to SMF as SMF type 101 records, and it includes trace information at the transaction level.

14.3.1 When DB2 accounting data is written

This section covers when the DB2 accounting records are written.

Case 1

The most common trigger for an accounting record to be written is when a thread ends (deallocates or abends), as shown in Figure 14-4.

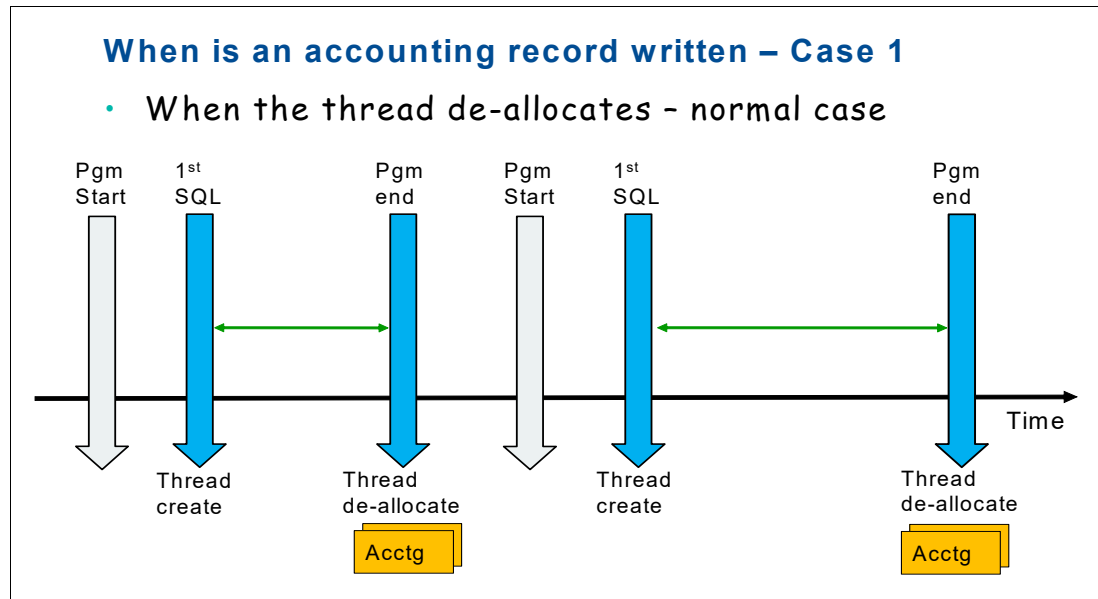


Figure 14-4 Thread Deallocate - Normal case

Case 2

When IMS or CICS reuses a thread, the sign-on or re-signon (not commit) will trigger DB2 to cut an accounting record. See Figure 14-5.

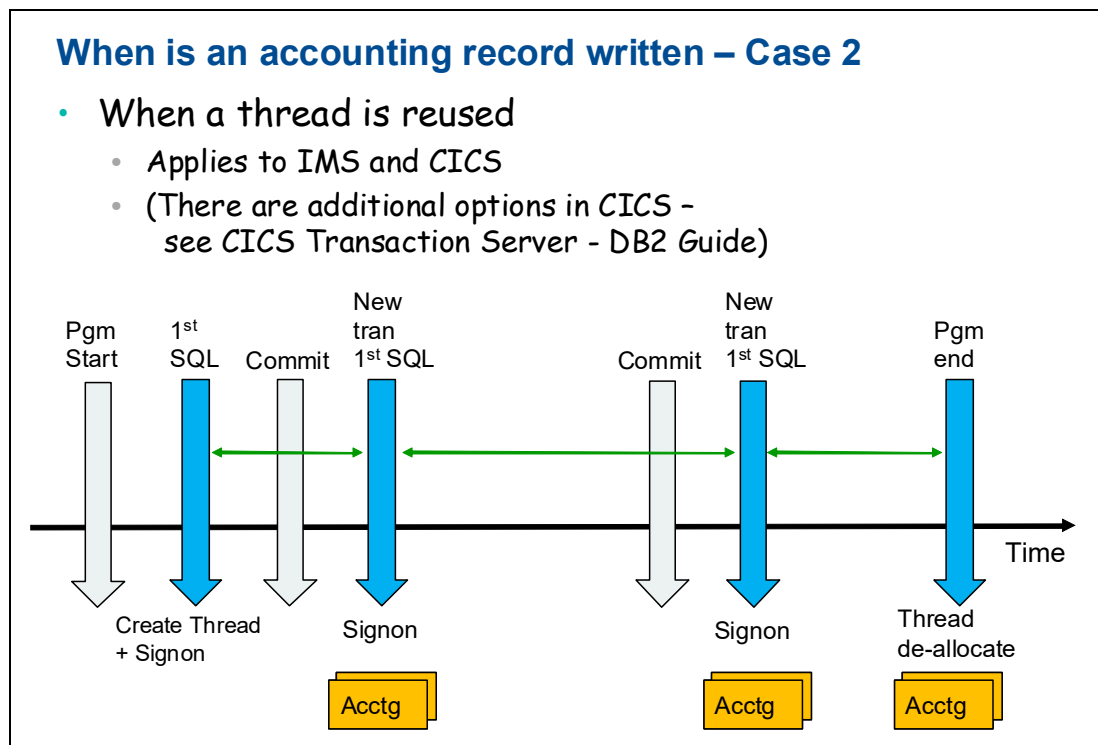


Figure 14-5 Thread Deallocate - IMS or CICS

Because accounting records are cut at re-signon, be careful when interpreting long DB2 response time information. Only a new transaction coming in will trigger DB2 to go through sign-on processing, so when there is a long time between the first transaction's end and the second transaction's start, DB2 class 1 time includes the idle time in the IMS or CICS region. See Figure 14-6.

For CICS, sometimes even a new transaction will not cut an accounting record, for example, if all of the following conditions exist:

- ▶ A thread is defined as protected (PROTECTNUM>0)
- ▶ All transactions with the same transaction code for this DB2ENTRY use the same authorization ID
- ▶ ACCOUNTREC=NONE is defined in its DB2ENTRY or DB2CONN definitions

Then only one accounting record is produced for all of the transactions that the thread processed. The other ACCOUNTREC values for CICS are UOW, TASK, or TXID. Setting ACCOUNTREC to one of these options tells CICS when to have DB2 cut an accounting record (at the end of a unit of work, a task, or a transid, respectively).

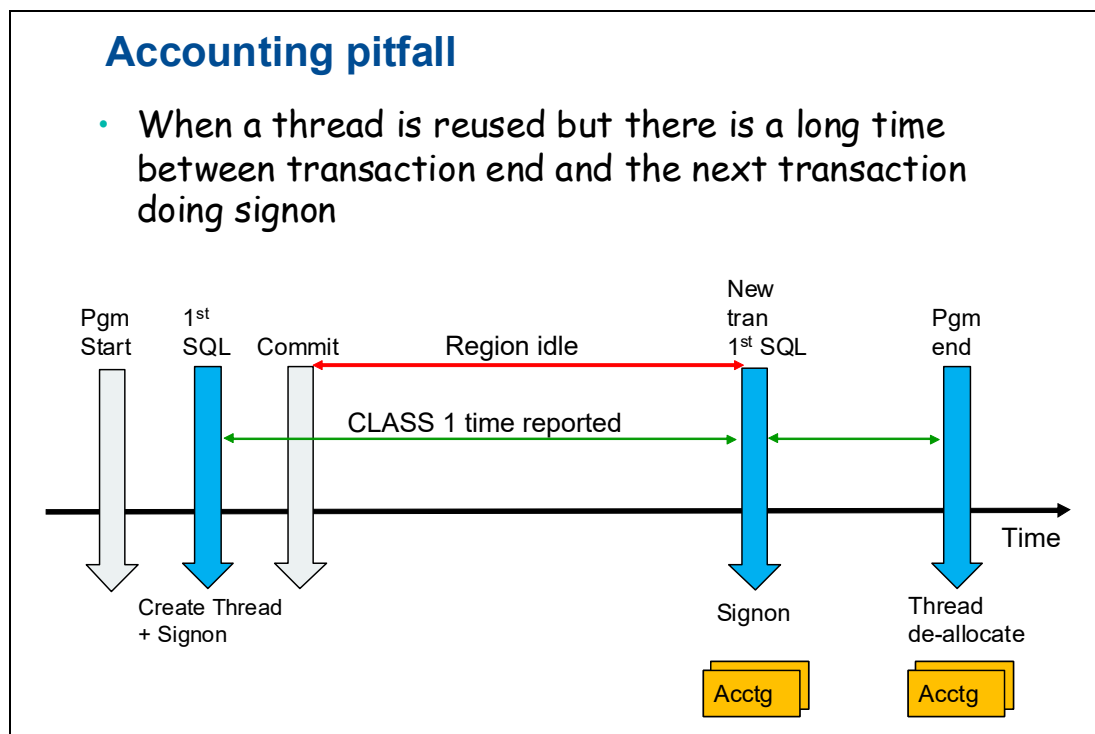


Figure 14-6 Accounting Pitfall

Case 3

In most cases, issuing an SQL COMMIT or the equivalent statement does not trigger DB2 to create an accounting records. For example, a long running batch job will not create a DB2 accounting record each time the job reaches an application checkpoint or commit, but only when the thread deallocates when the job finishes. However, as shown in Figure 14-7 and Figure 14-8, some accounting records are cut at commit. The two cases where accounting records are cut at commit are when RRS threads use accounting-interval commit (Case 3) or when DRDA threads go inactive (Case 4).

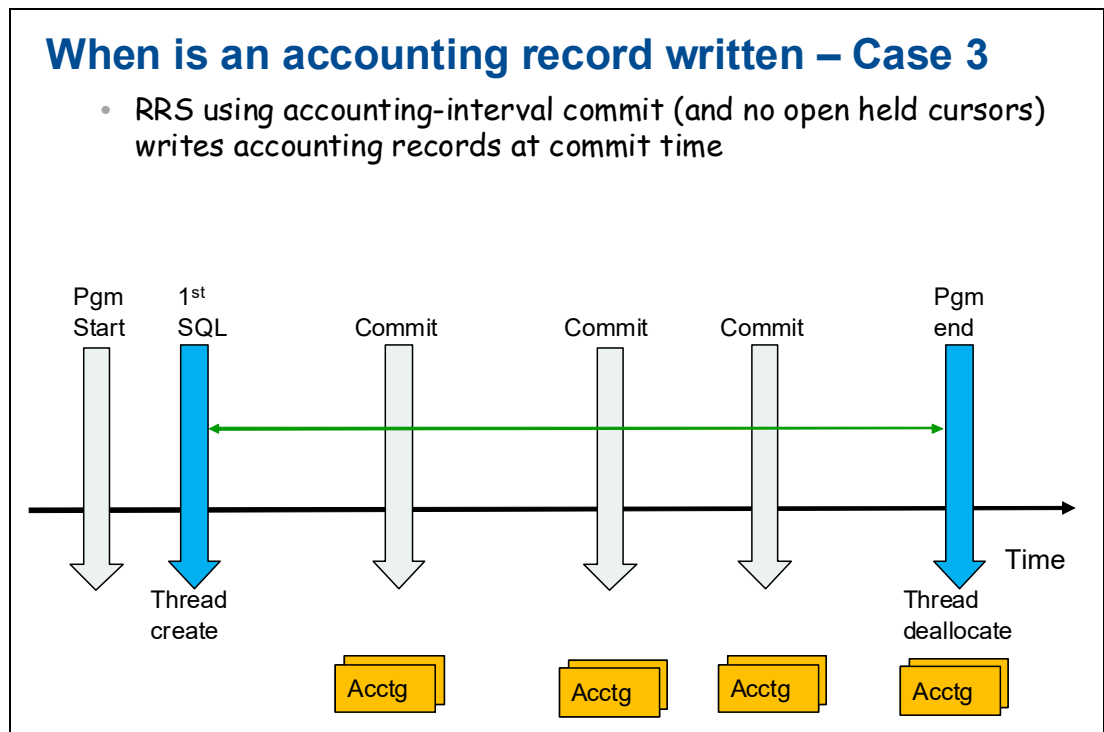


Figure 14-7 RRS using accounting interval commit it

Case 4

DDF creates accounting records at commit time, provided you are using CMTSTAT=INACTIVE and the conditions to allow a thread to be pooled are present. See Figure 14-8.

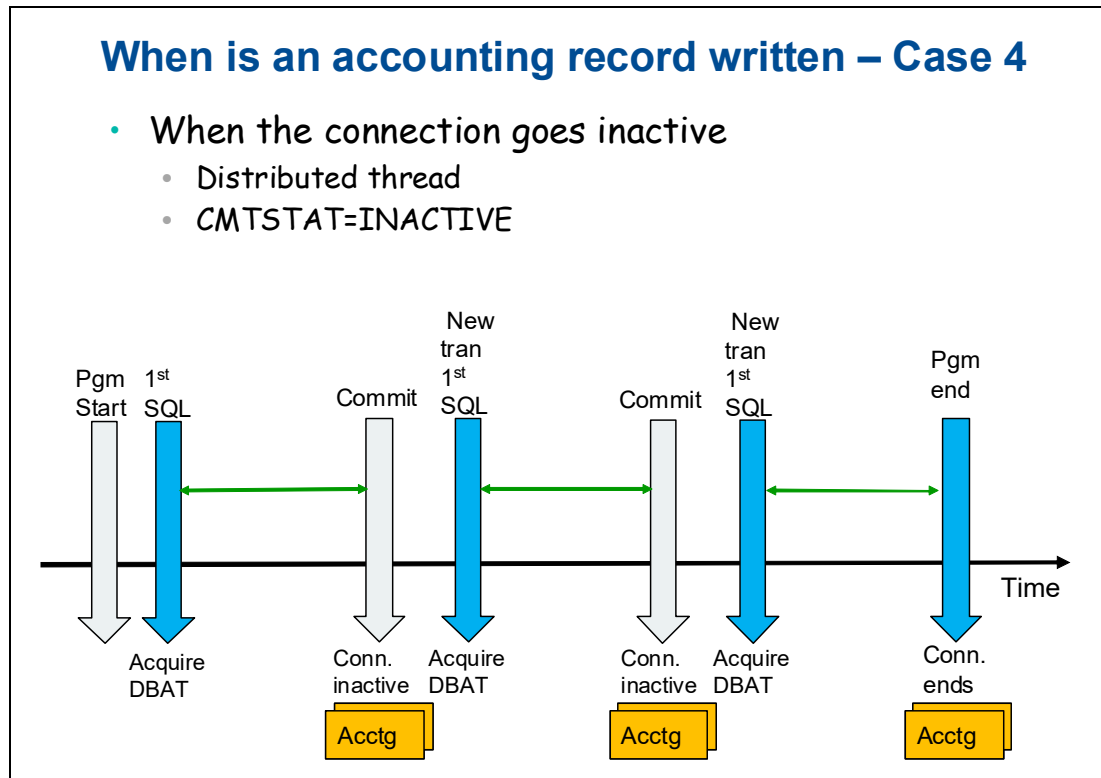


Figure 14-8 Accounting record creation when using CMTSTAT=INACTIVE

The following situations prevent a connection from going inactive:

- The transaction touched a package that uses KEEP DYNAMIC(YES). Note that we can still get an accounting record if none of the other bullets below apply.
- An active declared global temporary table (DGTT) was not explicitly or implicitly dropped.
- There is an open cursor with hold.
- There is a held LOB locator.

Note: HP DBAT still cuts accounting records just like other DBATs.

Case 5

When using rollup accounting ($ACCUMACC > 1$), DB2 will not cut an accounting record for each DDF or RRS transaction. Instead it will wait until the number of $ACCUMACC = x$ transactions have completed before creating an accounting record. This accounting record will contain the totals for all x transactions that were rolled up into this accounting record. See Figure 14-9.

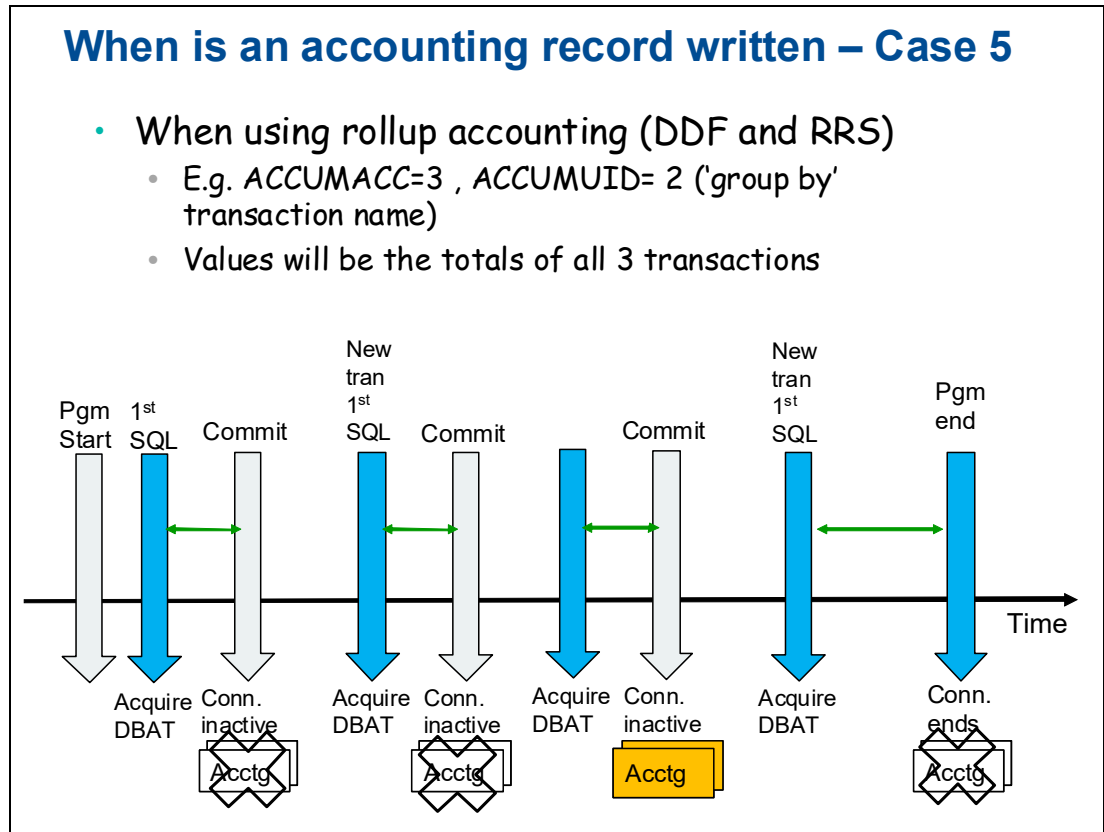


Figure 14-9 Accounting record creation when using $ACCUMACC > 1$

14.3.2 Accounting class 1 data

Class 1 time records the complete elapsed and CPU time as seen by DB2 for the life of the thread (or until it is reused). It records both time inside DB2 and time in the application (or network). Note that most of the create thread work is not included; also the last part of the thread termination process is not included in the class 1 times. See Figure 14-10.

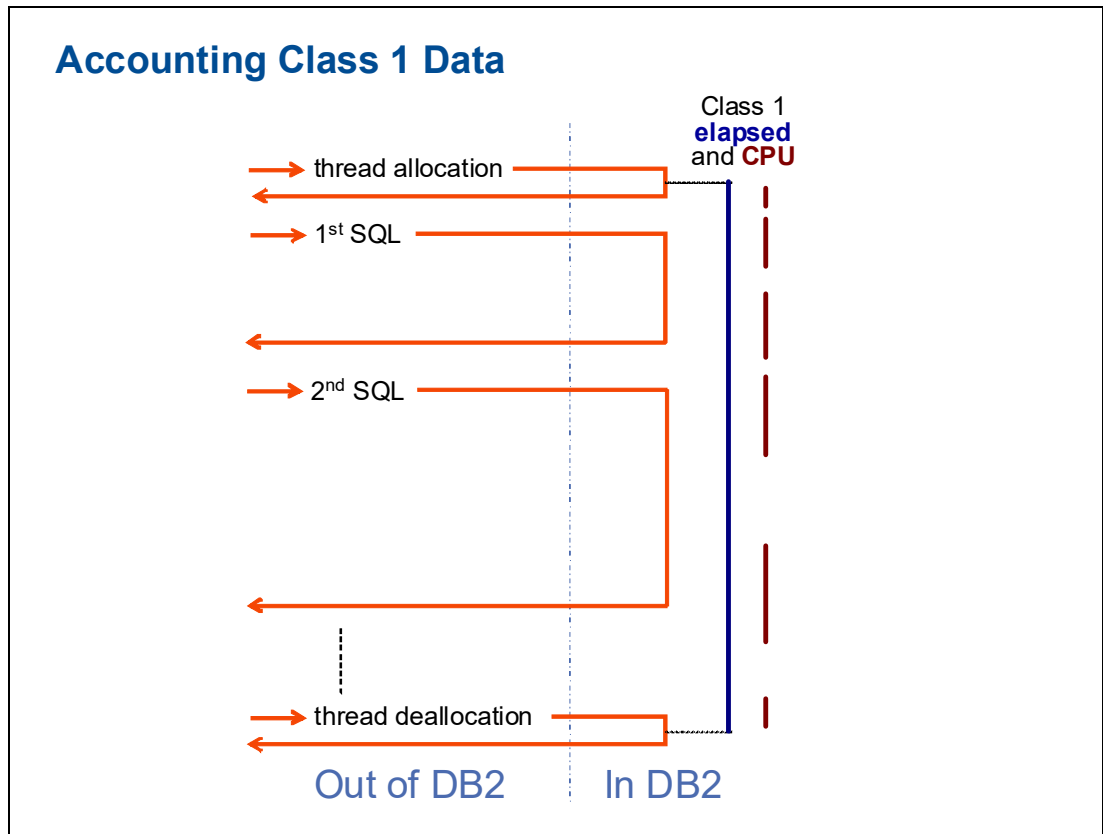


Figure 14-10 DB2 Accounting Trace Class 1

Accounting class 1 data does not only include timer information about elapsed and CPU time but also information related to the execution of SQL statement, such as the number of SELECTs, INSERTs, and so on, locking activity, the number of various lock requests, suspensions, time-outs, deadlocks, buffer pools statistics, the number of getpages, synchronous reads, various types of prefetches, writes, sort activity, parallelism data, DDF statistics, and many others.

For local applications, DB2 class 1 provides an accumulative timer that includes both CPU spent in application and CPU spent in DB2. The counter does not start until the end of thread allocation, and stops towards the end of thread deallocation, so a very small amount of “in-DB2” time is not reflected in class 1 and class 2 elapsed time.

For the applications that exploit thread reuse, such as CICS and IMS, or maintain very long connections, the class 1 data typically includes time spent in the network and the user’s think time which, makes it less relevant for monitoring and tuning purposes.

Example 14-1 shows how OMEGAMON PE lists DB2 class 1 timers. The report shows elapsed time and CPU time for nested activity (stored procedures, UDFs, and triggers) and for non-nested activities (activities that are not stored procedures, UDFs, or triggers).

Example 14-1 How OMEGAMON PE shows DB2 class 1 timers

AVERAGE	APPL (CL. 1)
-----	-----
ELAPSED TIME	0.012122
NONNESTED	0.012122
STORED PROC	0.000000
UDF	0.000000
TRIGGER	0.000000
CP CPU TIME	0.001482
AGENT	0.001482
NONNESTED	0.001482
STORED PRC	0.000000
UDF	0.000000
TRIGGER	0.000000
PAR. TASKS	0.000000

The overhead cost of accounting trace class 1 is generated mostly by writing the records out. Therefore the cost of the trace is proportional to the number of records cut. For applications with long living threads, that cost is negligible, and in general, typically very low. In some extreme cases, it could go up to 5%.

As discussed in Chapter 2, “DB2 traces” on page 13, the accounting (class 1) trace can be started automatically at DB2 start (via system parameter SMFACCT) or at any time via the **-START TRACE(A) CLASS(1)** command. Automated start of accounting trace reduces the administration overhead, especially when the cost of running the trace is low.

14.3.3 Accounting class 2 data

Unlike class 1, accounting trace class 2 includes the elapsed and CPU times spent solely in DB2. Accounting class 2 for DB2 time is normally captured immediately after getting into DBM1 address space and immediately before getting out of DBM1 address space via the DSNWVSR1/IEAVRT05 call. For a DRDA transaction, accounting class 2 time starts in DDF address space before going to DBM1 address space and after coming back from DBM1 address space. The timers are accumulative in nature. They show the elapsed and CPU time spent in DB2 since the thread allocation. See Figure 14-11.

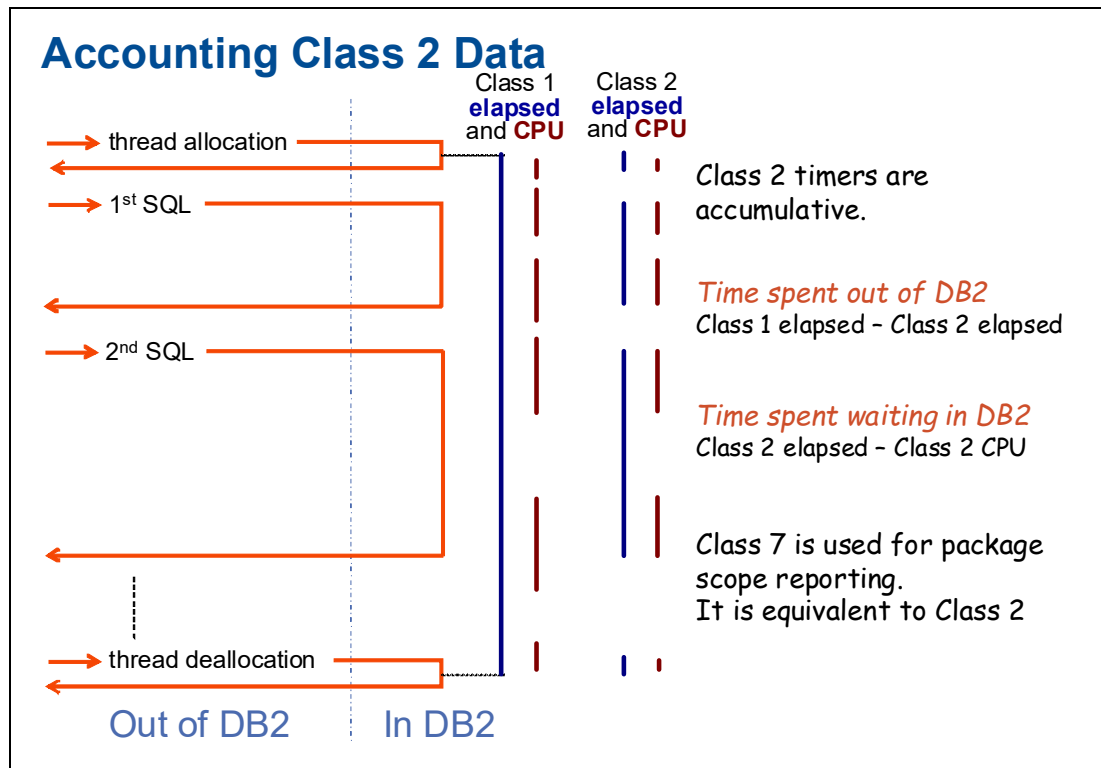


Figure 14-11 DB2 Accounting Trace Class 2

Example 14-2 illustrates how OMEGAMON PE shows DB2 class 2 timers. The report shows elapsed time and CPU time for nested activity (stored procedures, UDFs, and triggers) and for non-nested activities (activities that are not stored procedures, UDFs, or triggers). We also have SE CPU TIME, which means the CPU time spent in specialty engine (ZIIP). SUSPEND TIME is the waiting time for all types of class 3 suspensions by the originating task and parallel tasks, if parallelism is employed.

NOT ACCOUNT is the time not accounted for in DB2. This time determines whether there is a large percentage of time that has not been captured within the DB2 accounting record and whether system monitoring tools (such as RMF) should be examined to determine the cause of a performance problem. In the case of a DDF requester, this value is often large because it includes the time the requesting thread waited for responses from the server. Because there can be asynchronous activity at the requester, the DDF time is only an approximation. Refer to the DDF server blocks' requester elapsed time to determine the amount of time the thread waited for server responses. In query or utility parallelism, it is the unaccounted time of the originating task only.

DB2 ENT/EXIT is the total number of DB2 entry and exit events. This counter does not include the SQL entry and exit events processed by stored procedures (EN/EX-STPROC). EN/EX-UDF is the number of SQL entry/exit events performed by user-defined functions.

Example 14-2 How OMEGAMON PE shows DB2 class 2 timers

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)	IFI (CL.5)
-----	-----	-----	-----
ELAPSED TIME	1:42.49413	0.111129	N/P
NONNESTED	0.208342	0.057168	N/A
STORED PROC	0.000000	0.000000	N/A
UDF	1:42.23800	0.006165	N/A
TRIGGER	0.047797	0.047797	N/A
CP CPU TIME	28.858601	0.029797	N/P
AGENT	28.858601	0.029797	N/A
NONNESTED	0.016157	0.005022	N/P
STORED PRC	0.000000	0.000000	N/A
UDF	28.822345	0.004677	N/A
TRIGGER	0.020098	0.020098	N/A
PAR.TASKS	0.000000	0.000000	N/A
SECP CPU	0.000000	N/A	N/A
SE CPU TIME	0.000000	0.000000	N/A
NONNESTED	0.000000	0.000000	N/A
STORED PROC	0.000000	0.000000	N/A
UDF	0.000000	0.000000	N/A
TRIGGER	0.000000	0.000000	N/A
PAR.TASKS	0.000000	0.000000	N/A
SUSPEND TIME	27.525561	0.020768	N/A
AGENT	N/A	0.020768	N/A
PAR.TASKS	N/A	0.000000	N/A
STORED PROC	0.000000	N/A	N/A
UDF	27.525561	N/A	N/A
NOT ACCOUNT.	N/A	0.060564	N/A
DB2 ENT/EXIT	N/A	104	N/A
EN/EX-STPROC	N/A	0	N/A
EN/EX-UDF	N/A	0	N/A

Class 2 cost is proportional to the number of DB2 entry/exit events. The typical workloads that can increase this cost are high insert or fetch activity when not using multi-row operations. In that case, each insert or fetch involves one entry into and one exit from DB2. For such workloads, the overhead of running the trace can go as high as 10%.

Note that class 2 trace provides the data at the plan, or entire application level. As many applications are modular, consisting of a number of separate programs that are bound into separate packages, it is very often beneficial to have the same data on a per-program basis. That data is available only with accounting trace class 7, which is equivalent to accounting trace class 2, only at the package level.

14.3.4 Accounting class 3 data

DB2 collects detailed distribution of suspension times and events by means of accounting trace class 3. As in the class 2 trace, if the same data is needed at the program level, a separate class (8) record is available that provides equivalent information at the program and/or package level.

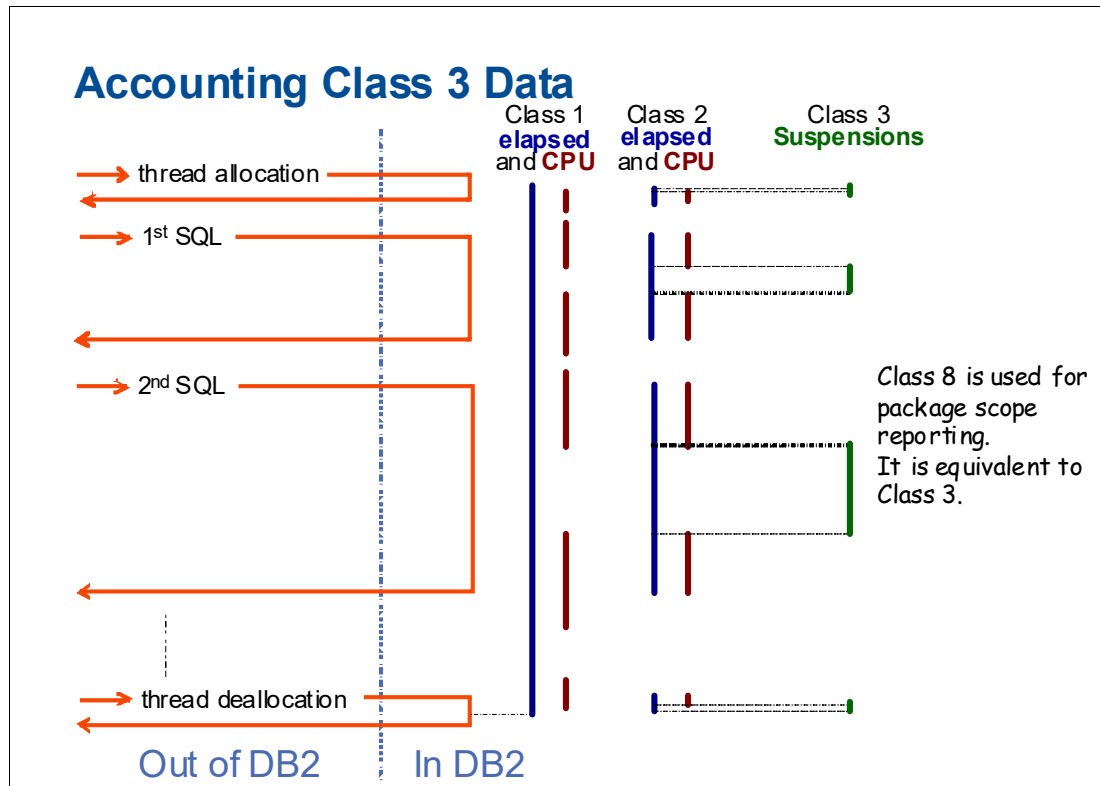


Figure 14-12 DB2 Accounting Trace Class 3

Example 14-3 shows how OMEGAMON PE lists DB2 class 3 timers. The fields includes timers spent in lock/latch, synchronous I/O, synchronous execution unit switching to DB2 services, processing of ARCHIVE LOG MODE(QUIESCE), wait time LOG READ, DRAIN LOCK, CLAIM RELEASE, PAGE LATCH contention, suspensions caused by sending notify messages to other members in the data sharing group, the suspension of IRLM lock requests due to global lock contention in a data sharing environment, TCP/IP LOB, and XML materialization.

Example 14-3 How OMEGAMON PE shows DB2 class 3 timers.

CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS
LOCK/LATCH(DB2+IRLM)	0.000000	0
IRLM LOCK+LATCH	N/A	N/A
DB2 LATCH	N/A	N/A
SYNCHRON. I/O	3.704352	8
DATABASE I/O	3.704352	8
LOG WRITE I/O	0.000000	0
OTHER READ I/O	0.000000	0
OTHER WRTE I/O	0.000000	0

SER.TASK SWTCH	0.000000	0
UPDATE COMMIT	0.000000	0
OPEN/CLOSE	0.000000	0
SYSLGNG REC	0.000000	0
EXT/DEL/DEF	0.000000	0
OTHER SERVICE	0.000000	0
ARC.LOG(QUIES)	0.000000	0
LOG READ	0.000000	0
DRAIN LOCK	0.000000	0
CLAIM RELEASE	0.000000	0
PAGE LATCH	0.000000	0
NOTIFY MSGS	0.000000	0
GLOBAL CONTENTION	0.000000	0
COMMIT PH1 WRITE I/O	0.000000	0
ASYNCH CF REQUESTS	0.000000	0
TCP/IP LOB XML	0.000000	0
TOTAL CLASS 3	3.704352	8

The suspension timers are started when a suspension event is encountered and stopped after the suspension is resolved. As in class 2, these times are accumulative in nature. Consequently, the cost of class 3 and 8 tracing is proportional to the number of suspensions. Here are some of them:

- ▶ Sync Database I/O wait is the wait for read or write I/O by this application agent.
- ▶ Other read I/O wait is the wait for read I/O by another application agent or prefetch engine.
- ▶ Other write I/O wait is the wait for write I/O by another application agent or write engine. It may include some time waiting for log write-ahead. Note that when updating a buffer being written, the first agent waits for OTHER WRITE I/O. All others wait for PAGE LATCH contention.
- ▶ The suspension type that has the largest potential to drive the cost of classes 3 and 8 is the DB2 internal latch suspension. If the number of internal DB2 latch suspensions is more than 10000/sec, the cost of classes 3 and 8 can grow significantly. If that happens, first try to address the cause of the problem by reducing the number of latch suspensions. Once you have the data you need for analysis, you can switch off classes 3 and 8 until the problem is resolved, allowing you more CPU for productive work. Otherwise the overhead could be so large that having the trace on would do more harm than good.

Typically, the class 3 cost is no higher than 3%, providing the number of internal latch suspensions is at the normal level. DB2 reports both the accumulated time spent being suspended and the number of these events. This allows calculation of an average suspension duration.

Note that when you use the raw DB2 accounting data, the divisor (the number of events) needs to be first divided by 2 for most class 3 counters, because DB2 increments the counter twice for each suspension (+1 when suspension starts, +1 when suspension ends). OMEGAMON PE already divides these counters where needed.

Class 5 accounting trace collects and reports elapsed and CPU times spent in the instrumentation facility interface, which monitors and data capture processing use.

14.3.5 Accounting class 1,2,3

In summary, Class 1 time records the complete elapsed and CPU time during DB2 thread duration; Class 2 includes the elapsed and CPU times spent inside DB2; and Class 3 times are suspension times. Figure 14-13 shows which part of the processing is recorded as class1/class2 elapsed and CPU times, as well as class 3 suspension time. It shows a simple transaction that only executes a few simple SQL statements, and how the time is recorded.

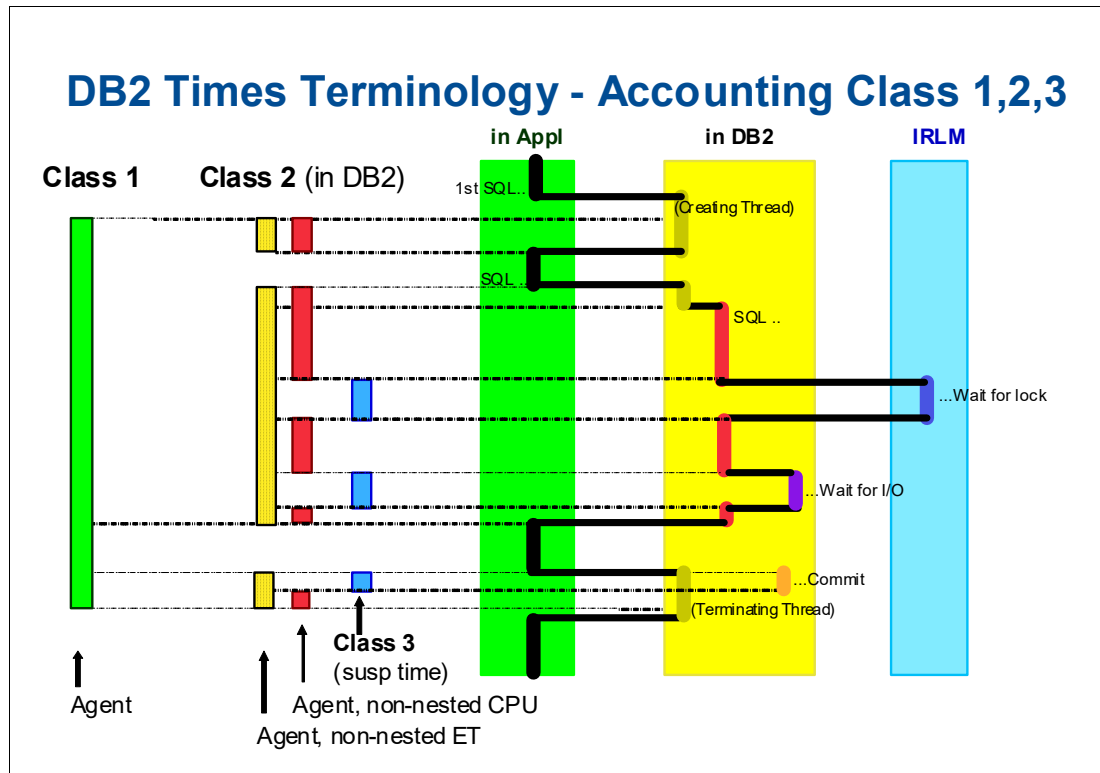


Figure 14-13 Recording class1/class2 elapsed and CPU times and class 3 suspension time



Analyzing accounting data: CPU and suspension time

You can use DB2 accounting reports to analyze the transactions grouped by connection type and then look at thread level for their execution time inside and outside of DB2.

This chapter contains the following topics:

- ▶ Top-down analysis
- ▶ Time outside of DB2 versus time in DB2
- ▶ In-DB2 CPU time versus elapsed time

15.1 Top-down analysis

Analyzing DB2 accounting data typically uses a top-down approach. We begin with the highest level, the connection type. An application connects to DB2 in one of the following ways:

- ▶ CICS attachment facility, used to access DB2 from CICS application programs.
- ▶ IMS attachment facility (MPP, BMP, DLI, and so on) used to access DB2 from IMS application programs.
- ▶ Time Sharing Option (TSO) attachment facility, used to communicate with DB2 in a TSO or batch environment. This facility invokes the DSN command processor.
- ▶ Call attachment facility (CAF), used for batch applications that require tight control over the session environment.
- ▶ Resource Recovery Services attachment facility (RRSAF), used as an alternative to the CAF. RRS is also used for WebSphere Application Server (WAS) when it runs on z/OS, and for stored procedures that run in a WLM-established address space.
- ▶ Distributed data facility (DDF), used for distributed transactions.
- ▶ Utility, the interface used for DB2 Utilities.

Example 15-1 shows an example of OMEGAMON PE Accounting Report JCL that you can use to group accounting data by connection type.

Example 15-1 OMEGAMON PE Accounting Report -JCL

```
/*JOB
//PE      EXEC PGM=FPECMAN
//STEPLIB DD DISP=SHR,DSN=OMPE.V520.D130306.V11DRP5.TKANMOD
//INPUTDD DD DISP=SHR,DSN=SMFDATA.DB2RECS.G5848V00
//        DD DISP=SHR,DSN=SMFDATA.DB2RECS.G5849V00
//        DD DISP=SHR,DSN=SMFDATA.DB2RECS.G5850V00
//JOBSUMDD DD SYSOUT=*
//SYSOUT  DD SYSOUT=*
//ACRPTDD DD SYSOUT=*
//UTTRCDD1 DD SYSOUT=*
//SYSIN   DD *
ACCOUNTING REPORT LAYOUT(LONG)
ORDER(CONNTYPE) EXCLUDE(PACKAGE(*))
EXEC
//
```

Example 15-2 shows an examples of OMEGAMON PE Accounting Report output ordered by connection type (DB2CALL).

Example 15-2 OMEGAMON PE accounting report for call attach connection type

```
***** TOP OF DATA *****
LOCATION: DB1A          OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R2M0)          PAGE: 1-1
GROUP: N/P              ACCOUNTING REPORT - LONG                                REQUESTED FROM: NOT SPECIFIED
MEMBER: N/P              ORDER: CONNTYPE                                         TO: NOT SPECIFIED
SUBSYSTEM: DB1A          SCOPE: MEMBER                                           INTERVAL FROM: 08/29/13 21:10:50.40
DB2 VERSION: V11                                               TO: 08/29/13 21:21:29.58

CONNTYPE: DB2CALL
```

ELAPSED TIME DISTRIBUTION

```

APPL |=====> 33%
DB2  |=====> 14%
SUSP |=====> 54%

```

CLASS 2 TIME DISTRIBUTION

```

CPU   |=====> 18%
SECPU |=> 2%
NOTACC
SUSP  |=====> 80%

```

AVERAGE	APPL(CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT	HIGHLIGHTS
ELAPSED TIME	1:28.07656	59.265543	N/P	LOCK/LATCH(DB2+IRLM)	0.046705	16900.67	#OCCURRENCES : 3
NONNESTED	4.911356	3.890972	N/A	IRLM LOCK+LATCH	0.001927	9.00	#ALLIEDS : 3
STORED PROC	54.741269	54.741269	N/A	DB2 LATCH	0.044778	16891.67	#ALLIEDS DISTRIB: 0
UDF	27.790672	0.000035	N/A	SYNCHRON. I/O	22.706446	115.1K	#DBATS : 0
TRIGGER	0.633267	0.633267	N/A	DATABASE I/O	22.582423	114.7K	#DBATS DISTRIB. : 0
				LOG WRITE I/O	0.124023	363.33	#NO PROGRAM DATA: 0
CP CPU TIME	11.762048	10.908406	N/P	OTHER READ I/O	19.445122	41687.33	#NORMAL TERMINAT: 3
AGENT	11.762048	10.908406	N/A	OTHER WRTE I/O	1.900742	1335.33	#DDFRSAF ROLLUP: 0
NONNESTED	1.479492	0.626170	N/P	SER.TASK SWTCH	3.235348	5600.67	#ABNORMAL TERMIN: 0
STORED PROC	10.126918	10.126918	N/A	UPDATE COMMIT	3.169034	5600.00	#CP/X PARALLEL. : 0
UDF	0.000353	0.000033	N/A	OPEN/CLOSE	0.000000	0.00	#UTIL PARALLEL. : 0
TRIGGER	0.155285	0.155285	N/A	SYSLGRNG REC	0.000000	0.00	#IO PARALLELISM : 0
PAR.TASKS	0.000000	0.000000	N/A	EXT/DEL/DEF	0.066314	0.67	#PCA RUP COUNT : 0
				OTHER SERVICE	0.000000	0.00	#RUP AUTONOM. TX: 0
SECP CPU	0.000000	N/A	N/A	ARC.LOG(QUIES)	0.000000	0.00	#AUTONOMOUS TX : 0
				LOG READ	0.000000	0.00	#INCREMENT. BIND: 5480
SE CPU TIME	0.004833	0.003775	N/A	DRAIN LOCK	0.000000	0.00	#COMMITTS : 16851
NONNESTED	0.001173	0.000115	N/A	CLAIM RELEASE	0.000000	0.00	#ROLLBACKS : 0
STORED PROC	0.003660	0.003660	N/A	PAGE LATCH	0.000000	0.00	#SVPT REQUESTS : 0
UDF	0.000000	0.000000	N/A	NOTIFY MSGS	0.000000	0.00	#SVPT RELEASE : 0
TRIGGER	0.000000	0.000000	N/A	GLOBAL CONTENTION	0.000000	0.00	#SVPT ROLLBACK : 0
				COMMIT PH1 WRITE I/O	0.000000	0.00	MAX SQL CASC LVL: 2
PAR.TASKS	0.000000	0.000000	N/A	ASYNCH CF REQUESTS	0.000000	0.00	UPDATE/COMMIT : 9.24
				TCP/IP LOB XML	0.000000	0.00	SYNCH I/O AVG. : 0.000197
SUSPEND TIME	27.701912	47.334363	N/A	ACCELERATOR	0.000000	0.00	
AGENT	N/A	47.334363	N/A	AUTONOMOUS TX	0.000000	0.00	
PAR.TASKS	N/A	0.000000	N/A	TOTAL CLASS 3	47.334363	180.6K	
STORED PROC	0.000000	N/A	N/A				
UDF	27.701912	N/A	N/A				
NOT ACCOUNT.	N/A	1.019000	N/A				
DB2 ENT/EXIT	N/A	22214.00	N/A				
EN/EX-STPROC	N/A	0.00	N/A				
EN/EX-UDF	N/A	0.00	N/A				
DCAPT.DESCR.	N/A	N/A	N/P				
LOG EXTRACT.	N/A	N/A	N/P				

After finding which connection type is having the problem, you can look at the different plans, transactions, and authorization id that are used by that connection type to see which ones are responsible for the problem you are experiencing. Once you identify the problem plan, authorization id, and transaction, review the packages that it invokes to see if there is a specific one or few that is the root cause of the problem.

DB2 accounting information does not provide information at the individual SQL statement level. That requires DB2 performance trace class 3 for static SQL, or the use of IFCID 318 with EXPLAIN STMTCACHE ALL for dynamic SQL, or can also use IFCID 400 (and 401 for static SQL). Section 20.4, “Capturing performance data with the statement cache” on page 284 describes the use of the dynamic statement cache to monitor dynamic SQL at the statement level.

In DB2 accounting data, you can distinguish between non-nested activity and nested activity such as trigger, stored procedure, and user defined functions. There is more information about nested activity in Chapter 19, “Stored procedures, user defined functions, and triggers” on page 265.

It is always recommended also to have a look at the DB2 statistics report for the same time frame and match that to the DB2 accounting information. See Example 15-3.

Example 15-3 OMEGAMON PE Statistics Report -JCL

```

/*JOB
//PE      EXEC PGM=FPECMAIN
//STEPLIB DD DISP=SHR,DSN=OMPE.V520.D130306.V11DRP5.TKANMOD
//INPUTDD DD DISP=SHR,DSN=SMFDATA.DB2RECS.G5848V00
//        DD DISP=SHR,DSN=SMFDATA.DB2RECS.G5849V00
//        DD DISP=SHR,DSN=SMFDATA.DB2RECS.G5850V00
//JOBSUMDD DD SYSOUT=*
//SYSOUT  DD SYSOUT=*
//ACRPTDD DD SYSOUT=*
//UTTRCDD1 DD SYSOUT=*
//SYSIN   DD *
STATISTICS REPORT LAYOUT(LONG)
EXEC
//

```

Example 15-4 shows an example of OMEGAMON PE Statistics Report output.

Example 15-4 OMEGAMON PE Statistics Report

LOCATION: DBOZ GROUP: DBOZG MEMBER: DOZ1 SUBSYSTEM: DOZ1 DB2 VERSION: DB2 10	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R2M0) STATISTICS REPORT - LONG SCOPE: MEMBER	PAGE: 1-1 REQUESTED FROM: NOT SPECIFIED TO: NOT SPECIFIED INTERVAL FROM: 08/29/13 21:11:00.00 TO: 08/29/13 21:21:00.00
--	--	--

```

---- HIGHLIGHTS -----
INTERVAL START : 08/29/13 21:11:00.00  SAMPLING START: 08/29/13 21:11:00.00  TOTAL THREADS   :    0.00
INTERVAL END   : 08/29/13 21:21:00.00  SAMPLING END   : 08/29/13 21:21:00.00  TOTAL COMMITS   :   10.00
INTERVAL ELAPSED:    9:59.996443      OUTAGE ELAPSED:    0.000000      DATA SHARING MEMBER: N/A

```

SQL DML	QUANTITY	/SECOND	/THREAD	/COMMIT	SQL DCL	QUANTITY	/SECOND	/THREAD	/COMMIT
SELECT	0.00	0.00	N/C	0.00	LOCK TABLE	0.00	0.00	N/C	0.00
INSERT	0.00	0.00	N/C	0.00	GRANT	0.00	0.00	N/C	0.00
NUMBER OF ROWS	0.00	0.00	N/C	0.00	REVOKE	0.00	0.00	N/C	0.00
UPDATE	10.00	0.02	N/C	1.00	SET HOST VARIABLE	0.00	0.00	N/C	0.00
NUMBER OF ROWS	10.00	0.02	N/C	1.00	SET CURRENT SQLID	0.00	0.00	N/C	0.00
MERGE	0.00	0.00	N/C	0.00	SET CURRENT DEGREE	0.00	0.00	N/C	0.00
DELETE	60.00	0.10	N/C	6.00	SET CURRENT RULES	0.00	0.00	N/C	0.00
NUMBER OF ROWS	0.00	0.00	N/C	0.00	SET CURRENT PATH	0.00	0.00	N/C	0.00
					SET CURRENT PRECISION	0.00	0.00	N/C	0.00
PREPARE	0.00	0.00	N/C	0.00					
DESCRIBE	0.00	0.00	N/C	0.00	CONNECT TYPE 1	0.00	0.00	N/C	0.00
DESCRIBE TABLE	0.00	0.00	N/C	0.00	CONNECT TYPE 2	0.00	0.00	N/C	0.00
OPEN	10.00	0.02	N/C	1.00	RELEASE	0.00	0.00	N/C	0.00
CLOSE	10.00	0.02	N/C	1.00	SET CONNECTION	0.00	0.00	N/C	0.00
FETCH	10.00	0.02	N/C	1.00					
NUMBER OF ROWS	10.00	0.02	N/C	1.00	ASSOCIATE LOCATORS	0.00	0.00	N/C	0.00
					ALLOCATE CURSOR	0.00	0.00	N/C	0.00
TOTAL DML	100.00	0.17	N/C	10.00					
					HOLD LOCATOR	0.00	0.00	N/C	0.00
					FREE LOCATOR	0.00	0.00	N/C	0.00
					TOTAL	0.00	0.00	N/C	0.00

STORED PROCEDURES	QUANTITY	/SECOND	/THREAD	/COMMIT	TRIGGERS	QUANTITY	/SECOND	/THREAD	/COMMIT
CALL STATEMENT EXECUTED	0.00	0.00	N/C	0.00	STATEMENT TRIGGER ACTIVATED	0.00	0.00	N/C	0.00
PROCEDURE ABENDED	0.00	0.00	N/C	0.00	ROW TRIGGER ACTIVATED	0.00	0.00	N/C	0.00
CALL STATEMENT TIMED OUT	0.00	0.00	N/C	0.00	SQL ERROR OCCURRED	0.00	0.00	N/C	0.00
CALL STATEMENT REJECTED	0.00	0.00	N/C	0.00					

USER DEFINED FUNCTIONS	QUANTITY	/SECOND	/THREAD	/COMMIT	ROW ID	QUANTITY	/SECOND	/THREAD	/COMMIT
EXECUTED	0.00	0.00	N/C	0.00	DIRECT ACCESS	0.00	0.00	N/C	0.00
ABENDED	0.00	0.00	N/C	0.00	INDEX USED	0.00	0.00	N/C	0.00
TIMED OUT	0.00	0.00	N/C	0.00	TABLE SPACE SCAN USED	0.00	0.00	N/C	0.00
REJECTED	0.00	0.00	N/C	0.00					

15.1.1 Analyzing thread activity time

The thread activity time coincides with the class 1 elapsed time, the times between the thread allocation, and the thread de-allocation. If class2 is active, it is further broken down into the time spent in DB2 and time spent in applications.

The time spent in DB2 can be divided into time spent processing, CPU time, and time spent waiting for some kind of suspension. Which kind of suspensions were involved in these waits is visible from the class 3 accounting trace data, which specifies the suspension events happened in the thread lifetime, such as these:

- ▶ I/O and locking
- ▶ Data set operations such as extensions or formatting
- ▶ Commits with logging

In most cases, this information provides very useful pointers to where the performance problem causes might be. There is also a part of the waiting time in DB2 that cannot be attributed to any suspension event, because DB2 is a formal subsystem of z/OS, and in some cases, the operating system events can cause waits of which DB2 is not aware of. For example, CPU contention or paging can cause increases in the elapsed time spent in DB2, but no other counter gets incremented. This time is called “not accounted” time and in most cases, it points to some operating system specific constraints. Figure 15-1 shows how the thread activity time is split.

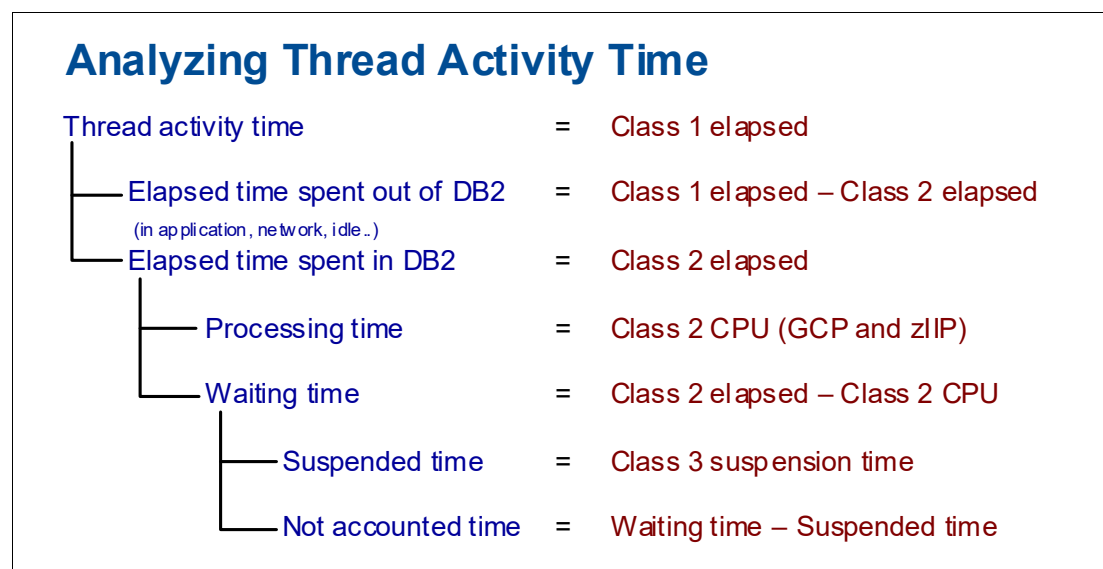


Figure 15-1 Analyzing thread activity time

15.2 Time outside of DB2 versus time in DB2

Tuning considerations apply only if the application experiences some performance problems. There are a large number of reasons why the overall applications performance might not be optimal. One of the most important is the application design and code efficiency itself.

The following sections on accounting times present the situations when the elapsed time distribution is skewed towards a particular activity.

15.2.1 Time spent outside of DB2 is bigger than time in-DB2

A very large time spent outside of DB2 could be a normal application pattern. An example is applications that use DB2 threads as “service queues,” where the thread’s duration is extremely long. In this case, very large “out-of-DB2” time is normal because it includes the time spent at presentation server, application server, network, and most importantly, users’ think-time.

When the time spent outside of DB2 is not the application pattern, look for potential application logic inefficiencies. They are regularly accompanied with unproportionally large class 1 CPU time. Another frequent culprit is inappropriate network performance, either due to configuration errors or simply over utilization. See Figure 15-2.

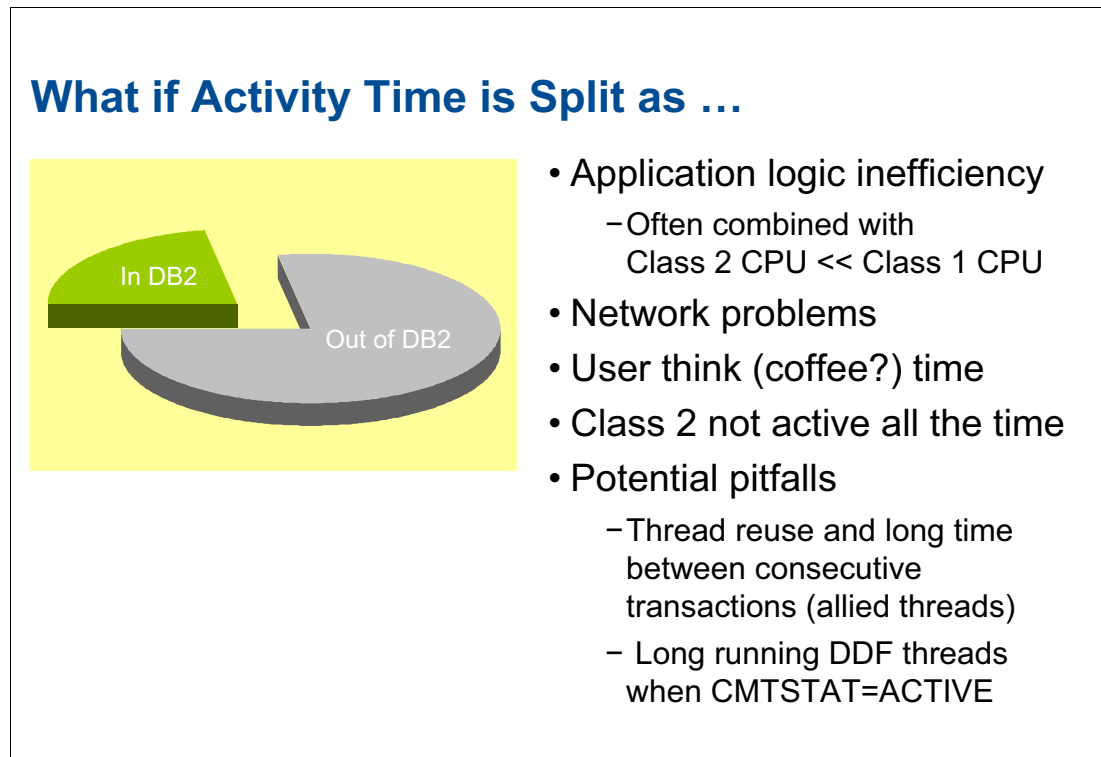


Figure 15-2 Time spent outside of DB2 is bigger than time in-DB2

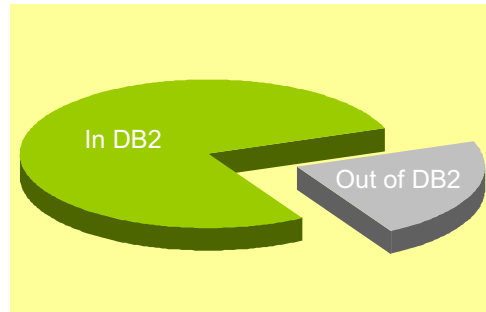
Note: DB2 class 2 (or 7 for packages) trace must be active all the time to report the precise time distribution. If it is not, the time spent outside of DB2 may be incorrectly reported as higher than it is.

15.2.2 Time spent inside of DB2 is bigger than time outside of DB2

Be aware that there is nothing wrong with a thread spending lots of time in DB2. For example, a batch job with very light application logic could report high in-DB2 time.

If the time spent in DB2 is larger than what the application characteristics would dictate, the elapsed time distribution analysis is due. We need to find out if the excessive time is spent processing or waiting, and if waiting, what kind of suspensions are causes for the waits. Also we should not just look at the percentages, but also the actual values. A transaction that takes 0.1 ms and spends 90% in DB2 may not be something to worry about. See Figure 15-3.

What if Activity Time is Split as ...

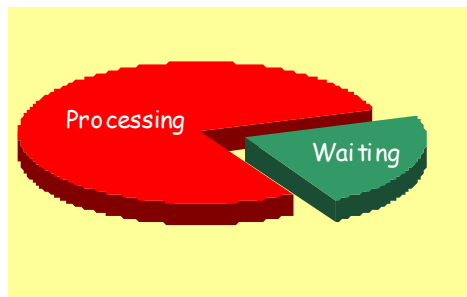


- What's the actual number (not just the %)
- Is this a DB2 intensive process anyway ?
- Need to **analyze time distribution in DB2**:
Where is the time really spent inside DB2 ?

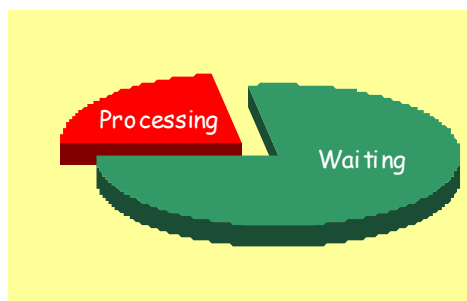
Figure 15-3 Time spent inside of DB2 is bigger than time outside of DB2

Once the conclusion is made that the problem is somewhere in DB2, the first question that needs to be answered is: "is most of the time spent in burning CPU or waiting"? See Figure 15-4.

What if In DB2 Time is Split as ...



- Turn off expensive traces
- Inefficient access paths
 - Explain
 - Check tran profile
 - #SQL stmts,
 - #locks,
 - #getpages



- What is the largest contributor
 - Class 3 and 8 analysis

Figure 15-4 Is most of the time spent in burning CPU or waiting

15.3 In-DB2 CPU time versus elapsed time

Ideally, the processing time should be a major part of the time spent in DB2. However, it should not be larger than absolutely needed for accomplishing the task.

15.3.1 CPU time

An easy problem cause to solve is if some expensive traces have been turned on and inadvertently stayed active longer than planned. For example, some global or performance traces can bring very large overhead. Turning off these traces is a quick and very efficient “tuning” step.

However, the most common reason for an excessive CPU time is a less than optimal access path. Resolving the problem in this case is variable in its difficulty. The prime tool to start with is the DB2 Explain. All the statements that could use inefficient access paths need to be explained and the explain output analyzed for possible less than optimal access paths. This topic is a book for itself and we need to leave it at this level of explanation. Here is a list of tips on what to watch for:

- ▶ Table space scans
- ▶ Non-matching index accesses
- ▶ Wrong join orders
- ▶ Fewer than the maximum possible columns in a matching index scan
- ▶ Out-of-date catalog statistics; use DB2 real time statistics to know when RUNSTATs is needed, and when it is needed, to run it

Specialty engines

Figure 15-5 shows a sample accounting report showing CPU time distribution with specialty engine values, in this case, a zIIP.

Accounting CPU time reporting		
CONNTYPE: DRDA		
AVERAGE	APPL (CL.1)	DB2 (CL.2)
-----	-----	-----
ELAPSED TIME	2.840103	1.913457
NONNESTED	2.839901	1.913339
STORED PROC	0.000202	0.000118
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
CP CPU TIME	0.260398	0.251655
AGENT	0.260398	0.251655
NONNESTED	0.260326	0.251584
STORED PRC	0.000072	0.000071
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
PAR.TASKS	0.000000	0.000000
SECP CPU	0.000726	N/A
SE CPU TIME	0.261583	0.246227
NONNESTED	0.261583	0.246227
STORED PROC	0.000000	0.000000
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
PAR.TASKS	0.000000	0.000000

Figure 15-5 zIIP off load

In this report, we see the following fields:

- CP CPU time:
 - CPU time used on a general purpose engine
- SE CPU time:
 - Any CPU used on a specialty engine, zIIP, or zAAP, not included in CP CPU time
- SECP CPU:
 - Could have run on specialty engine but did not; this CPU time is already included in the CP CPU time

Note that in subcapacity machines, SE CPU time is reported as if had run on general CP. It can result in total CPU= (CP + SE CPU) > class 2 elapse time for CPU intensive work.

Example 15-5 is an example of an accounting trace when class 2 SE CPU time is longer than class 2 elapsed time (the SE CPU TIME is 0.003645 and ELAPSED TIME is 0.002441).

Example 15-5 When class 2 SE CPU time is longer than class 2 elapsed time

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)
-----	-----	-----
ELAPSED TIME	0.007249	0.002441
NONNESTED	0.007249	0.002441
STORED PROC	0.000000	0.000000
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
CP CPU TIME	0.000448	0.000448
AGENT	0.000448	0.000448
NONNESTED	0.000448	0.000448
STORED PRC	0.000000	0.000000
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
PAR.TASKS	0.000000	0.000000
SECP CPU	0.000448	N/A
SE CPU TIME	0.003744	0.003645
NONNESTED	0.003744	0.003645
STORED PROC	0.000000	0.000000
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
PAR.TASKS	0.000000	0.000000
SUSPEND TIME	0.000000	0.000000
AGENT	N/A	0.000000
PAR.TASKS	N/A	0.000000
STORED PROC	0.000000	N/A
UDF	0.000000	N/A

15.3.2 Suspension time

A large wait time requires a deeper look into the suspensions time distribution. This requires accounting class 3 and in most cases accounting class 8 data to detect which suspensions were major contributors to the excessive wait times. Class 3 and 8 accounting provides information about the type, number, and total time of suspensions. Note that it is always important to look at both the number of suspensions as well as the suspension time.

Note: For raw SMF data, keep in mind that for most suspensions DB2 increments the suspension count at the start and at the end of the suspension. Monitors such as OMEGAMON PE divide this number by 2 whenever needed, which provides the actual number of suspensions.

Class 3 suspension types

If the DB2 suspension time is the major part of “in-DB2 time,” its distribution needs to be examined. DB2 has been steadily adding new suspension 3 data over time, significantly improving its monitoring and tuning capabilities. The table in Figure 15-6 shows DB2 class 3 suspension types.

Category	Class 3 Suspension Type	V5	V6	V7,8	V9	V10	V11
I/O	Synchronous read/write & log write	✓					
	Synchronous read/write		✓	✓	✓	✓	✓
	Log write		✓	✓	✓	✓	✓
	Other agents' read	✓	✓	✓	✓	✓	✓
	Other agents' write	✓	✓	✓	✓	✓	✓
	Force-at-commit DB writes (LOG NO LOBs)		✓	✓	✓	✓	✓
	TCP/IP LOB XML				✓	✓	✓
Locking	IRLM lock/latch & DB2 internal latch	✓	✓	✓	✓		
	IRLM lock/latch					✓	✓
	DB2 internal latch					✓	✓
	Page latch	✓	✓	✓	✓	✓	✓
	Drain lock	✓	✓	✓	✓	✓	✓
	Claim release	✓	✓	✓	✓	✓	✓
Execution Unit Switch	Synchronous EU Switch total	✓					
	Open/Close		✓	✓	✓	✓	✓
	Define/Extend/Delete		✓	✓	✓	✓	✓
	SYSLGRNX recording		✓	✓	✓	✓	✓
	Commit		✓	✓	✓	✓	✓
	Other services		✓	✓	✓	✓	✓
Archiving	Archive Log command	✓	✓	✓	✓	✓	✓
	[Archive] log read	✓	✓	✓	✓	✓	✓
Waits for autonomous transactions	Stored procedures						✓
	UDFs						✓
Data Sharing	Global locks total	✓	✓				
	Parent L-locks			✓	✓	✓	✓
	Child L-locks			✓	✓	✓	✓
	Other L-locks			✓	✓	✓	✓
	Page set/Partition P-locks			✓	✓	✓	✓
	Page P-locks			✓	✓	✓	✓
	Other P-locks			✓	✓	✓	✓
	Sending Notify messages	✓	✓	✓	✓	✓	✓
	Asynchronous CF request completion		✓	✓	✓	✓	✓
Accelerator	Accelerator					✓	✓

Figure 15-6 Class 3 suspension types

Wait times are discussed in detail in Chapter 16, “I/O suspensions” on page 231 through Chapter 19, “Stored procedures, user defined functions, and triggers” on page 265.

Other DB2 wait time

One DB2 delay that you may see that has no class 3 counters occurs when a stored procedure or a UDF is waiting for a WLM TCB or for the WLM environment itself. These simply show up as class 1 suspension times. There is more information on this example in Chapter 19, “Stored procedures, user defined functions, and triggers” on page 265.

15.3.3 What is left: NOT ACCOUNT time

Suppose in-DB2 CPU plus in-DB2 waits is less than the in-DB2 elapsed time of the transaction? What you have left is NOT ACCOUNT(ed) time. See Figure 15-7.

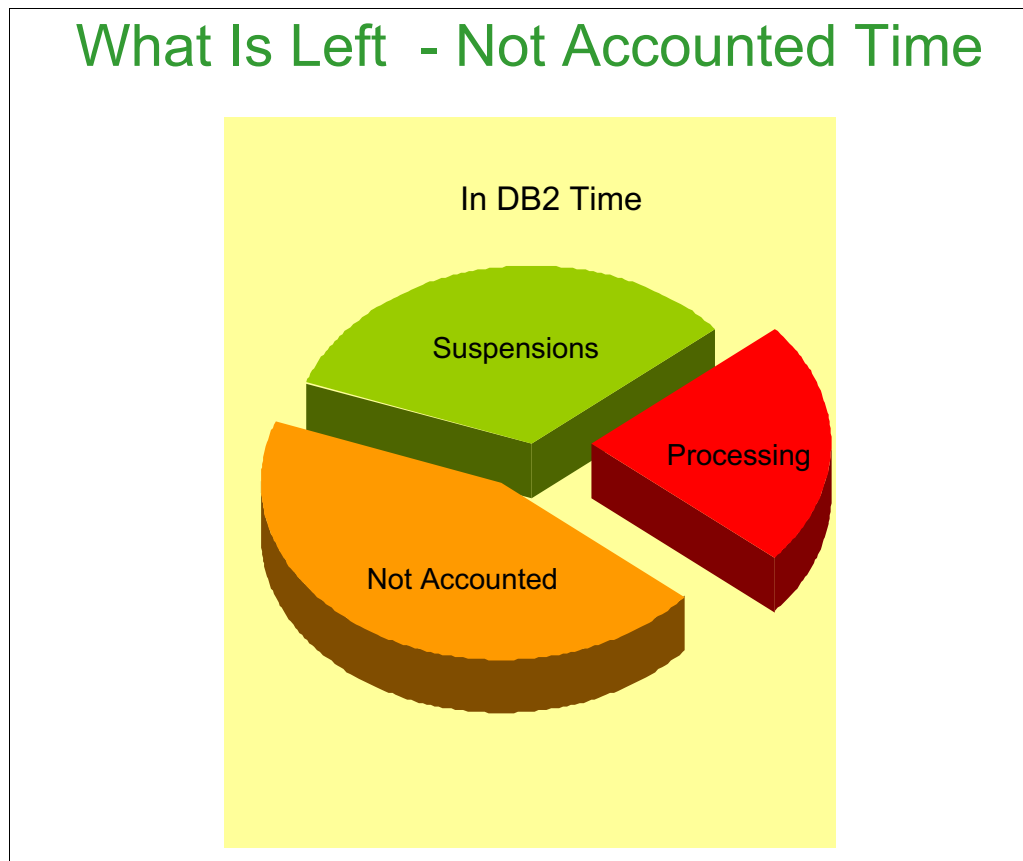


Figure 15-7 What is left

Whenever a z/OS event happens that is outside of DB2 control, DB2 has no opportunity to classify it into a proper bucket, and appears as a prolonged elapsed time without any further pointer.

Some common causes of NOT ACCOUNT time are as follows:

- ▶ Time spent waiting for a free processor in conditions of overcommitted processor resources (CPU contention).
- ▶ Incorrect WLM dispatching priorities of the service classes that DB2 or the transactions are running in. If the importance levels are too low, then the processor can reduce the amount of processor resources that are available to DB2 and the transaction. See *DB2 11 for z/OS Managing Performance*, SC19-4060, for recommendations on WLM settings for DB2 address spaces.
- ▶ Non-dedicated LPARs losing CPU. In extreme situations, DB2 can be swapped out.
- ▶ Paging. To address paging, make sure that there is enough real storage on the machine. If you are not able to add storage, investigate whether large storage consumers such as page-fixed buffer pools can be reduced. There is more information on storage concerns in Chapter 3, "System z related information" on page 29.

- ▶ Too much detailed online tracing, or problems with some vendor performance monitors. This situation is usually the primary cause of high NOT ACCOUNT time on systems that are not CPU-constrained.
- ▶ In DB2 11, the following two suspension times are reported as Class 3 suspension time instead of reporting under NOT ACCOUNT time:
 - Parallel query child and parent task synchronization (reported as PQ Synchronization)
 - Utility format write suspension (reported under Other Write I/O)

The technote at <http://www.ibm.com/support/docview.wss?uid=swg21045823> provides a comprehensive list of all of the considerations that can contribute to NOT ACCOUNT time.

Example 15-6 shows an example of high NOT ACCOUNT time.

Example 15-6 NOT ACCOUNT time

AVERAGE	APPL (CL.1)	DB2 (CL.2)
-----	-----	-----
ELAPSED TIME	3.163435	3.129203
CP CPU TIME	0.018574	0.007995
SUSPEND TIME	0.000136	0.113785
NOT ACCOUNT.	N/A	3.007423

Effect of lack of CPU resources on CL3 or CL8 timers

Note that a lack of system resources can impact not only the NOT ACCOUNT time, but the class 3 and 8 timers as well. Figure 15-8 illustrates that DB2 starts a clock when it goes into a class 3 wait; however, the clock on the wait time can only be stopped after the thread's TCB or SRB is dispatched again.

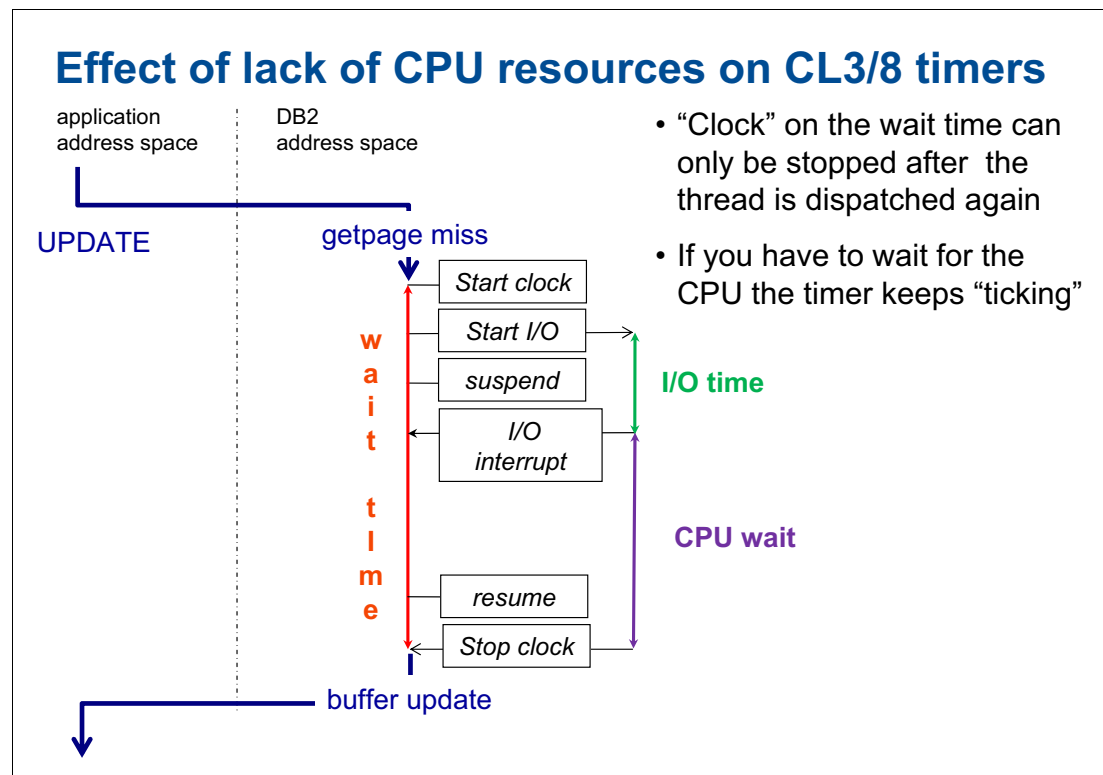


Figure 15-8 Effect of lack of CPU resources on CL3/8 timers

So when DB2 is waiting to be dispatched, the class 3 and 8 wait timers will continue to be incrementing elapsed time waits for locks or other class 3 timers. This means that when CPU and other system resources are constrained, you may not only see high NOT ACCOUNT TIME, but also higher CLASS 3 and 8 wait times.



I/O suspensions

You must monitor the average I/O elapsed time for read I/O and write I/O. A high average I/O elapsed time indicates that I/O contention is occurring. The requested I/O is accessing data from a busy data set, volume, or control unit and is continually being suspended.

For a volume or control unit level I/O problem, use the MVS performance management component to determine which component in the path is the bottleneck. If the problem is at this level, the MVS system programmers should be involved.

High average I/O elapsed times may also occur when there is heavy processor contention, without an existing I/O contention problem.

The I/O wait time is the total application wait time for DB2 I/O operations that are related to synchronous read I/O, asynchronous reads, and asynchronous writes. An unexpectedly high I/O suspension time may be associated with higher synchronous read I/Os and synchronous write I/Os.

This chapter contains the following topics:

- ▶ Synchronous I/O suspensions: Large number
- ▶ Log read suspensions
- ▶ Synchronous log write I/O suspensions
- ▶ Suspensions for reads by other agents

16.1 Synchronous I/O suspensions: Large number

The first question that needs to be answered is: Where the large waiting time comes from? Is it a large number of relatively short suspensions or a small number of long suspensions? If both the number and time are large, then it is really a major I/O problem. See Figure 16-1.

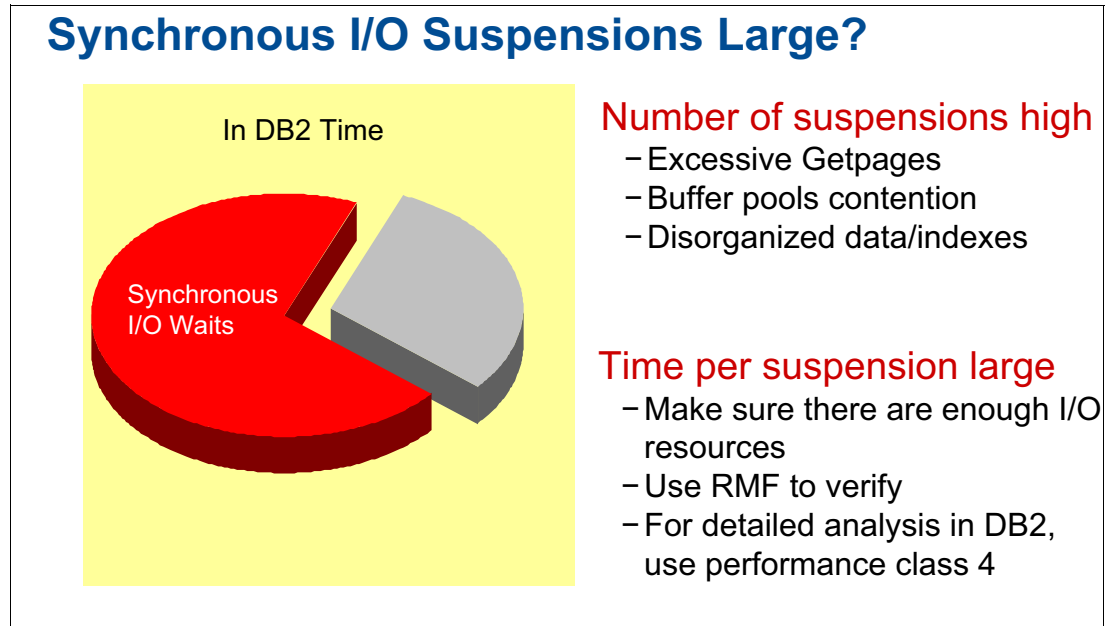


Figure 16-1 Synchronous I/O suspensions large

The most important driver of the large number of synchronous reads is the excessive number of getpages that are generated by an inefficient access path. Therefore, we first need to analyze access path and check if the optimal selection has been made. A better access path will result in less getpages and they will consequently drive fewer synchronous reads, which is a win/win situation for both CPU and buffer pools/disk performance.

The next cause of excessive number of reads could be buffer pools contention, for example:

- ▶ Too small buffer pools are causing very short page residency and the data needs to be re-read many times. The DB2 statistics provide plenty of indicators that help detect this situation. See Chapter 10.1, “Local buffer pools” on page 126 for more information on buffer pool residency time.
- ▶ The problem may not be the application that we are looking at, it could be another application that is pushing out pages that are normally in the buffer pools and our application is just the “victim.”

If the rows in data and indexes are not at their optimal position, that will cause more synchronous read activity than needed. The remedy is to reorganize data and indexes.

When the average time per suspension is large, that points to inefficiency in your disk subsystem. A typical reason is channels over utilization and DASD contention: Check data set placement or, better, use PAV feature of ESS. Another typical reason is poor performance with data replication when remotely copying data to a remote location. In some vendor implementations, even a read cache hit in the control unit will queue up behind the writes, in which case, local read cache hits will queue up behind the writes.

Example 16-1 shows an accounting trace example when DASD over utilization causes slow I/O. Look at SYNCH I/O AVG.: 0.463044.

Example 16-1 DASD over utilization causes slow I/O

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS	HIGHLIGHTS
ELAPSED TIME	6.298649	3.705151	N/P	LOCK/LATCH(DB2+IRLM)	0.000000	0	THREAD TYPE : ALLIED
NONNESTED	6.298649	3.705151	N/A	IRLM LOCK+LATCH	N/A	N/A	TERM.CONDITION: NORMAL
STORED PROC	0.000000	0.000000	N/A	DB2 LATCH	N/A	N/A	INVOKE REASON : NEW USER
UDF	0.000000	0.000000	N/A	SYNCHRON. I/O	3.704352	8	PARALLELISM : NO
TRIGGER	0.000000	0.000000	N/A	DATABASE I/O	3.704352	8	QUANTITY : 0
				LOG WRITE I/O	0.000000	0	COMMITTS : 1
CP CPU TIME	0.001479	0.000861	N/P	OTHER READ I/O	0.000000	0	ROLLBACK : 0
AGENT	0.001479	0.000861	N/A	OTHER WRTE I/O	0.000000	0	SVPT REQUESTS : 0
NONNESTED	0.001479	0.000861	N/P	SER.TASK SWITCH	0.000000	0	SVPT RELEASE : 0
STORED PROC	0.000000	0.000000	N/A	UPDATE COMMIT	0.000000	0	SVPT ROLLBACK : 0
UDF	0.000000	0.000000	N/A	OPEN/CLOSE	0.000000	0	INCREM.BINDS : 0
TRIGGER	0.000000	0.000000	N/A	SYSLGRNG REC	0.000000	0	UPDATE/COMMIT : 0.00
PAR.TASKS	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000000	0	SYNCH I/O AVG.: 0.463044
				OTHER SERVICE	0.000000	0	PROGRAMS : 2
SECP CPU	0.000000	N/A	N/A	ARC.LOG(QUIES)	0.000000	0	MAX CASCADE : 0
				LOG READ	0.000000	0	
SE CPU TIME	0.000000	0.000000	N/A	DRAIN LOCK	0.000000	0	
NONNESTED	0.000000	0.000000	N/A	CLAIM RELEASE	0.000000	0	
STORED PROC	0.000000	0.000000	N/A	PAGE LATCH	0.000000	0	
UDF	0.000000	0.000000	N/A	NOTIFY MSGS	0.000000	0	
TRIGGER	0.000000	0.000000	N/A	GLOBAL CONTENTION	0.000000	0	
				COMMIT PH1 WRITE I/O	0.000000	0	
PAR.TASKS	0.000000	0.000000	N/A	ASYNCH CF REQUESTS	0.000000	0	
				TCP/IP LOB XML	0.000000	0	
SUSPEND TIME	0.000000	3.704352	N/A	TOTAL CLASS 3	3.704352	8	
AGENT	N/A	3.704352	N/A				
PAR.TASKS	N/A	0.000000	N/A				
STORED PROC	0.000000	N/A	N/A				
UDF	0.000000	N/A	N/A				

Besides I/O subsystem issues, there can be a number of other reasons that can result in long I/O response, such as CPU contention, which means that after an I/O completes, the application is not dispatched for some time. By default, the priority of I/O is directly proportional to the priority of the initiating address space. For example, synchronous I/O has application address space priority and prefetch has DBM1 priority, which has the following results:

- ▶ Prefetches ahead of synchronous reads/writes
- ▶ Asynchronous writes ahead of synchronous reads/writes
- ▶ Asynchronous writes equal to prefetches

16.1.1 Synchronous database I/O suspensions

Synchronous I/O suspensions is the most common reason for excessive wait times and the overall elapsed time problems. Since DB2 6, the counter for synchronous log writes has been separated from the bucket and that enables better identification of the causes of the problem. See Figure 16-2.

Although the timer shows a cumulative value for both the synchronous reads and writes, you should typically assume that a large majority were synchronous reads.

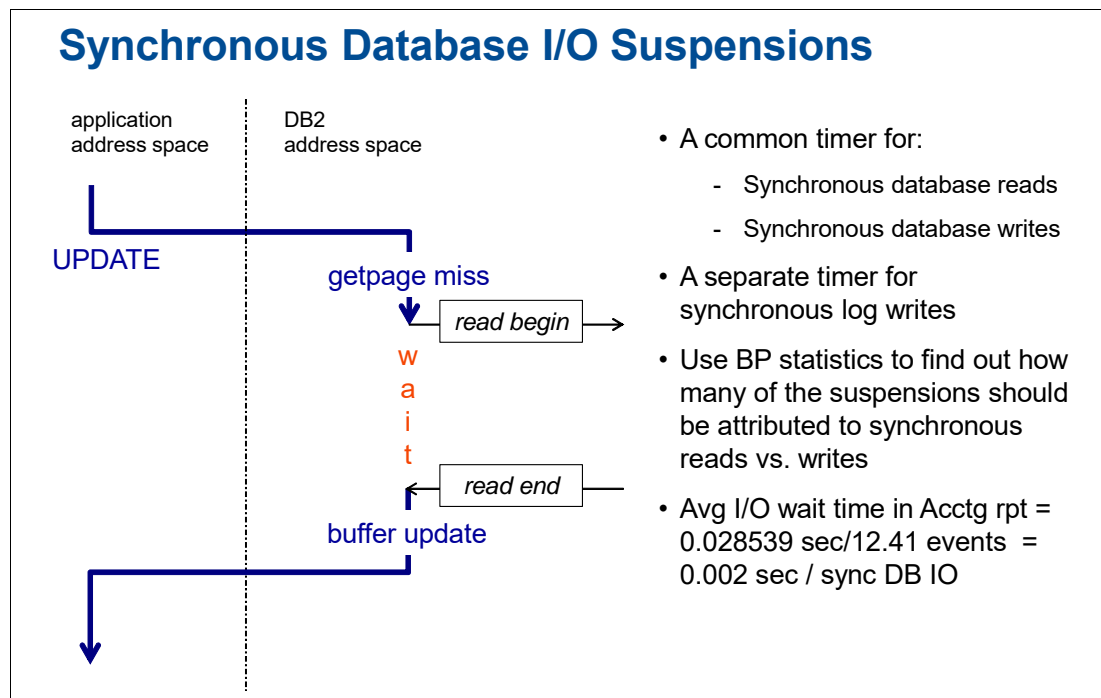


Figure 16-2 Synchronous database I/O suspensions

Synchronous buffer pool reads occur under these conditions:

- ▶ One or few consecutive pages are retrieved.
- ▶ Requested pages are not consecutive.
- ▶ Prefetch is disabled via VPSEQT=0.
- ▶ Prefetch threshold (90% of VPSIZE) is reached.
- ▶ Prefetched pages are stolen before the application gets to them.

Synchronous buffer pool writes occur under these conditions:

- ▶ Immediate Write threshold (97.5 % of VPSIZE) is reached.
- ▶ More than two system checkpoints have passed without the page being written.
- ▶ Write engine is not available.

16.1.2 Tuning the number of synchronous I/O suspensions

If many database I/O waits, check the following activities:

- ▶ Excessive getpages, which could point to an SQL statement with a bad access path. You can use Explain with Data Studio or other query analysis products to tune access path problems.
- ▶ Buffer pools contention. Sometimes other applications using the same buffer pool can overwhelm the available space in the buffer pool and cause our application to have to read in the same page multiple times, impacting the hit ratio. This can also happen when the buffer pool is too small. Still other issues with excessive I/O are apparent when you review the buffer pool statistics in the statistics report. See Chapter 10, “Buffer pools and group buffer pools” on page 125 for details.
- ▶ Disorganized data and indexes. Real-time statistics can give you an idea whether your data and indexes need reorganizing.
- ▶ Index splits. When an index page is full, a new page is acquired (either by using an existing empty (free) page, or by allocating a new page. IFCID 359 can help identify index leaf page split activity. IFCID 359 records when index page splits occur and on what indexes. Note that IFCID 359 is not included with any trace class, and that you need to start it explicitly.
- ▶ In heavy insert with minimum read activity, define the table space with zero free space and/or APPEND YES to ensure that the new rows are always inserted at the end of the table space, which minimizes the search for free space. However, this can hurt performance of multiple record scan by clustering index as the clustering is destroyed and a REORG may become necessary. This is really a trade-off between insert and retrieval performance.

16.1.3 Tuning I/O wait time

If high database I/O wait, check if there are enough I/O resources. We can use RMF to verify volumes with high activity, long suspension time and high IOSQ times or any other indicator of DASD stress. For high IOSQ time, we use PAV, for faster I/O devices, we can use SSD drives. Also check the I/O priority, as we discussed prior the I/O priority comes from the priority of the address space that generate the I/O. Here are others items to check: Control Unit cache misses, CPU contention and dispatching delays.

The use of I/O striping where appropriate even though is less effective with newer I/O devices. It is still beneficial for non-saturated I/O configuration with infrequent commit and large amount of data to be read or written in one I/O, for example, list prefetch or heavy write.

16.2 Log read suspensions

LOG READ is the accumulated wait time for: Archive LOG reads, Active LOG reads, and Active LOG prefetch reads. This time can be extremely large if the unit is not available or if a tape volume mount is required. See Figure 16-3.

In general, the number of times this kind of suspensions happens should be kept at a minimum. If it is not so, you should check why your transactions abend and got rolled back.

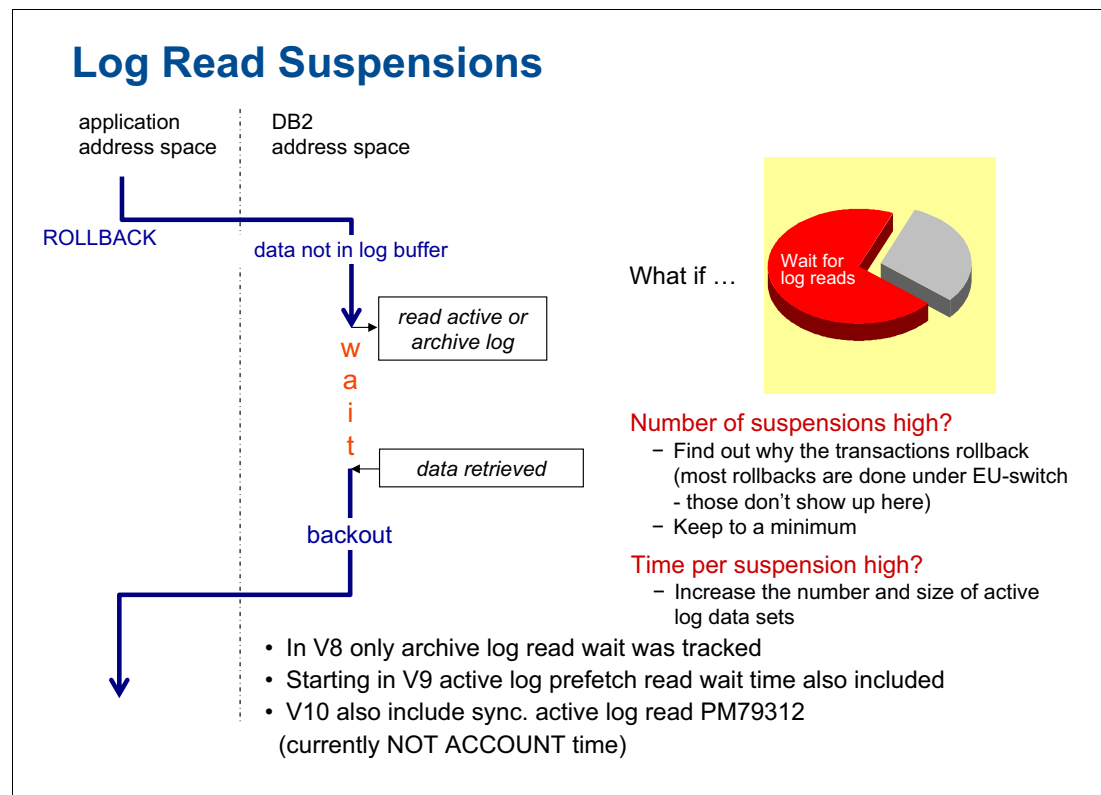


Figure 16-3 Log read suspensions

If the average time per suspension is high, you should try to avoid the situation that the data is being read from the archive logs. The only way to achieve that is to increase the number and size of the active logs, so the exposure of needed undo/redo record not being in the active log gets reduced.

IFCID 34 can be used to determine which type of log reading is performed (active, archive, prefetch, and so on) in case it is not obvious.

Another general problem with a backout (rollback or restart) can arise from the fact that it receives very high priority (the process of reading log records is done in SRB mode by db2MSTR which normally has one of the highest priorities. That can negatively affect other concurrent tasks. Also, the necessary pages from the table are retrieved by synchronous I/O in order to restore the original data. Although the CPU cost of reading the log is charged to the db2MSTR ASID, the synchronous I/O gets attributed to the agent being backed-out and it is reflected in its accounting Class 3 and CPU counters.

16.2.1 Archive Log Mode (Quiesce) suspensions

The command **ARCHIVE LOG MODE(QUIESCE)** is used to establish a system-wide point of consistency; you can do a prior point in time recovery to this time. See Figure 16-4.

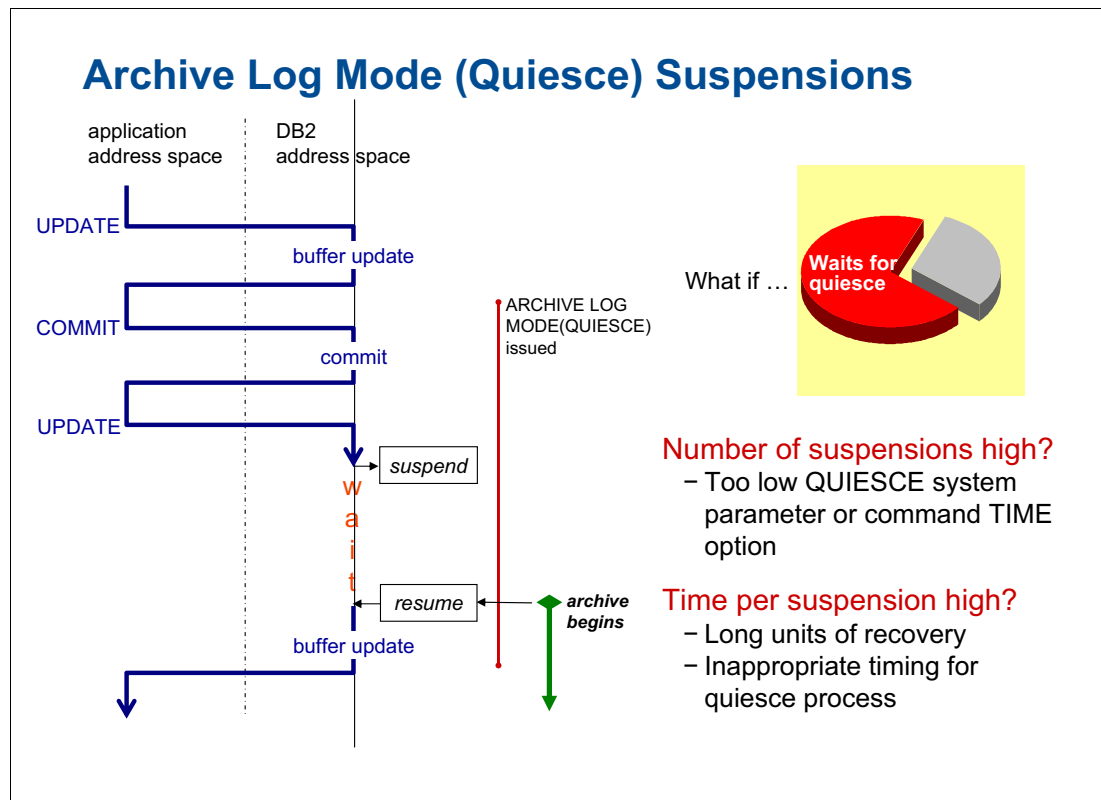


Figure 16-4 Archive Log Mode (Quiesce) suspensions

DB2 reports the following data:

- ▶ Accumulated time spent suspended
- ▶ Total number of suspensions

The problem with this command is that, if there are not well behaving (non-committing) transactions in the system when the command is issued, the **ARCHIVE LOG** command will wait until the transaction in progress finally commits.

All other transactions that would in the meantime want to get started must wait, queued behind the command. This can cause a major disruption to the overall system and needs to be monitored carefully. In some cases it is better to abandon the attempt of establishing a quiesce point rather than blocking other time critical transactions. This can be achieved by lowering the time (in seconds) that the command would wait for the concurrent transactions to commit.

Of course, if the time is too low, you are running a risk of ending up with too many suspensions due to **ARCHIVE LOG** command that was attempted (and failed) too frequently.

In a data sharing group, **ARCHIVE LOG MODE(QUIESCE)** might result in a delay before activity on all members has stopped.

16.3 Synchronous log write I/O suspensions

DB2 uses log write ahead protocol, which means that DB2 always writes to the log first before externalizing data to disk or coupling facility. Normally, from an application point of view, DB2 only forces write to active log at commit time. Therefore, any activity that triggers synchronous write I/O, to disk or coupling facility, is likely to trigger log write I/Os first; for example, in the data sharing case, an index split forces log write for non-leaf pages. See Figure 16-5.

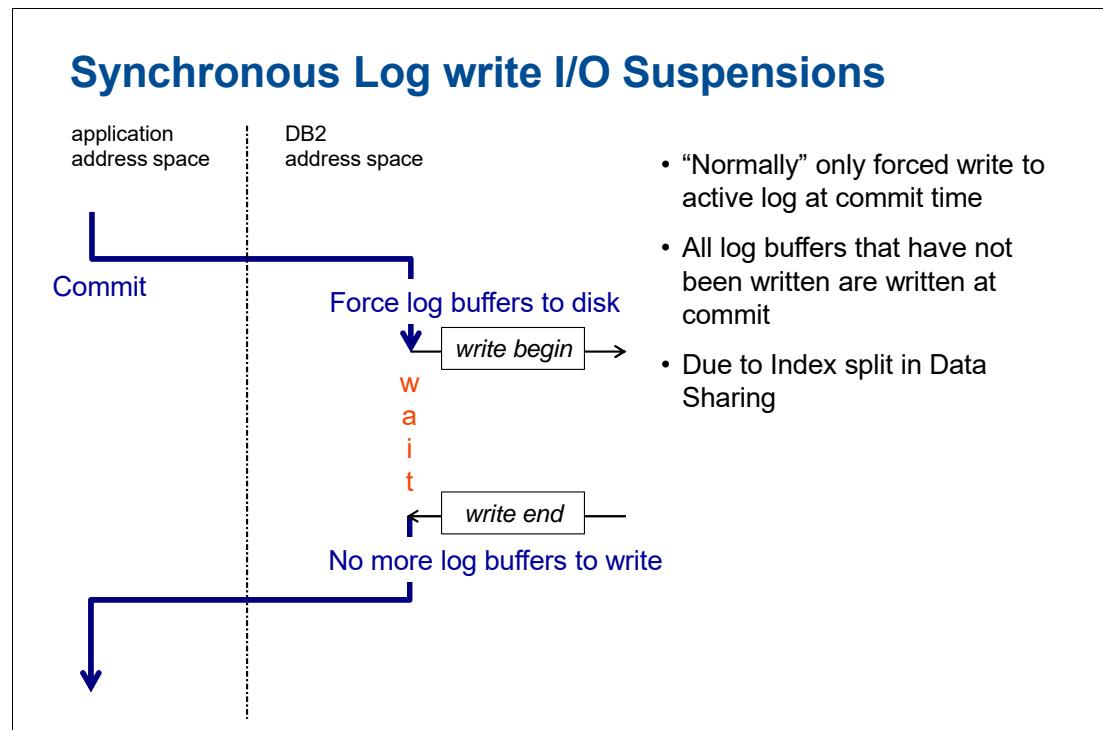


Figure 16-5 Synchronous log write I/O suspensions

Starting with DB2 V6, the counter for synchronous log writes is separated from the synchronous I/O bucket and that enables better identification of the cause of the problems.

Note that the log writes timer reports the waits due to log writes that happen in the Phase 1 of the 2-phase commit. The writes that occur in the Phase 2, single phase commits are reported elsewhere; we show this later.

What triggers synchronous log write I/Os

In data sharing environment, the reasons for forced log writes are as follows:

- ▶ An index split due to heavy Inserts forces log write for non-leaf pages.
- ▶ Heavy delete causes index tree structure modification that forces log writes.
- ▶ Inserting to a new page to GBP
- ▶ Identity columns or sequences without the cache option.

The log writes timer also reports the waits due to log writes that happen in the Phase 1 of the 2-phase commit. The writes that occur in the Phase 2, single phase commits, or read only commits are reported as Update/Commit under Service task suspension. The reasons for DB2 triggering synchronous log write I/O are as follows:

- ▶ For transactions that use 2-phase commit, Phase 1 of 2-phase commit
- ▶ Synchronous write (to disk or GBP):
 - Log write ahead protocol
- ▶ Running out of log output buffers
- ▶ Identity columns or sequences that are using NO CACHE, or low cache value, or ORDER is used, in data sharing
- ▶ Index split in data sharing:
 - Reduce index page splits as follows:
 - Use non-zero FREESPACE, FREEPAGE, PCTFREE, for random insert
 - Use PCTFREE 0 for sequential insert
 - Larger index page size in DB2 9, which has a trade-off with possibly higher index page latch contention and P-lock contention
 - Asymmetric index page split in DB2 9
- ▶ P-lock negotiation
- ▶ IMMEDIATE(YES) for GBP-dependent objects
- ▶ If GBP-dependent, when taking a completely empty index page off the index chain

If forced log writes, triggered by taking a completely empty index page off the IX chain for GBP-dependent objects, is really hurting your business when during massive delete operations, for example, check with IBM service to see if PM43695 and PM56537 can help you. This APAR, for DB2 9 and 10, provides a serviceability enhancement for index pseudo empty page cleanup logic. The change only applies to group buffer pool dependent non-partitioned indexes. DB2 11 implements the pseudo deleted cleanup asynchronous engines.

To improve long logging I/O suspensions, you can use the same techniques as for database I/Os. However, it is important to point out that good logging I/O performance is absolutely critical for the performance of the overall DB2 system, much more than database I/O performance.

In DB2 9, when a log CI had to be written a second time to disk, because the CI was not “full” the first time, DB2 would write to logcopy1 and wait for I/O completion and then write to logcopy2 and wait for completion. In DB2 10, these writes occur in parallel, which improves the performance.

DB2 10 delivery up to 40% reduction in dual log I/O wait because of parallel log write I/Os.

16.4 Suspensions for reads by other agents

Threads can also be suspended for sequential prefetch, list prefetch, dynamic prefetch, or synchronous reads performed by other agents. Figure 16-6 illustrates this.

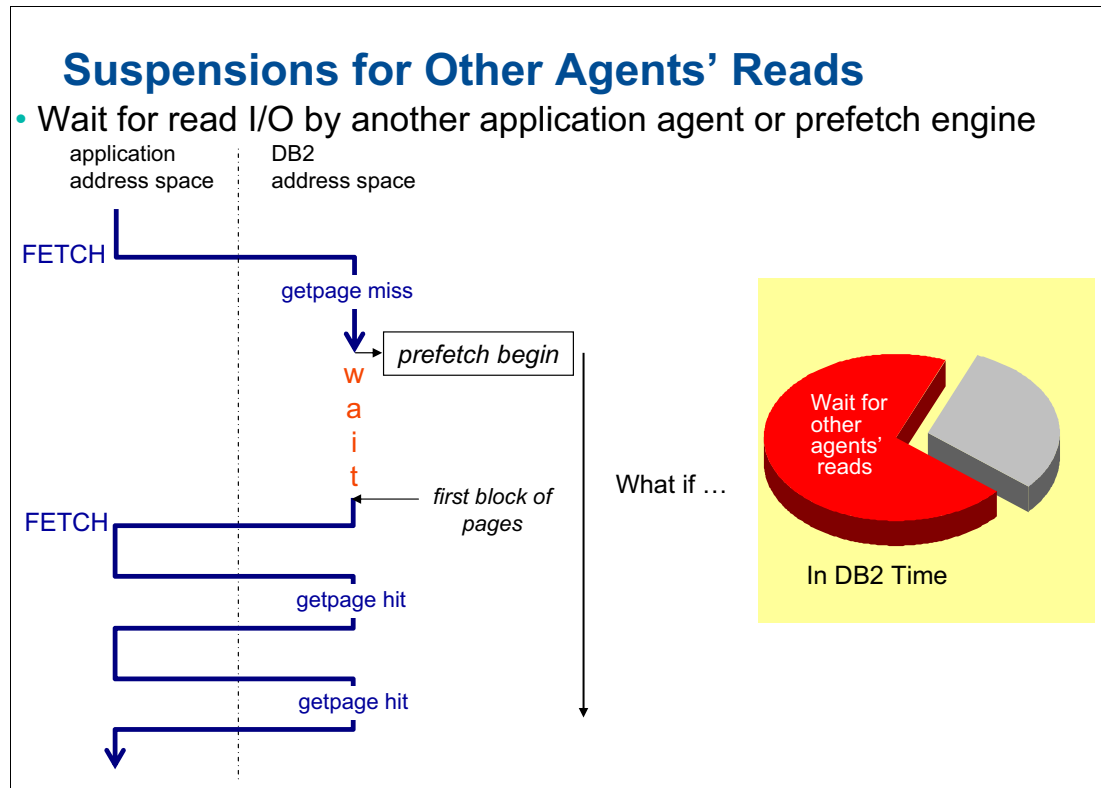


Figure 16-6 Suspensions for reads by other agents

A wait for any type of prefetch operation does not need to be caused by the subject agent itself; some other agent might have requested the operation. The wait time is counted in both agents. The numbers of various kinds of prefetches are recorded in the buffer pools statistics block and it is used only as an orientation, as a prefetch request does not necessarily result in a read operation.

Example 16-2 shows high suspension for reads by other agents, as reported on OMEGAMON PE.

Example 16-2 High suspension for others agents reads

CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT
-----	-----	-----
OTHER READ I/O	33.000014	9.01

In DB2 10, prefetch and deferred write processing are redirected to zIIP. If the workload has high prefetch activity and there are not enough zIIPs configured, you may see high other read I/O suspension time due to zIIP processing scheduling delays. Check for high APPL% IIPCP value for the DBM1 address space WLM reporting class in the RMF Workload activity report. If the APPL% IIPCP value is high, you may consider adding an additional zIIP engine.

With DB2 11, Utility format write suspensions are reported under "Other Write I/O" suspension. Prior to DB2 11, the Utility format write suspension time was under not accounted time.

16.5 Suspensions for writes by other agents

When the write I/O is in progress for a page, the page is not allowed to be used by another application. The update transaction will be suspended until the write I/O is completed. This wait event and wait times are captured either in the Other Write I/O or in the Page Latch category; the first updating transaction wait time is captured in the former. All other subsequent update transactions' wait time are captured as part of the latter. Consequently, if the number of these suspensions is high, check the system checkpoint frequency or very low buffer pool deferred write threshold (DWQT or VDWQT) or very small buffer pool. In general, checkpoint frequency should not be used for optimizing write performance, that should be controlled via deferred write thresholds.

If the average suspension time per event is high the most common reasons are DASD contention and channel over-utilization. You should also ensure that the disk subsystem features such as nonvolatile storage and Fast Write are turned on for the disk with a high write performance expectation. Figure 16-7 shows suspensions for writes by other agents.

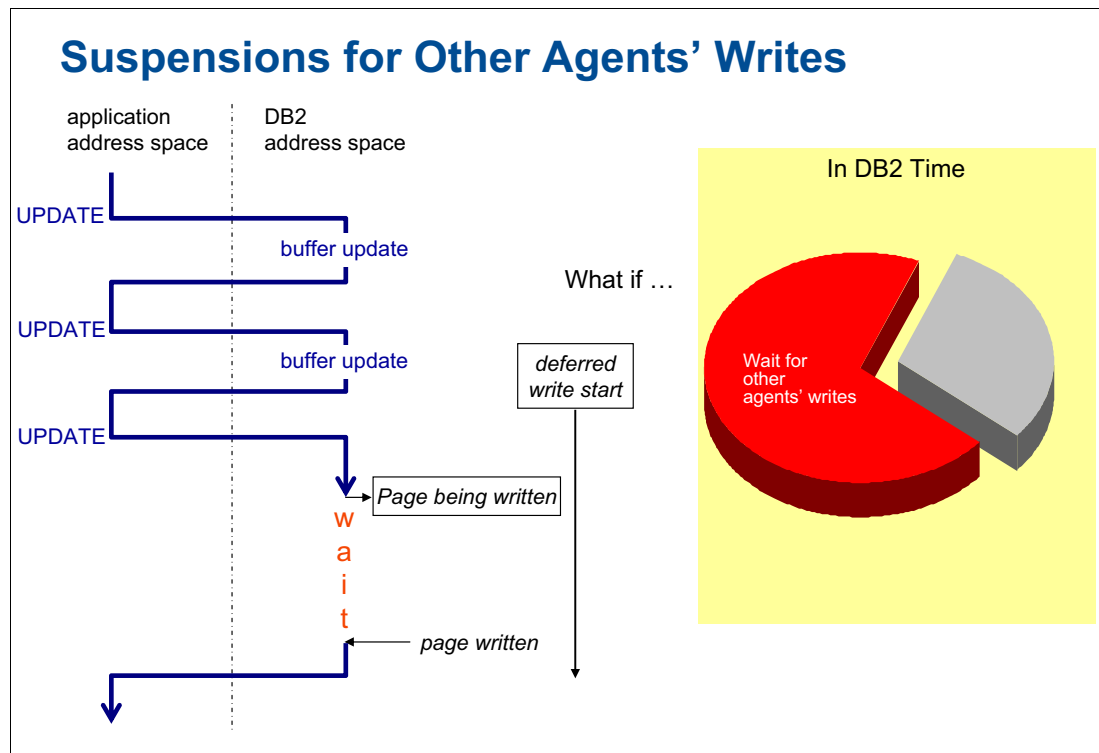


Figure 16-7 Suspensions for writes by other agents

Example 16-3 shows high suspension for writes by other agents, as reported on OMEGAMON PE.

Example 16-3 High suspension for others agents write

CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT
-----	-----	-----
OTHER WRTE I/O	0.030097	10.01

Note that other write I/O wait is wait for write I/O by another application agent or write engine.



Locking, latching, and buffer pool accounting counters

In this chapter, we discuss serialization counters related to suspensions caused by locks and latches. We also discuss buffer pool counters as caused by the application. We include data sharing information.

This chapter contains the following topics:

- ▶ Lock and latch
- ▶ Locks
- ▶ Latches
- ▶ Buffer pool information in accounting at plan level
- ▶ Group buffer pool information

17.1 Lock and latch

The data in a DB2 subsystem can be accessed by many different users at the same time. To protect the integrity of the data while maximizing the number of concurrent users and minimizing the overall cost of executing, DB2 uses a number of locking mechanisms, such as these:

- ▶ Transaction locks: They are used with SQL statements to indicate the type of access a user has or needs on a database resource.
- ▶ Restrictive states: They prevent resources, particularly table spaces and index spaces, from being accessed when DB2 cannot ensure the integrity or recoverability of the data or when a utility program is processing the space. Some restrictive states are CHKP (check pending), COPY (copy pending), UTUT (utility exclusive), UTRO (utility read only), and STOP.
- ▶ Claims and drains: They are a type of lock used by DB2 utilities and commands to indicate the type of access they need to a database resource.
- ▶ Latches: They are a type of memory lock used to ensure the integrity of pages in memory and the buffer pool.

Transaction locks are related to some type of problems, such as these:

- ▶ Timeout: An application process is said to time out when it is terminated because it has been suspended for longer than a preset interval. DB2 terminates the process, issues two messages to the console, and returns SQLCODE -911 or -913 to the process if statistics trace class 3 is active, DB2 writes a trace record with IFCID 196.
- ▶ Deadlock: A deadlock occurs when two or more application processes each hold locks on resources that the others need and without which they cannot proceed. After a preset time interval, the value of DEADLOCK TIME, DB2 can roll back the current unit of work for one of the processes or request a process to terminate. That frees the locks and allows the remaining processes to continue. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0172.
- ▶ Lock escalation: Locks are taken on pages if LOCKSIZE is PAGE or on rows if LOCKSIZE is ROW. When the maximum number of locks per table space (LOCKMAX) is reached, locks escalate to a table lock for tables in a segmented table space without partitions, or to a table space lock for tables in a non-segmented table space.
- ▶ Long lock suspension times: An application process is suspended when it requests a lock that is already held by another application process and cannot be shared. The suspended process temporarily stops running and it resumes running when all processes that hold the conflicting lock release it.
- ▶ Excessive number of locks acquired: This occurs when application request an excessive number of locks bigger than what the application characteristics would dictate.

While a synchronous I/O request always causes a suspension of the requesting agent, a lock request causes a suspension if it addresses a resource already owned by another process with incompatible lock attributes. If the lock becomes available before the application times out or deadlocks, processing continues. See Figure 17-1.

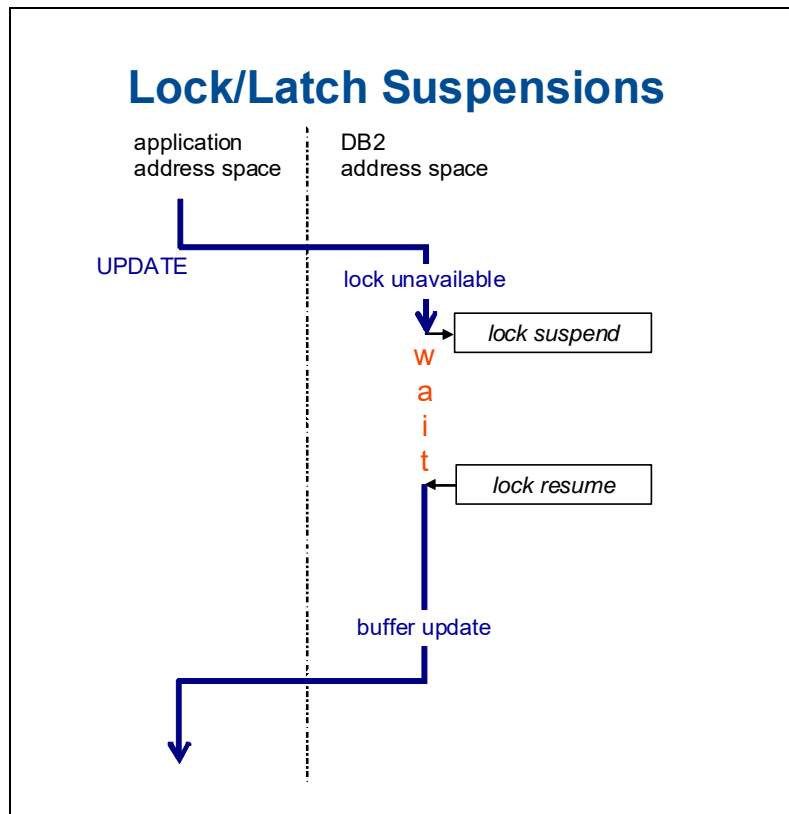


Figure 17-1 Lock/latch suspensions

17.2 Locks

The tuning actions differ depending on whether the excessive waits are caused by the large number of lock suspensions or along average suspension time or both. The most common reason for a large number of lock suspensions is the application design which does not take into account that the application needs to run in parallel, with many concurrent users and with many other applications in the same system that might be operating on the same tables.

From the data base design point of view, you may need to consider using row level locking if the larger footprint table space, table, page level locking is the culprit for the large number of contentions. Another reason could be concurrent DDL, especially if your application is bound with `RELEASE(DEALLOCATE)`.

If the average time per suspension is high you need to check for presence of long running, not committing concurrent transactions also check for concurrent DDL special when `RELEASE(DEALLOCATE)` is been used. For detail analysis, use performance trace classes 6 and 7.

17.2.1 Locking information in accounting at plan level

Example 17-1 shows locking information in accounting reports at the plan level.

Example 17-1 Locking info in accounting at plan level

LOCKING	AVERAGE	TOTAL
-----	-----	-----
TIMEOUTS	0.00	0
DEADLOCKS	0.00	0
ESCAL. (SHARED)	0.00	0
ESCAL. (EXCLUS)	1.00	4
MAX PG/ROW LOCKS HELD	5004.00	5004
LOCK REQUEST	5042.00	20168
UNLOCK REQUEST	34.00	136
QUERY REQUEST	0.00	0
CHANGE REQUEST	3.00	12
OTHER REQUEST	0.00	0
TOTAL SUSPENSIONS	1.75	7
LOCK SUSPENSIONS	0.00	0
IRLM LATCH SUSPENS.	1.75	7
OTHER SUSPENS.	0.00	0

The TIMEOUTS field shows the number of times lock suspension that resulted in a timeout. DEADLOCKS shows the number of times lock suspension that resulted in an unresolved contention for the use of a resource. This happens when two or more application processes each hold locks on resources that the others need and without it, they cannot proceed.

ESCAL.(SHARED/EXCLUSIVE) shows the number of times the LOCKS PER TABLE(SPACE) parameter on the panel DSNTIPJ was exceeded and the lock size was promoted from a page lock to a table space lock for this thread. For example, an application updates most pages in a table without issuing commits. MAX PG/ROW LOCKS HELD from accounting trace is a useful indicator of commit frequency, which in general, try to issue commit to keep max. locks held below 100.

DB2 uses ISO(CS) CURRENTDATA(NO) as default now. it gives DB2 much greater opportunity for avoiding locks. DB2 can test to see if a row or page has committed data on it. If it has, DB2 does not have to obtain a lock on the data at all. Unlocked data is returned to the application, and the data can be changed while the cursor is positioned on the row. (For SELECT statements in which no cursor is used, such as those that return a single row, a lock is not held on the row unless you specify WITH RS or WITH RR on the statement.). To take the best advantage of this method of avoiding locks, make sure all applications that are accessing data concurrently issue COMMIT statements frequently.

A good lock avoidance can significantly improve performance. A quick check in accounting data can provide some clues as to whether or not lock avoidance is fine. If you are using ISO(CS) and lock avoidance is not working, it means that for each page/row you access, you have to lock, process it, and unlock it when you move to the next page. So if there is a high ratio of unlock versus lock requests (lock requests can be readers and updates), it is a good indication that you have to lock the pages/rows before processing them, which means DB2 is unable to successfully use the lock avoidance mechanisms, for example, 50 Lock and 40 Unlock requests per commit.

17.2.2 Data sharing locking

The internal resource lock manager (IRLM) is DB2's lock manager and assumes responsibility for global locking in a data sharing environment. Each IRLM in a data sharing group continues to manage local locks within the DB2 member. When IRLM needs to acquire a global lock on a resource, such as a table space partition or a page of a table, it uses the CF lock structure to acquire the lock.

IRLM applies a hashing algorithm to the resource that needs to be locked. The result is the hash class for the resource. IRLM then issues a lock request to acquire the lock for that hash class. The lock structure in the CF determines if that hash class is available. If so, the lock request can be granted, and only this interaction takes place. If not, a conflict exists, or lock contention, then one IRLM has to communicate with other IRLMs to resolve the conflict.

Note that when another DB2 member needs a lock on a resource, its IRLM issues a lock request for the corresponding hash class to the lock structure. As long as the lock requests can be granted, no messaging between IRLMs is required. It means that most lock requests can be granted without contention.

DB2 and IRLM use two kinds of locks in a data sharing environment, L-locks (logical locks) and P-locks (physical locks):

- ▶ L-locks, are locks held by transactions. These are the traditional locks DB2 has always used. In a data sharing environment, the IRLMs use L-locks to manage concurrency across the members of the data sharing group.
- ▶ P-locks, apply only to data sharing. P-locks are part of the process to manage buffer coherency. For example, P-locks on data sets, such as table space partitions or index spaces, are used to determine when there is inter-DB2 read/write interest in the data set. Once there is inter-DB2 read/write interest in a data set, DB2 uses group buffer pools for the data set.

Example 17-2 shows an example of OMEGAMON PE data sharing locking section report

Example 17-2 Data Sharing Locking Section

DATA SHARING	AVERAGE	TOTAL
-----	-----	-----
GLOBAL CONT RATE(%)	0.48	N/A
FALSE CONT RATE(%)	0.00	N/A
P/L-LOCKS XES(%)	43.83	N/A
LOCK REQ - PLOCKS	0.67	270412
UNLOCK REQ - PLOCKS	0.63	252966
CHANGE REQ - PLOCKS	0.01	2550
LOCK REQ - XES	5.23	2101826
UNLOCK REQ - XES	0.89	358098
CHANGE REQ - XES	0.04	16744
SUSPENDS - IRLM	0.03	11906
SUSPENDS - XES	0.00	0
CONVERSIONS- XES	0.02	7204
FALSE CONTENTIONS	0.00	0
INCOMPATIBLE LOCKS	0.00	0
NOTIFY MSGS SENT	0.01	2278

In the foregoing report, we can look at these areas:

- ▶ Global Contention should be less than 3-5% of XES IRLM Requests - $(IRLM + XES + \text{False SUSP}) / ((IRLM + XES + \text{False SUSP}) + (LOCK+CHANGE+UNLOCK))$
- ▶ False contention should be less than 1-3% of sum of all XES IRLM Requests
- ▶ P/L LOCKS XES% - Gives an idea how many of the locks requests are propagated to XES. it shows the degree of sharing

17.2.3 Global lock suspensions

A global lock provides concurrency control within and among DB2 subsystems. The scope of the global lock is across all DB2 subsystems of a data sharing group. Conflicts on locking requests between different DB2 members of a data sharing group when those members are trying to serialize shared resources are called Global lock suspensions. See Figure 17-2.

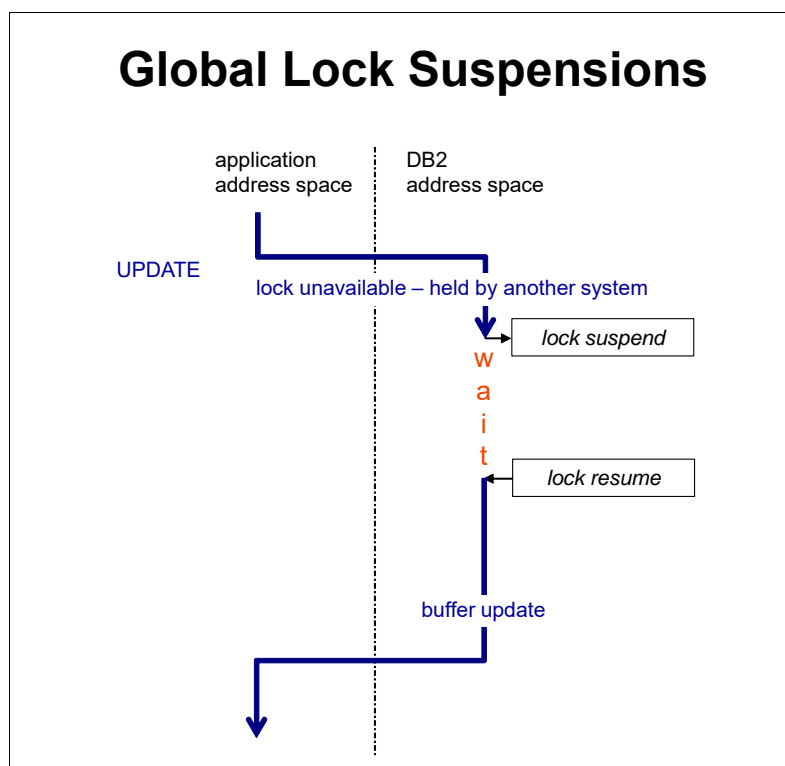


Figure 17-2 Global lock suspensions

Global lock suspensions can cause a thread to be suspended for:

- ▶ Global L-parent lock
- ▶ Global L-child (row/page) lock
- ▶ Other global L-lock
- ▶ Global pageset P-lock
- ▶ Global page P-lock (Data, IX leaf pages, SM pages)
- ▶ Other P-locks

Example 17-3 shows an example of global contention L-locks and P-locks counters.

Example 17-3 Global contention L-locks and P-locks

GLOBAL CONTENTION L-LOCKS	AV.TIME	AV.EVENT	GLOBAL CONTENTION P-LOCKS	AV.TIME	AV.EVENT
L-LOCKS	0.000372	0.06	P-LOCKS	3.001678	13.39
PARENT (DB,TS,TAB,PART)	0.000000	0.00	PAGESET/PARTITION	0.000000	0.00
CHILD (PAGE,ROW)	0.000372	0.06	PAGE	3.000167	13.10
OTHER	0.000000	0.00	OTHER	0.001511	3.28

If there is a high global lock contention for Events/Time, here are some considerations:

- ▶ Keep CF utilization below 50% for availability, response time, CPU time.
- ▶ If most time or most events are “OTHER” P-lock or L-lock, use the following commands for a short time to get detailed information:
-STA TRA (P) C(6) ...

To minimize P-lock contention/negotiation, here are some considerations:

- ▶ First find out from GBP statistics what is contributing to P-lock contention.
- ▶ In case of spacemap page P-lock and/or data page P-lock, consider using the MEMBER CLUSTER table space:
 - DB2 8/9/10 partitioned table space
 - DB2 10 UTS
 - High spacemap page latch contention possible when the latch is held waiting for data spacemap page P-lock contention to resolve.
- ▶ Consider setting VDWQT=0 to free page P-locks, but may cause “other write I/O” wait time or page latch wait time, so be careful.
- ▶ If index tree P-lock (high index splits), here are some considerations:
 - Free space tuning (PCTFREE FREEPAGE)
 - Minimize index key size especially for unique or semi-unique multi-column index key
 - V8 non-padded index if large VARCHAR(s) in index key
 - V9 bigger index page size and asymmetric page split
- ▶ If data page P-lock contention on small tables with LOCKSIZE ROW, consider using MAXROWS=1 and LOCKSIZE PAGE to “simulate” row locking and/or reduce spacemap page free space update

Figure 17-3 illustrates a global lock contention where even though part of the wait time for transaction C is waiting for a global lock held by transaction A, all suspension time is reported as local because at resume time, the lock that was released was local to this member. while transaction B had a global lock suspension because it was waiting for a lock held on another member of data sharing.

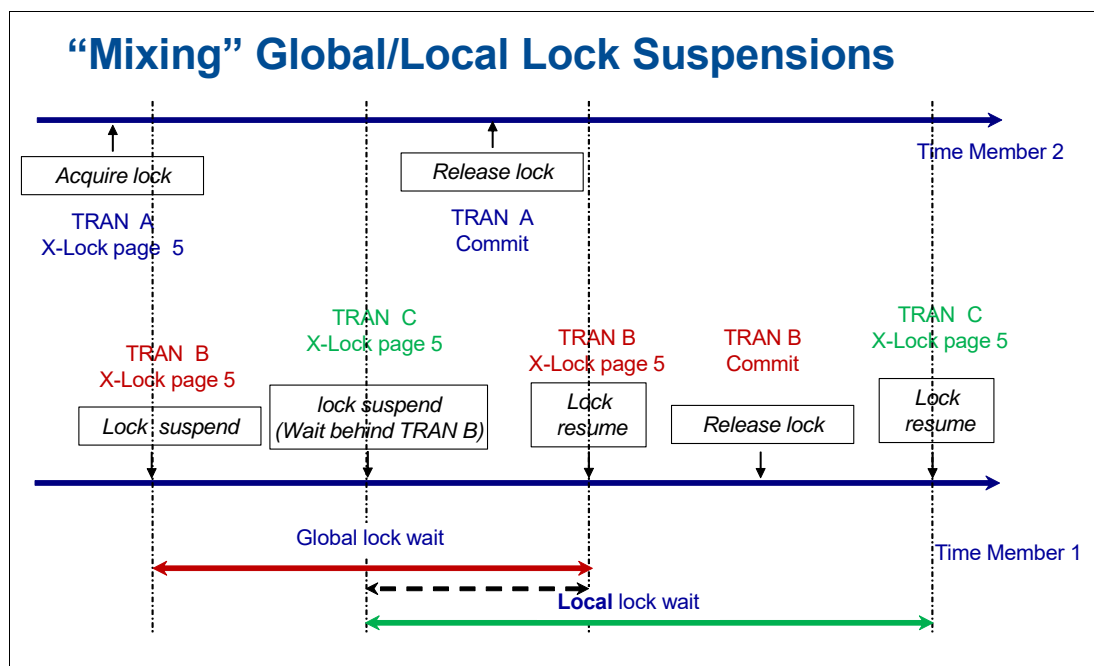


Figure 17-3 "Mixing" Global/Local Lock Suspensions

17.3 Latches

DB2 latches are acquired internally by various DB2 resource managers for short term serialization of resources. Example 17-4 shows a description of DB2 latches. it can also be found at SDSNMACS(DSNDQVLS).

Example 17-4 DB2 latches

QVLSLC01	DS	F	/* Infrequently used	*/
QVLSLC02	DS	F	/* Global authorization cache	*/
QVLSLC03	DS	F	/* DDF disconnect	*/
QVLSLC04	DS	F	/* SYSSTRING cache	*/
QVLSLC05	DS	F	/* IRLM data sharing exits	*/
*			/* or RLF	*/
QVLSLC06	DS	F	/* Data sharing index split	*/
QVLSLC07	DS	F	/* Index lotch and OBD	*/
*			/* allocation	*/
QVLSLC08	DS	F	/* Query parallelism	*/
QVLSLC09	DS	F	/* Utilities or stored	*/
*			/* procedure URIDs	*/
QVLSLC10	DS	F	/* Allied agent chain or	*/
*			/* sequence descriptors	*/
QVLSLC11	DS	F	/* DGT allocation	*/
QVLSLC12	DS	F	/* Global transaction ID table	*/
QVLSLC13	DS	F	/* Pageset operations	*/
QVLSLC14	DS	F	/* Bufferpool LRU	*/
QVLSLC15	DS	F	/* ARCHIVE LOG MODE(QUIESCE)	*/
QVLSLC16	DS	F	/* UR chain	*/
QVLSLC17	DS	F	/* RURE chain	*/
QVLSLC18	DS	F	/* DDF resynch list	*/
QVLSLC19	DS	F	/* Log write	*/

QVLSLC20 DS	F	/* System checkpoint	*/
QVLSLC21 DS	F	/* Accounting rollup	*/
QVLSLC22 DS	F	/* Internal checkpoint	*/
QVLSLC23 DS	F	/* Buffer manager	*/
QVLSLC24 DS	F	/* EDM pool or prefetch	*/
QVLSLC25 DS	F	/* Workfile allocation	*/
QVLSLC26 DS	F	/* Dynamic statement cache	*/
QVLSLC27 DS	F	/* Stored procedures or	*/
*		/* authorization cache	*/
QVLSLC28 DS	F	/* Stored procedures or	*/
*		/* authorization cache	*/
QVLSLC29 DS	F	/* Field procs and DDF	*/
*		/* transaction manager	*/
QVLSLC30 DS	F	/* Agent services	*/
QVLSLC31 DS	F	/* Storage manager	*/
QVLSLC32 DS	F	/* Storage manager	*/
QVLSLC254 DS	F	/* Index latch	*/

Before DB2 10, in order to find out the breakup of the number of number of lock and DB2 and IRLM latch suspension, we needed to look at the locking section where the IRLM lock and IRLM latch suspensions are separately listed and the internal DB2 latch suspensions are reported in statistics. Starting in DB2 10, DB2 latch suspensions are reported separately. Example 17-5 shows a DB2 latch report extracted from the OMPE statistics report.

Example 17-5 DB2 latch counters

LATCH CNT	/SECOND	/SECOND	/SECOND	/SECOND
-----	-----	-----	-----	-----
LC01-LC04	0.00	0.00	0.00	0.00
LC05-LC08	0.00	0.01	0.00	0.00
LC09-LC12	0.00	0.00	0.02	0.00
LC13-LC16	0.00	0.00	0.00	0.00
LC17-LC20	0.00	0.00	0.00	0.00
LC21-LC24	0.00	0.00	0.00	0.00
LC25-LC28	0.00	0.00	0.00	0.00
LC29-LC32	0.00	0.00	0.00	0.00
LC254	0.00			

IRLM latch is used by IRLM to serialize access within itself, for example, IRLM latches are held during the deadlock cycle. In order to find out the breakup of number of suspensions, look at the locking section where the lock and IRLM latch suspensions are separately listed. See Example 17-6.

Example 17-6 Lock Section

LOCKING ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
SUSPENSIONS (ALL)	8.00	0.00	N/C	0.09
SUSPENSIONS (LOCK ONLY)	0.00	0.00	N/C	0.00
SUSPENSIONS (IRLM LATCH)	2.00	0.00	N/C	0.02
SUSPENSIONS (OTHER)	6.00	0.00	N/C	0.07

17.3.1 Latch suspensions

The large number of internal DB2 latches in most cases points to DB2 internal problems and should be reported to the DB2 service. The exceptions are Latch classes 06, 07, 11, 14, 19, and 24, where user tuning can address the problem, for example:

- ▶ When LC06 is high, Index tree P-lock latch contention caused most likely by splits of GBP-dependent index pages, we can specify large PCTFREE for indexes that are causing the problem and reorganize them more often to reestablish the free space and reduce index splits at subsequent inserts. Also when LC06, LC07 are high, we can use a larger index page size to help.
- ▶ Updating the MAXASSIGNEDVAL column in SYSSEQUENCES to figure out the next available value for an identity column is done under internal DB2 latch Class 11, Generating Identity Column. For data sharing it is additionally done under page P-lock. Using the CACHE option should be strongly recommended for high insert rate workloads, especially in data sharing.
- ▶ LC14: For BP LRU chain, we can isolate small highly accessed tables into separate pools.
- ▶ Class 19: We can reduce the number of class 19 latch contentions by speeding up the log write activity allocating logs at the fastest devices, turning z/OS, avoiding log DASD contention, rearranging columns to minimize the number of log data written and so on.
- ▶ Class 24: It can be either EDM LRU latch or Buffer Manager latch. If latch contention trace, IFCID 51 and 56, is collected, QW0051LC/QW0056LC will show 24 for EDM LRU latch and 56 for Buffer Manager latch. If 24, the best way to fix is to use EDMBFIT DSNPARM of NO as an order of magnitude reduction in LC 24 latch contention has been observed. If 56, there is a number of cases Buffer Manager uses latching. For example, there is one latch per data set in prefetch scheduling. This has been noticeable in dynamic prefetch in star join and also in CPU parallelism with many degrees. Having more partitions can reduce this latch contention. Real Time Statistics also uses LC 56 latch.

The criteria for assessing that the number of IRLM latch suspensions is high is if that number is higher than 10% of all the IRLM requests. When the number of IRLM latch suspensions is high, check the IRLM dispatching priority, IRLM Trace on, frequent IRLM query requests, low deadlock detection cycle, and many lock suspensions.

17.3.2 Page latch suspensions

Page latch contentions happen when a page “in construction,” for example, the page that is just being written into, is attempted to be accessed; page latch suspensions are totally unrelated to DB2 internal latch contention that was discussed in the previous section. See Example 17-7.

Example 17-7 PAGE LATCH suspension

CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT
-----	-----	-----
PAGE LATCH	10.000013	30.01

Also, when a write I/O is in progress for a page, the page is not allowed to be updated. The update transaction will be suspended until the write I/O is completed. This wait event and wait time are captured either in the Other Write I/O or in the Page Latch category; the first updating transaction wait time is captured in the former. All other subsequent update transactions' wait time are captured as part of the latter.

The tuning actions include relieving space map contention via partitioning and in some specific cases using MEMBER CLUSTER table space attribute. Which is the exact cause of the page latch contentions can be found by analyzing page latch contention reports that are based on the IFCIDs 226 and 227 that report each individual page latch contention event and identify its duration and the object of contention. See Figure 17-4.

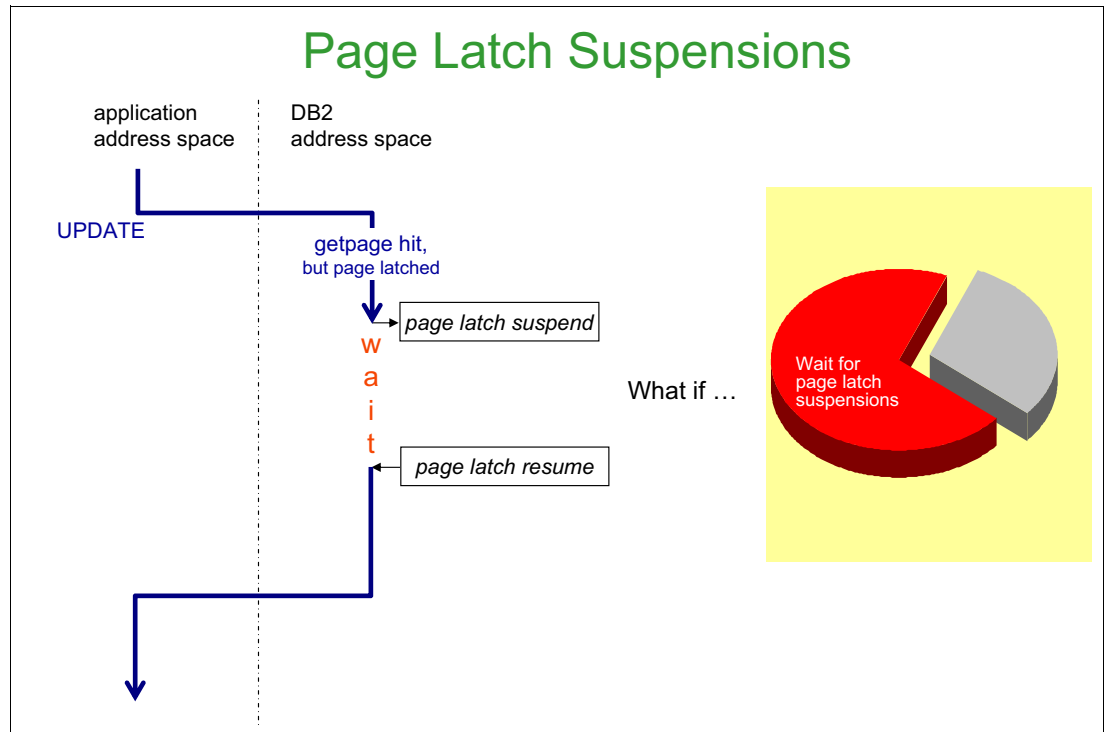


Figure 17-4 Page latch suspensions

For Page latch contention, also consider higher VDWT to possibly reduce write frequency or use faster active log and database write I/O device. If there is an issue with the spacemap page, try using TRACKMOD NO instead of default YES, set concurrent inserts to different partitions, or use classic partitioned rather than segmented. If there is an issue with the index page, try changing the index page for smaller if random key insert, bigger if sequential key insert.

Page latch contention is often a symptom of another problem, as typically only the first applications incurs the "actual wait," such as "wait for other write I/O," and all other transactions that want to access the page will now wait on the page latch that is held by the first application, the one waiting for the write I/O to complete.

In an OMEGAMON PE DB2 accounting report that combines the data of a lot of transaction executions, it is highly likely that the page latch waits will show up with higher values, as many transactions are suspended on the page latch, but only one transaction is showing wait for write I/O.

17.4 Buffer pool information in accounting at plan level

Example 17-8 shows buffer pool information in the accounting plan level report.

Example 17-8 Buffer pool information in accounting at plan level

BPO	BPOOL ACTIVITY	AVERAGE	TOTAL
-----	-----	-----	-----
BPOOL HIT RATIO (%)		92.22	N/A
GETPAGES		711.64	12605326
BUFFER UPDATES		226.48	4011708
SYNCHRONOUS WRITE		0.00	0
SYNCHRONOUS READ		49.71	880538
SEQ. PREFETCH REQS		0.03	597
LIST PREFETCH REQS		0.19	3313
DYN. PREFETCH REQS		1.11	19595
PAGES READ ASYNCHR.		5.67	100352

An increase in the number of getpages for the same amount of work, which means the same number of SQL statements, is typically an indicator that the access path has changed or object disorganized or overflow records. When going from index access to a table space scan, for example, DB2 will now have to do a getpage for each page in the table, instead of accessing only those pages that qualified after index filtering, so this will very likely result in an increase in the number of getpages. Remember that before DB2 can evaluate a predicate against a data or index, it first has to perform a getpage to make sure the page is in the VP, to allow DM, IXM, RDS to process the data on the page. Also a lot of prefetch activity can be sign of an access path problem.

Buffer pool information in accounting at plan level does not show as many buffer pools counters in DB2 accounting compared to DB2 statistics. DB2 typically increments the BUFFER UPDATES counter for each row that is changed. So if you update two rows on the same page, you are likely to see getpages 1, buffer updates 2.

17.5 Group buffer pool information

The challenge of buffer coherency, or managing changed data, lies in the fact that DB2 buffers data. DB2 attempts to keep data or index pages in the buffers for potential reuse, reducing I/O and CPU costs. In a data sharing environment it is likely that two members of a data sharing group have many pages in common in their local buffer pools. If one member changes one of these pages, the other member has a back-level page. Buffer coherency consists of ensuring that the back-level page is brought current, to reflect the change, before the second member allows applications to access the page.

DB2 allocates cache structures in the coupling facility to manage buffer coherency. These structures are called group buffer pools (GBPs). When multiple members of a data sharing group open the same table space, index space, or partition, and at least one of them opens it for writing, the data is said to be of inter-DB2 read/write interest to the members. When there is inter-DB2 read/write interest in a particular table space, index, or partition, it is dependent on the group buffer pool, or GBP-dependent.

When a transaction changes a page of data, DB2 caches that page in the group buffer pool. The coupling facility invalidates any images of that page in the buffer pools associated with the other members. Then, if a request for that same page is subsequently made by another member, that member looks for the page in the group buffer pool.

Group buffer pools consist of directory entries, to track interest in a page; and data elements, to cache the changed page for high-speed access:

- ▶ Directory entries track pages of table spaces or index spaces that are GBP-dependent. When a member reads a page of a GBP-dependent data set into its local buffer pools, it registers that fact in the GBP by creating or updating a directory entry. One directory entry can track the registered interest in a page by all the members in the group. In addition, the directory entry tracks the specific location of the page in the member's local buffers and, if the page is written there, also in the GBP. This location information is used when a page is changed and the image of that page in the other members' buffers is invalidated. See "Cross-invalidation" next.
- ▶ Data elements cache the changed pages of GBP-dependent data sets. This makes the pages available for high speed access from a member whose local image of the page is no longer valid.

There are two actions that DB2 must perform to preserve the integrity of data across the members of the data sharing group:

- ▶ Cross invalidation:

When a changed page is externalized to the GBP, the directory entry indicates which other members have an image of that page and in what buffer locations those images are. The coupling facility control code (CFCC) sends a signal to those members that have an image of the changed page to invalidate that page in those members' buffers. This message is processed on the receiving systems without an interrupt. When one of those members attempts to access the page, it detects that the image of the page in its buffer pool is invalid. The member then accesses the GBP to see if the page is still there. If it is, the page is read from the GBP to the member's local buffer pool. If the page is no longer in the GBP, the member reads it from disk.

DB2 ensures changed pages of a GBP-dependent data set are externalized to the GBP at commit. (It is also possible the page will be externalized before commit if local buffer thresholds are hit.) DB2 data sharing uses store-in caching to the CF, which means that DB2 assumes that a page written to the GBP is recoverable and the local buffer containing the changed page is stealable. One implication of store-in caching is that the changed pages occupying the data elements must eventually be written to disk. This process is called castout and is described next.

- ▶ Castout:

Periodically, DB2 must write changed pages from the primary group buffer pool to disk. This process is called *castout*. Typically the responsibility of castout gets assigned in turn to every member of the group, with the castout work getting eventually spread evenly across all members. Each time, the member, called a castout owner, that is responsible for casting out the changed data uses its own address space because no direct connection exists from a coupling facility to disk. The data passes through a private buffer, not through the DB2 buffer pools.

The GBP tuning is mostly done via statistics information and not by accounting reports, but on the report shown in Example 17-9, we can check the GBP hit/miss ratio for application, the application page P-lock activity (number of Page P-lock requests, number of page P-lock suspensions, type of page P-lock SM and data index leaf).

Example 17-9 Global buffer pool information

GROUP BPO	AVERAGE	TOTAL
-----	-----	-----
GBP-DEPEND GETPAGES	14707.80	2191462
READ(XI)-DATA RETUR	0.32	47
READ(XI)-NO DATA RT	0.00	0
READ(NF)-DATA RETUR	101.45	15116
READ(NF)-NO DATA RT	313.67	46737
PREFETCH PAGES READ	9.48	1413
CLEAN PAGES WRITTEN	0.00	0
UNREGISTER PAGE	19.46	2899
ASYNCH GBP REQUESTS	9.31	1387
EXPLICIT X-INVALID	0.00	0
ASYNCH SEC-GBP REQ	0.00	0
PG P-LOCK LOCK REQ	497.81	74174
SPACE MAP PAGES	17.58	2620
DATA PAGES	0.80	119
INDEX LEAF PAGES	479.43	71435
PG P-LOCK UNLOCK REQ	146.15	21776
PG P-LOCK LOCK SUSP	1.17	175
SPACE MAP PAGES	0.58	87
DATA PAGES	0.05	8
INDEX LEAF PAGES	0.54	80
WRITE AND REGISTER	176.83	26347
WRITE & REGISTER MULT	17.73	2642
CHANGED PAGES WRITTEN	304.57	45381

Note: There is no GBP information at the package level.



Service task suspension

This chapter continues our look into class 3 wait times with the service task suspension, which is the accumulated wait time from switching synchronous execution units.

This chapter contains the following topics:

- ▶ Synchronous execution unit switch suspensions
- ▶ Large suspensions for synchronous EU switches
- ▶ How to find which service task is invoked

18.1 Synchronous execution unit switch suspensions

Some operations have to be performed under a different execution unit (EU) than the one that is used by the thread. See Figure 18-1.

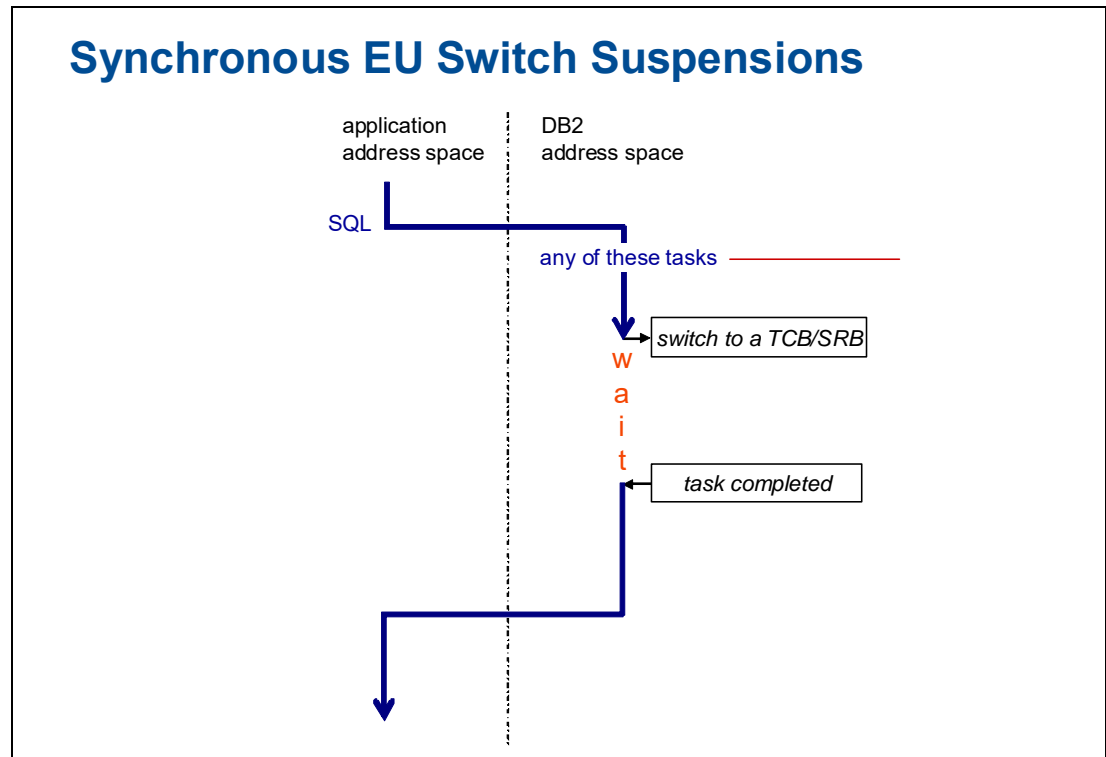


Figure 18-1 Synchronous EU Switch Suspensions

If the transaction has to wait for such an operation, this is called a synchronous execution unit switch, and such an event and its time are reported in a number of DB2 class 3 suspension counters.

In the OMEGAMON PE accounting report, the total of this information is reported in the SER.TASK SWTCH field. The field is the total of the five fields that follow it. See the lines in bold in Example 18-1.

Example 18-1 Execution unit switch class 3 timers

CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS
LOCK/LATCH(DB2+IRLM)	3.617687	5099
IRLM LOCK+LATCH	3.189665	670
DB2 LATCH	0.428021	4429
SYNCHRON. I/O	58.548574	35577
DATABASE I/O	14.743293	14783
LOG WRITE I/O	43.805281	20794
OTHER READ I/O	46.796096	8653
OTHER WRTE I/O	3.691792	11293
SER.TASK SWTCH	10:51.323655	28316
UPDATE COMMIT	1:00.608372	6227
OPEN/CLOSE	42.095973	2016
SYSLGRNG REC	16.326233	12039

EXT/DEL/DEF	8:47.463937	6018
OTHER SERVICE	4.829141	2016
ARC.LOG(QUIES)	0.000000	0
LOG READ	0.000000	0
DRAIN LOCK	0.067752	65
CLAIM RELEASE	0.000000	0
PAGE LATCH	25.765936	2637
NOTIFY MSGS	15.199547	8610
GLOBAL CONTENTION	1.467414	958
COMMIT PH1 WRITE I/O	0.000000	0
ASYNCH CF REQUESTS	13.199188	46676
TCP/IP LOB XML	0.000000	0
ACCELERATOR	0.000000	0
AUTONOMOUS PROCEDURE	0.000000	0
PQ SYNCHRONIZATION	0.000000	0
TOTAL CLASS 3	13:39.677641	147883

There is a separate counter for these items:

► **UPDATE COMMIT:**

Update commit is an application commit that follows an open unit of recovery, meaning that there was at least one database changing operation involved: insert, update or delete. (Read-only commits are not included in this counter). Update commit also includes the close and delete work done by DROP table space or index if the object is open.

► **OPEN/CLOSE:**

Open and close of data sets

► **SYSLGRNG REC:**

The recording in SYSLGRNG and down-level detection

► **EXT/DEL/DEF:**

Actions by data space manager tasks such as define, extend, delete, or reset data set. Note that data set define or delete is only recorded here if it is stogroup-managed.

► **OTHER SERVICE:**

Other services tasks such as these:

- VSAM catalog update
- Parallel query synchronization prior to DB2 11.

In DB2 11, time for parallel query synchronization is reported as a separate PQ SYNCHRONIZATION Class 3 suspension field.

A high value in “Other Service Tasks” can also occur when using DDF via a TCP/IP connection and the DB2 system you are looking at is a DRDA requester. In that case, the value is often large because it includes the time the requesting thread waited for responses from the server. (When an SNA connection is used, this type of wait time shows up as NOT ACCOUNT time.) You can then look into the “distributed activity” section of the accounting report to find the DB2 application requestor elapsed time. If the “other service task switch time” is similar to the DB2 application requestor elapsed time, the requestor side has no issues, and you need to look at the application server side to see how this time is used.

18.2 Large suspensions for synchronous EU switches

For an initial analysis of long service task time, see Figure 18-2.

Most of the commit-related waits are caused by log writing, so when you see long suspensions for synchronous EU switches, first check the log activity and find out if there are some inefficiencies. As indicated in Chapter 7, “Log activity” on page 83, the logs should be allocated to the fastest disk subsystem available. In extreme cases, consider striping for these data sets.

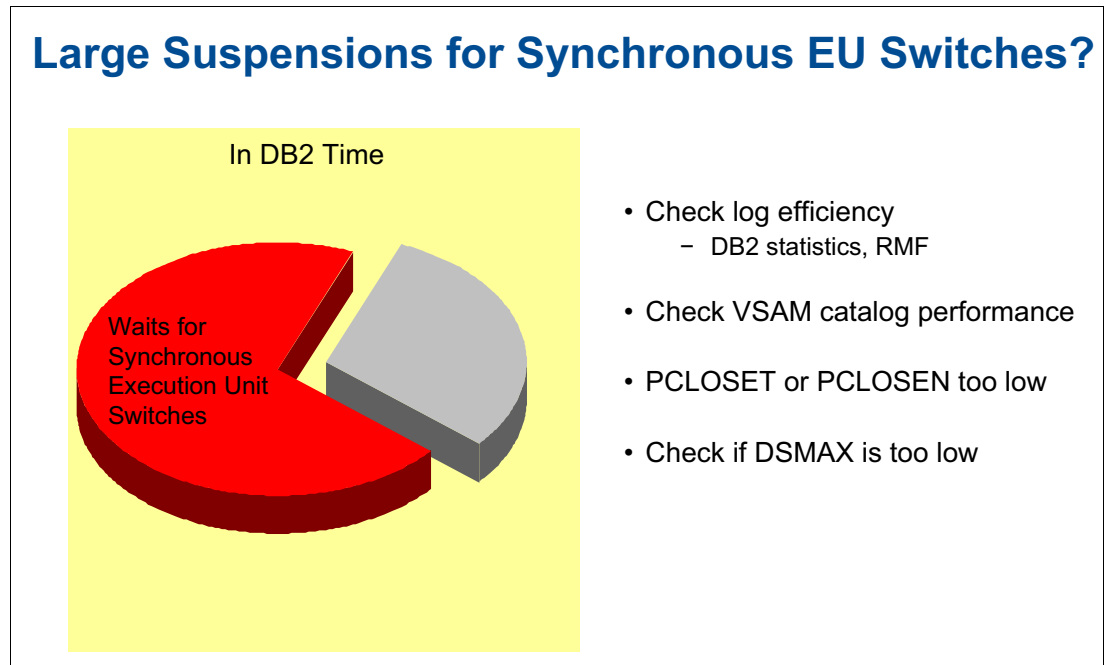


Figure 18-2 Large Suspensions for Synchronous EU Switches

In general, when data space manager activity is slow, the make sure that VSAM catalog performance is ok. You should check if there are any problems with the volumes and DASD paths that contain the VSAM catalog.

You can collect IFCIDs 258 and 92 to find more information. IFCID 258 captures data set extends (and identifies the object of extension) and IFCID 92 reports which AMS statement (for example, DEFINE/DELETE CLUSTER) has been issued on behalf of DB2.

18.2.1 Service task wait: Commit related timers

Some of the work for commit is done under the application TCB, and some is done under DB2 MSTR address space. For some components the application gets suspended and waits. Read-only commit, single-phase commit and two-phase commit have different time distributions. Figure 18-3 shows service task wait times for the various types of commit.

Service Task Wait - Commit Related Timers

Class 3 Timer (Allied Threads)	Read-only commit	Single- phase commit	Two-phase commit
Wait for log writes	no	no ¹	yes ²
Wait for synchronous EU switch for update commit	no	yes	yes
Wait for force-at- commit database writes	no	no ¹	yes ³

1 Included in wait for synchronous EU switch for 'update' commit.

2 These are log writes that happen in phase 1 of two-phase commit.

Log writes that happen in phase 2 are included in wait for synchronous EU switch for 'update' commit.

3 Applies to LOBs with LOG(NO) only. These database writes happen in phase 1 of two-phase commit.

Figure 18-3 Service Task Wait - Commit Related Timers

These are the main components of commit processing:

- ▶ Synchronous log writes
- ▶ Cursors, locking, clean-up
- ▶ Force-at-commit synchronous database writes, or GBP writes in data sharing

Force-at-commit database writes happen for GBP-dependent page sets and are normally writes to the GBPs, except for GBPCACHE NONE or LOBs with LOG(NO) when they go directly to disk).

DB2 does not record any wait time for read-only commit, because that it is entirely done under application's TCB.

18.2.2 Data set open/close

Long waits for OPEN/CLOSE are rarely isolated to one transaction. If you see long waits for OPEN/CLOSE, see Chapter 6, "Data set open and close" on page 77.

IFCID 370 and 371 trace data set open/close activity, and distinguishes between SVC99 time and VSAM catalog time. Contrary to popular belief, IFCID 107 does not trace open/close activity; instead it maps data set names to DBIDs and OBIDs.

18.2.3 SYSLGRNG REC

DB2 inserts a record into SYSLGRNG on the first update to a data set after it is open. (DB2 updates this record later at data set close. Records for dropped table spaces are not deleted until the DBID/OBID is reused). A high value in SYSLGRNG REC could indicate that data sets are being opened, closed, and reopen too frequently.

SYSLGRNG entries are also updated when a page set is converted from read-write state to read-only state. When this conversion occurs for table spaces, the SYSLGRNG entry is closed and any updated pages are externalized to disk.

If you see long waits in SYSLGRNG REC, here are some considerations:

- ▶ If there are a lot of data set open/close events, look for frequently accessed data sets that are defined as CLOSE(YES), and investigate whether they can be defined as CLOSE(NO).
- ▶ Check statistics to see if DB2 is reaching DSMAX too frequently; if so, consider increasing DSMAX.
- ▶ Consider increasing RO SWITCH TIME (PCLOSET and PCLOSEN subsystem parameter) for logged data sets that are switching between read-only and read-write too frequently.
- ▶ Clean up SYSLGRNX using the MODIFY RECOVERY utility and/or REORG of SYSLGRNX

18.2.4 Data set Extend/Define/Delete

DB2 calls data space manager to handle VSAM defines, extends, and deletes. Here we list some of the reasons for suspends in this area:

- ▶ Preformatting for data set extension, which takes between 0.02 and 1 second per extend, depending on device type and allocation unit/size.
- ▶ Counting pieces of a segmented table space at open time. DB2 has 20 service tasks available to do this work.
- ▶ Data set delete and define for objects defined in a stogroup.

High EXT/DEL/DEF times can indicate problems with a load on the ICF catalog. z/OS *V1R13.0 DFSMS: Managing Catalogs*, SC26-7409-11 has a section called, "Possible causes and solutions for catalog performance problems," which you should review if you see high values here. Some examples of catalog items to check for when you see high EXT/DEL/DEF times are as follows:

- ▶ If the RMF ENQ report shows enqueues on SYSZRPL, check the value for STRNO on your DB2 ICF catalog.
- ▶ SYSZVDS/SYSVTOC should be defined in the conversion RNL.

If you need more information for EXT/DEL/DEF, you can use the following IFCIDs:

- ▶ IFCID 258, which shows the DBID/OBID for Data set Extend
- ▶ IFCID 92, which contains Define Cluster statement for Define/Delete

18.2.5 OTHER SERVICE tasks

Here are some of the most frequent service tasks that increment the OTHER SERVICE timer and counter:

- ▶ Parallel query cleanup. Normally DB2 does not consider queries for parallelism if their estimated cost is less than 120 ms. You may see long waits in OTHER SERVICE if there are many executions of short-running queries using parallelism. If this is the case, consider increasing the ZPARM parameter SPRMPH to increase the minimum estimated cost of the queries that are considered for parallelism. This will eliminate the cleanup for the shorter queries. Keep in mind it may also increase elapsed time, since the queries will no longer be parallel.

To find out which of the service task switches occurred, you can look up the correlation name in the “System-Agent Correlation Identifiers” table in the *DB2 11 for z/OS Diagnosis Guide and Reference*, LY37-3222 to identify the service task and the RMID. See Table 18-1.

Table 18-1 Common DB2 service tasks

Task	RMID = Resource Manager ID	FC = Function Code
Data set close	x'4F'	x'0A'
Data set count	x'88'	x'12'
Data set define	x'68'	x'12'
Data set delete	x'6A'	x'12'
Data set extend	x'65'	x'12'
Data set open	x'59'	x'0A'
Data set reset	X'6C'	x'12'
DB2 requestor wait for server response (TCP/IP only)	x'8F'	x'1B'
Parallel query cleanup	x'84'	x'14' or x'77'
SYSLGRNX Update	x'44'	x'15'
Update Commit	x'49'	x'03'
VSAM catalog update	x'94'	x'0A'

If there is a long time gap between IFCID 46 and 47 (begin SRB task) or 49 (begin TCB task), then it is possible that the transaction is waiting for a service task to become available.



Stored procedures, user defined functions, and triggers

Stored procedures, user defined functions (UDFs), and triggers are considered “nested” activities. DB2 provides separate Class 1 and Class 2 (elapsed and CPU) timers for stored procedures, UDFs, and triggers. IFCID 3 provides plan level information and aggregates all executions of stored procedures or user defined functions into common fields. DB2 accounting data can distinguish between these activities:

- ▶ Non-nested activity
- ▶ Nested activity:
 - Triggers
 - Stored procedures
 - User defined functions

Note that accounting traces cannot distinguish between the level of nesting. When there is only one of each of these nested types, plan level accounting provided by IFCID 3 might provide enough information for your purposes.

When tuning multiple procedures or functions that are executed in a given transaction, detail package level information is available in IFCID 239 when accounting classes 7 and 8 are activated. Package-level accounting provides better granularity than IFCID 3, but still may not be sufficient for all transactions. This is because multiple package executions within a transaction are reported together, so CPU, elapsed and suspend time reflect averages across many stored procedure packages.

Performance traces can distinguish between the level of nesting and also detail individual runs of a stored procedure.

This chapter contains the following topics:

- ▶ Stored procedures
- ▶ User defined functions
- ▶ Triggers
- ▶ Nested activity accounting
- ▶ IFCIDs for nested activity

19.1 Stored procedures

There are four types of stored procedures:

- ▶ External stored procedures
- ▶ External SQL procedures
- ▶ Native SQL procedures
- ▶ Native autonomous SQL stored procedures (new with DB2 11)

External stored procedures and external SQL procedures run in a WLM-managed address space. Native stored procedures and autonomous run in the DB2 engine, unless they are run in debug mode.

Accounting can distinguish between class 1 and class 2 times for external stored procedures. For native SQL procedures, the class 1 time is always the same as the class 2 time. See Example 19-1.

Example 19-1 Stored procedure accounting - caller and procedure

----- Native SQLPL Procedure -----			----- External Procedure -----		
TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)
ELAPSED TIME	0.006572	0.001347	ELAPSED TIME	0.225686	0.001515
NONNESTED	0.006178	0.000953	NONNESTED	0.006713	0.000970
STORED PROC	0.000394	0.000394	STORED PROC	0.218973	0.000544
UDF	0.000000	0.000000	UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000	TRIGGER	0.000000	0.000000
CP CPU TIME	0.000964	0.000530	CP CPU TIME	0.003311	0.001110
AGENT	0.000964	0.000530	AGENT	0.003311	0.001110
NONNESTED	0.000643	0.000209	NONNESTED	0.001423	0.000708
STORED PROC	0.000321	0.000321	STORED PROC	0.001889	0.000402
UDF	0.000000	0.000000	UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000	TRIGGER	0.000000	0.000000

Autonomous stored procedures are native procedures that execute under their own units of work, separate from the calling program. They commit when they finish without committing the work of the calling program. Because these procedures are independent of the calling program, the caller and the stored procedure each generate their own independent accounting records.

Example 19-2 shows the accounting record for the calling program. It shows stored procedure elapsed time plus a nominal amount of stored procedure CPU time to create the new thread. The class 3 suspensions show wait time under the new AUTONOMOUS TX class 3 timer. And the data contains no timers or counters for the work that the stored procedure does.

Example 19-2 Autonomous stored procedure accounting - caller

TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	CLASS 3 SUSPENSIONS	ELAPSED	EVENTS
ELAPSED TIME	0.023385	0.017709	LOCK/LATCH(DB2+IRLM)	0.000000	0
NONNESTED	0.005840	0.000164	IRLM LOCK+LATCH	0.000000	0
STORED PROC	0.017545	0.017545	DB2 LATCH	0.000000	0
UDF	0.000000	0.000000	SYNCHRON. I/O	0.000000	0
TRIGGER	0.000000	0.000000	DATABASE I/O	0.000000	0
			LOG WRITE I/O	0.000000	0

CP CPU TIME	0.000645	0.000166	OTHER READ I/O	0.000000	0
AGENT	0.000645	0.000166	OTHER WRTE I/O	0.000000	0
NONNESTED	0.000615	0.000136	SER.TASK SWTCH	0.000105	1
STORED PRC	0.000030	0.000030	UPDATE COMMIT	0.000000	0
UDF	0.000000	0.000000	OPEN/CLOSE	0.000000	0
TRIGGER	0.000000	0.000000	SYSLGRNG REC	0.000000	0
PAR.TASKS	0.000000	0.000000	EXT/DEL/DEF	0.000000	0
			OTHER SERVICE	0.000105	1
			ARC.LOG(QUIES)	0.000000	0
			...		
			ACCELERATOR	0.000000	0
			AUTONOMOUS TX	0.017414	1
			TOTAL CLASS 3	0.017520	2

Example 19-3 shows the accounting data for the autonomous stored procedure. It shows no nested activity because it is its own thread, executing autonomously from the caller.

Example 19-3 Autonomous stored procedure accounting - stored procedure

TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	CLASS 3 SUSPENSIONS	ELAPSED	EVENTS
ELAPSED TIME	0.017307	0.017305	LOCK/LATCH(DB2+IRLM)	0.000000	0
NONNESTED	0.017307	0.017305	IRLM LOCK+LATCH	0.000000	0
STORED PROC	0.000000	0.000000	DB2 LATCH	0.000000	0
UDF	0.000000	0.000000	SYNCHRON. I/O	0.001504	5
TRIGGER	0.000000	0.000000	DATABASE I/O	0.000936	4
			LOG WRITE I/O	0.000568	1
CP CPU TIME	0.000605	0.000602	OTHER READ I/O	0.000000	0
AGENT	0.000605	0.000602	OTHER WRTE I/O	0.000000	0
NONNESTED	0.000605	0.000602	SER.TASK SWTCH	0.015105	3
STORED PRC	0.000000	0.000000	UPDATE COMMIT	0.000000	0
UDF	0.000000	0.000000	OPEN/CLOSE	0.000000	0
TRIGGER	0.000000	0.000000	SYSLGRNG REC	0.001295	1
PAR.TASKS	0.000000	0.000000	EXT/DEL/DEF	0.000000	0
...			OTHER SERVICE	0.013810	2
			ARC.LOG(QUIES)	0.000000	0
			...		
			TCP/IP LOB XML	0.000000	0
			ACCELERATOR	0.000000	0
			AUTONOMOUS TX	0.000000	0
			TOTAL CLASS 3	0.016609	8

Table 19-1 shows some high level diagnostics for external stored procedures, which you can also use for external user-defined functions:

Table 19-1 External stored procedure diagnostics

When these times are high	Class 1	Class 1 + Class 2
Elapsed	When only class 1 elapsed is high for external stored procedures and functions, investigate whether the nested activity is waiting on a WLM TCB, or whether the WLM environment is quiesced. See Example 19-4 for data from a stored procedure awaiting the WLM environment to be started.	Normal class 1 and class 2 CPU times coupled with high class 1 and class 2 elapsed times point to some suspension in DB2, or to NOT ACCOUNT time. See Chapter 16, "I/O suspensions" on page 231 for more information.

When these times are high	Class 1	Class 1 + Class 2
Elapsed + CPU	Very high stored procedure CPU times with an absence of high class 2 CPU points to an application issue. Check for loops and expensive processing within the application. See Example 19-5	High CPU time within DB2 could still be an indicator of a loop in the application, but one in which there is SQL. If the procedure shows that many SQL statements have been executed, review the application for a loop. If there have not been many SQL statements executed, investigate the package accounting data for more information.

Example 19-4 shows the accounting data for a stored procedure with a long wait for the WLM environment.

Example 19-4 Accounting for a stored procedure with a long wait for the WLM application environment

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)
-----	-----	-----
ELAPSED TIME	3:21.13843	0.000331
NONNESTED	3:21.13843	0.000248
STORED PROC	0.000000	0.000083
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
CP CPU TIME	0.001049	0.000190
AGENT	0.001049	0.000190
NONNESTED	0.001049	0.000120
STORED PRC	0.000000	0.000071
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
PAR.TASKS	0.000000	0.000000

Example 19-5 shows the accounting data for a looping stored procedure with no SQL in the loop.

Example 19-5 Accounting data from a looping stored procedure (no SQL in the loop)

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)
-----	-----	-----
ELAPSED TIME	33.313160	0.001041
NONNESTED	0.007218	0.000823
STORED PROC	33.305942	0.000219
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
CP CPU TIME	29.440969	0.000876
AGENT	29.440969	0.000876
NONNESTED	0.001608	0.000710
STORED PRC	29.439361	0.000166
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000

Example 19-6 shows the accounting data for a looping stored procedure with SQL in the loop.

Example 19-6 Application data from a looping stored procedure with SQL in the loop

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)
-----	-----	-----
ELAPSED TIME	9:54.02373	9:45.57101
NONNESTED	9:54.02373	0.000203
STORED PROC	0.000000	9:45.57080
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
CP CPU TIME	8:16.52814	8:08.39994
AGENT	8:16.52814	8:08.39994
NONNESTED	0.000733	0.000166
STORED PRC	8:16.52741	8:08.39977
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
PAR.TASKS	0.000000	0.000000

Detailed package level accounting when accounting class 10 is on provides the following values. Note the 20 million selects that are the cause of the high CPU.

GETPRMB	TOTAL
-----	-----
SELECT	20200930
INSERT	0
UPDATE	0
DELETE	0
DESCRIBE	0
PREPARE	0
OPEN	1
FETCH	0
CLOSE	0
LOCK TABLE	0
CALL	0

19.1.1 Nested activity for WITH RETURN processing

WITH RETURN on a define cursor statement specifies that the cursor is intended to be used to return a result set from the stored procedure.

In such a scenario, the stored procedure typically only performs the OPEN cursor, and the calling application does the FETCHing from the result set cursor. Figure 19-1 shows a transaction that involves a cursor defined as WITH RETURN.

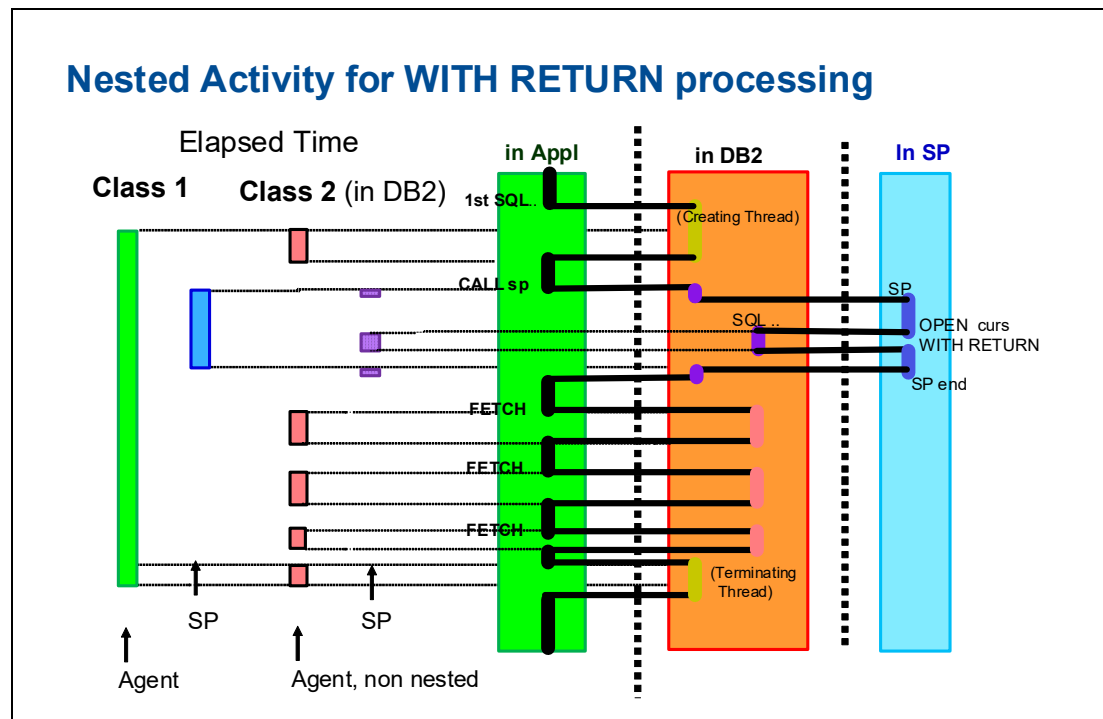


Figure 19-1 Nested activity for WITH RETURN

The application connects to DB2 and calls a stored procedure. The stored procedure opens a cursor WITH RETURN. In such a scenario, the stored procedure typically only performs the open cursor, and the calling application does the fetching from the result set cursor.

Figure 19-2 shows the DB2 accounting class 1 and 2 plan level sections. Note that there are rows fetched at the plan level, and that the time it took to do the fetch shows up under "NONNESTED."

DB2 Accounting Class 1-2 (Plan Level) Sections

TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)		
-----	-----	-----		
ELAPSED TIME	0.941479	0.608886		
NONNESTED	0.731353	0.402154	SQL DML	TOTAL
STORED PROC	0.210126	0.206732	-----	-----
UDF	0.000000	0.000000	SELECT	0
TRIGGER	0.000000	0.000000	INSERT	0
			ROWS	0
CP CPU TIME	0.249935	0.192192	UPDATE	0
AGENT	0.249935	0.192192	ROWS	0
NONNESTED	0.248412	0.191915	MERGE	0
STORED PRC	0.001523	0.000277	DELETE	0
UDF	0.000000	0.000000	ROWS	0
TRIGGER	0.000000	0.000000		
PAR.TASKS	0.000000	0.000000	DESCRIBE	0
...	DESC.TBL	0
SUSPEND TIME	0.000323	0.000000	PREPARE	0
AGENT	N/A	0.000000	OPEN	1
PAR.TASKS	N/A	0.000000	FETCH	5204
STORED PROC	0.000323	N/A	ROWS	5203
UDF	0.000000	N/A	CLOSE	1
NOT ACCOUNT.	N/A	0.416693	DML-ALL	5206
...		

Figure 19-2 DB2 Accounting Class 1-2 Plan Level

Figure 19-3 shows the package level activity for the sample transaction. One anomaly to note is that, in this situation, package level data shows the FETCHes (as well as CPU and elapsed time incurred) as if they were done by the stored procedure package that performed the OPEN cursor.

DB2 Accounting Report Class 7-8-10 (Package Lvl)

Package level data is being reported as if the FETCHes are executed by the SP

MRSBMCBM (main)	TOTAL	MRSBMS (SP)	TOTAL
-----	-----	-----	-----
SELECT	0	SELECT	0
INSERT	0	INSERT	0
UPDATE	0	UPDATE	0
DELETE	0	DELETE	0
DESCRIBE	0	DESCRIBE	0
PREPARE	0	PREPARE	0
OPEN	0	OPEN	1
FETCH	0	FETCH	5204
CLOSE	0	CLOSE	1
LOCK TABLE	0	LOCK TABLE	0
CALL	1	CALL	0

Figure 19-3 Package level accounting for WITH RETURN cursor in a stored procedure.

19.2 User defined functions

As with stored procedures, UDFs may or may not be run in a WLM-managed address space. UDFs that do not run in a WLM managed address space are known as SQL UDFs; the WLM-managed UDFs are called external UDFs.

As you can see in Example 19-7, SQL UDFs are recorded under NONNESTED time. Only External UDFs have time recorded in the accounting record, and in that record you can distinguish between class 1 and class 2 times.

Example 19-7 User-defined function accounting

----- SQL UDF -----			----- External UDF -----		
TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)
ELAPSED TIME	0.012346	0.006077	ELAPSED TIME	0.285920	0.000364
NONNESTED	0.012346	0.006077	NONNESTED	0.007835	0.000338
STORED PROC	0.000000	0.000000	STORED PROC	0.000000	0.000000
UDF	0.000000	0.000000	UDF	0.278085	0.000027
TRIGGER	0.000000	0.000000	TRIGGER	0.000000	0.000000
CP CPU TIME	0.004432	0.003055	CP CPU TIME	0.002239	0.000313
AGENT	0.004432	0.003055	AGENT	0.002239	0.000313
NONNESTED	0.004432	0.003055	NONNESTED	0.001822	0.000287
STORED PRC	0.000000	0.000000	STORED PRC	0.000000	0.000000
UDF	0.000000	0.000000	UDF	0.000417	0.000026
TRIGGER	0.000000	0.000000	TRIGGER	0.000000	0.000000

19.3 Triggers

Triggers execute completely within the DB2 engine and so you will always see their class 1 time equal to their class 2 times. See Example 19-6.

Example 19-8 Trigger Accounting

TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)
ELAPSED TIME	0.011908	0.004547
NONNESTED	0.010481	0.003120
STORED PROC	0.000000	0.000000
UDF	0.000000	0.000000
TRIGGER	0.001428	0.001428
CP CPU TIME	0.003568	0.002120
AGENT	0.003568	0.002120
NONNESTED	0.003252	0.001804
STORED PRC	0.000000	0.000000
UDF	0.000000	0.000000
TRIGGER	0.000316	0.000316
PAR.TASKS	0.000000	0.000000

19.4 Nested activity accounting

Figure 19-4 demonstrates a transaction that involves some nested activity: triggers and user defined functions.

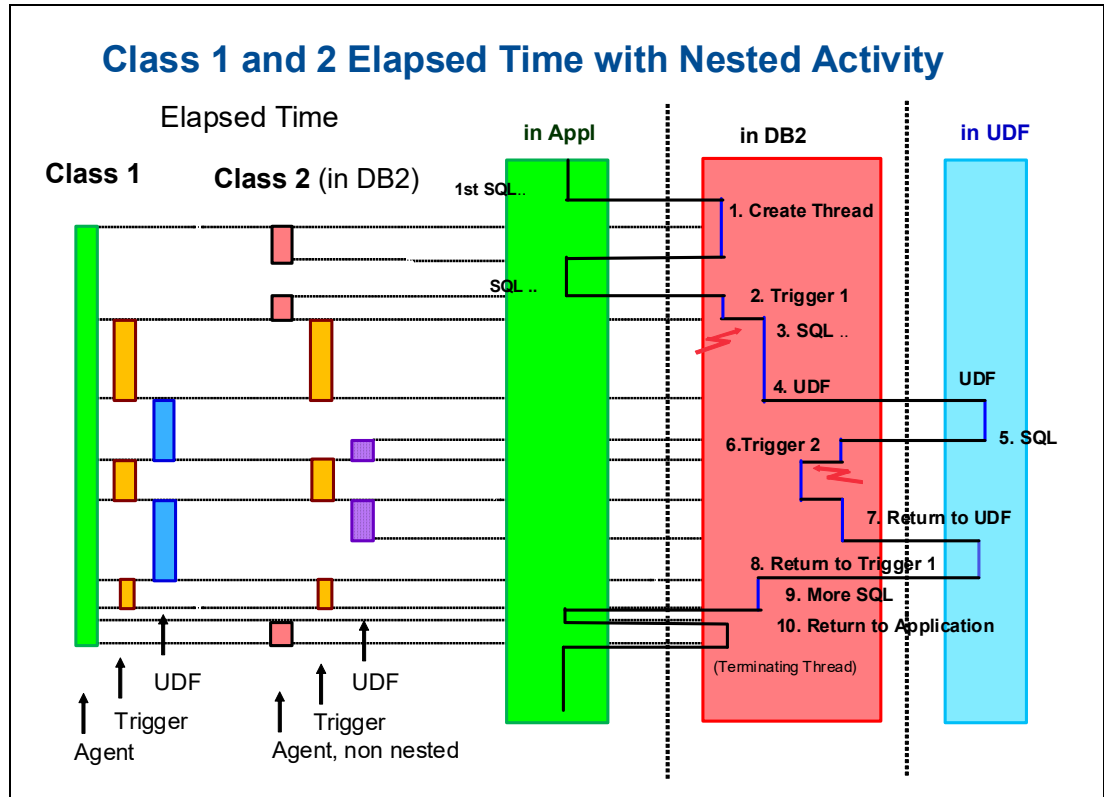


Figure 19-4 Accounting class 1,2,3 times with nested activities

The example can be used to understand which timers get activated and when. It represents the following process flow:

1. The allied thread is created when DB2 is first accessed.
2. An UPDATE statement bound in package A fires an "Instead Of" trigger (TR1). Triggers consist of pure SQL and are processed completely within the DB2 address spaces.
3. The trigger's SQL invokes an external (as opposed to internal and type casting functions) user defined function (UDF1).
4. The UDF starts in a WLM managed address space, i.e. outside DB2 address spaces.
5. The UDF accesses DB2. Control is passed back to DB2 address spaces. The UDF's SQL is bound in function package B.
6. The SQL statement executed by the UDF also contains a (before) trigger (TR2). Again we go back into DB2 to perform the trigger work.
7. When TR2 ends, and the SQL that invoked is completed, control returns to the function program UDF1.
8. After the UDF1 has completed, control returns to the initial trigger TR1.
9. Further SQL might be processed by the trigger.
10. Finally, the UPDATE statement of package A is completed, and control is passed back to the application program.

Figure 19-5 shows how the time of a nested transaction is reported on by DB2.

DB2 Accounting Report Class 1-2-3 Sections							
TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS	
ELAPSED TIME	1:48.52460	59.368366	N/P	LOCK/LATCH (DB2+IRLM)	0.046051	16774	
NONNESTED	4.407247	3.487657	N/A	IRLM LOCK+LATCH	0.001481	8	
STORED PROC	55.336590	55.336590	N/A	DB2 LATCH	0.044570	16766	
UDF	48.236686	0.000041	N/A	SYNCHRON. I/O	22.701562	115132	
TRIGGER	0.544077	0.544077	N/A	DATABASE I/O	22.592161	114809	
				LOG WRITE I/O	0.109400	323	
CP CPU TIME	11.144682	10.385807	N/P	OTHER READ I/O	20.312254	43086	
AGENT	11.144682	10.385807	N/A	OTHER WRTE I/O	1.657450	1199	
NONNESTED	1.321451	0.562962	N/P	SER.TASK SWTCH	2.796972	4942	
STORED PRC	9.684917	9.684917	N/A	UPDATE COMMIT	2.796972	4942	
UDF	0.000424	0.000038	N/A	OPEN/CLOSE	0.000000	0	
TRIGGER	0.137891	0.137891	N/A	SYSLGRNG REC	0.000000	0	
PAR.TASKS	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000000	0	
				OTHER SERVICE	0.000000	0	
SECP CPU	0.000000	N/A	N/A	ARC.LOG(QUIES)	0.000000	0	
				LOG READ	0.000000	0	
SE CPU TIME	0.003234	0.002733	N/A	DRAIN LOCK	0.000000	0	
NONNESTED	0.000561	0.000060	N/A	CLAIM RELEASE	0.000000	0	
STORED PROC	0.002674	0.002674	N/A	PAGE LATCH	0.000000	0	
UDF	0.000000	0.000000	N/A	NOTIFY MSGS	0.000000	0	
TRIGGER	0.000000	0.000000	N/A	GLOBAL CONTENTION	0.000000	0	
				COMMIT PH1 WRITE I/O	0.000000	0	
PAR.TASKS	0.000000	0.000000	N/A	ASYNCH CF REQUESTS	0.000000	0	
				TCP/IP LOB XML	0.000000	0	
SUSPEND TIME	48.103572	47.514289	N/A	ACCELERATOR	0.000000	0	
AGENT	N/A	47.514289	N/A	AUTONOMOUS TX	0.000000	0	
PAR.TASKS	N/A	0.000000	N/A	TOTAL CLASS 3	47.514289	181133	
STORED PROC	0.000000	N/A	N/A				
UDF	48.103572	N/A	N/A				

Figure 19-5 OMEGAMON Accounting Report

19.5 IFCIDs for nested activity

You can use the following IFCIDs to provide more effective performance and tuning analysis of stored procedures and UDFs:

- ▶ IFCID 233 is written at the beginning and end of a Stored Procedure or User-Defined Function invocation. This record includes the invoking statement ID, the invoking statement type, the version ID (for versioned procedures) and the routine ID.
- ▶ IFCIDs 380 (for stored procedures) and IFCID 381 (for user-defined functions) report CP, specialty engine, and elapsed time details for nested activity.
- ▶ IFCIDs 497, 498, and 499 show statement level detail. These records track dynamic and static DML statements executed by a transaction, including those executed within a stored procedure or user-defined function.
 - IFCID 497 is the statement ID detail record for statements executed outside of a stored procedure or UDF environment.
 - IFCID 498 is the statement ID detail record for statements executed inside of UDF environment. This includes WLM UDFs and non-inline scalar functions.
 - IFCID 499 is the statement ID detail record for statements executed inside of stored procedure environment. This includes both WLM and native stored procedures.
- ▶ A new performance class 24 encapsulates IFCID 380 and IFCID 499 for stored procedure detail analysis.

See Figure 19-6 for an illustration of stored procedure monitoring.

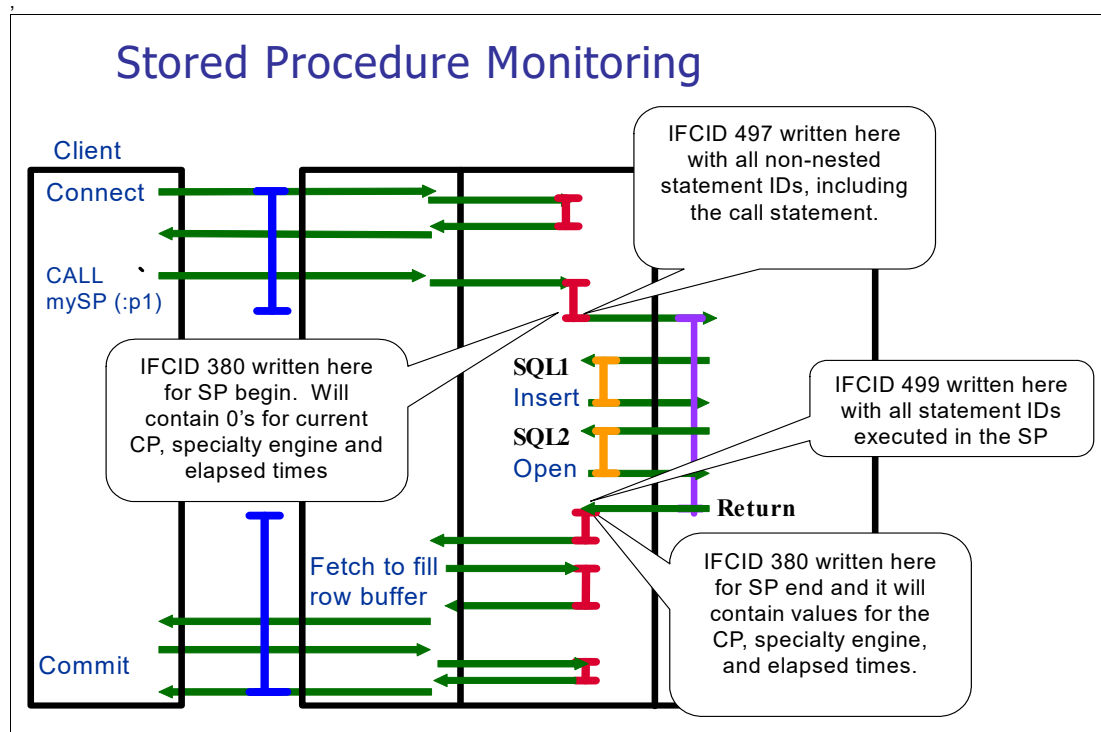


Figure 19-6 Stored procedure monitoring

If a trace was active to collect IFCIDs 233, 380, 497 and 499, these are the records that would be written:

- ▶ 497 to record the statement IDs of the SQL statements executed in the calling program, prior to and including the call procedure statement.
- ▶ 233 to record the beginning of the call statement. It shows the calling program's name, schema collection ID, and so on, as well as the routine ID and the nesting level.
- ▶ 380 to provide the beginning stored procedure detail record with beginning times.
- ▶ 499 to record the statement IDs of the SQL statements inside the stored procedure, including how many times they were executed.
- ▶ 233 to show the end of the stored procedure with the SQL code and the SQLCA of the call statement.
- ▶ 380 to show the ending class 1 and class 2 times for the stored procedure.
- ▶ 497 to record the SQL statement executed in the calling program after the stored procedure call.

For more information, see the descriptions of the IFCIDs in SDSNIVPD(DSNWMSGs).



DRDA, parallelism, and statement cache

In this chapter, we examine accounting for DDF work, query parallelism, capturing information from the statement cache, and performance traces.

The chapter contains the following topics:

- ▶ Accounting for DDF work
- ▶ Setting extensions to client information
- ▶ Accounting for parallel tasks
- ▶ Capturing performance data with the statement cache
- ▶ Performance traces for SQL tuning

20.1 Accounting for DDF work

For most connection types, when we talk about “time in DB2”, we mean class 2 time.

When DDF is involved, “time in DB2”, may also be interested in “time in the DB2 server”.

This includes the regular class 2 time, as well as elapsed time in stored procedures, UDFs, application code, CPU time spent inside the DDF address space, and so on.

Time in DB2 server is defined as follows:

- ▶ Class 2 non-nested elapsed time +
- ▶ Class 1 stored procedure, UDF, and trigger elapsed time +
- ▶ Non-nested (Class 1 CPU - Class 2 CPU) +
- ▶ Non-nested (class 1 SE CPU - class2 SE CPU)

(Note that the last two bullets calculate CPU time spent in the DDF address space).

- ▶ Time outside of DB2 server is total Class 1 elapsed less previous calculation.
- ▶ Active thread accounting records are created at thread deallocation (disconnect).
- ▶ Inactive thread accounting records are created at DBAT inactive (look for DBAT inactive).

DDF Accounting at plan level

The total time spent in DB2 can be determined as in Figure 20-1.

DDF Accounting at Plan Level			
TIMES/EVENTS	APPL (CL.1)	DB2 (CL.2)	
ELAPSED TIME	3:10.24012	3.387696	• Time in DB2 = class 2 nonnested ET
NONNESTED	3:10.24012	3.387696	+ class 1 nested activity ET
STORED PROC	0.000000	0.000000	[stored procedure + UDF + Trigger]
UDF	0.000000	0.000000	+ nonnested (class 1 CP CPU - class 2 CP CPU)
TRIGGER	0.000000	0.000000	+ nonnested (class 1 SE CPU - class2 SE CPU)
CP CPU TIME	1.020178	0.980084	• Time in DB2 = 3.387696
AGENT	0.979296	0.939458	+ (0.000000 + 0.000000 + 0.000000)
NONNESTED	0.979296	0.939458	+ (0.979296 - 0.939458)
STORED PROC	0.000000	0.000000	+ (0.008714 - 0.008714)
UDF	0.000000	0.000000	= 3.427534
TRIGGER	0.000000	0.000000	• Time outside DB2 = class 1 ET - Time in DB2
PAR.TASKS	0.040882	0.040626	• Time outside of DB2 =
SECP CPU	0.000000	N/A	3:10.24012 - 3.427534 = 3:06.8126
SE CPU TIME	0.008714	0.008714	• Note: Be careful with SE CPU time when
NONNESTED	0.008714	0.008714	running on a 'knee-capped' machine - using
STORED PROC	0.000000	0.000000	normalized CPU times
UDF	0.000000	0.000000	
TRIGGER	0.000000	0.000000	
PAR.TASKS	0.000000	0.000000	
SUSPEND TIME	0.000000	1.440549	
...	

Figure 20-1 DDF accounting at plan level

Let us look a little bit closer at the individual components that make up the “time in DB2.”

The Class 2 non-nested elapsed time is the total time spent by the application in DB2. It is the time spent by the application between every entry into DB2 and every exit out of DB2.

This time does not include the time taken by the application to execute a stored procedure, trigger, or UDF. (That is why those are to be added later on.)

The time consumed by UDFs, triggers, and stored procedures is part of the time spent in DB2, but is reported separately in the DB2 accounting report. So when determining the total time spent in DB2, the time spent in UDFs, triggers, and stored procedures needs to be added.

The difference between the class1 and class 2 non-nested CPU time accounts for the CPU time used by the thread (class 1 CPU time) to do other work than executing SQL statements (class 2 CPU time): It contains part of the thread creation and termination CPU time, but is mostly the CPU time that is needed to move data in and out of communications buffers. See Figure 20-2.

DDF Accounting at Plan Level -2			
----- DISTRIBUTED ACTIVITY -----			
REQUESTER	: DDF1	ROLLBCK(1) RECEIVED:	0
PRODUCT ID	: DB2	SQL RECEIVED	: 601
PRODUCT VERSION	: V9 R1 M5	MESSAGES SENT	: 603
METHOD	: DRDA PROTOCOL	MESSAGES RECEIVED	: 604
COMMIT(1) RECEIVED:	0	BYTES SENT	: 2087111
		BYTES RECEIVED	: 809497
COMMIT(2) RECEIVED :	1	COMMIT(2) RESP.SENT:	1
BACKOUT(2) RECEIVED:	0	BACKOUT(2) RESP.SENT:	0
COMMIT(2) PERFORMED:	0	BACKOUT(2) PERFORMED:	0
TRANSACTIONS RECV. :	1		
		THREADS INDOUBT	: 0
		ROWS SENT	: 46515
		BLOCKS SENT	: 340
		CONVERSAT.INITIATED:	1
		NBR RLUP THREADS	: N/A
		PREPARE RECEIVED	: 1
		LAST AGENT RECV.	: 0
		MESSAGES IN BUFFER	: 46515
		FORGET SENT	: 0
SQL DML TOTAL			

SELECT			0
INSERT			226
ROWS			282
UPDATE			30
ROWS			84
MERGE			0
DELETE			0
ROWS			0
DESCRIBE			0
DESC.TBL			0
PREPARE			144
OPEN			142
FETCH			199
ROWS			46515
CLOSE			1
DML-ALL			742

Figure 20-2 Accounting at plan level report

Blocking is crucial for DRDA performance: Rows are put into blocks which are then sent out in messages

Note the number of rows in DML operations for multi-row operations (implicit or explicit).

20.2 Setting extensions to client information

DB2 11 supports up to 255 bytes of client information in the fields shown in Table 20-1.

Prerequisite for this support is DB2 Connect V10.5 FP2 or equivalent level of drivers.

Table 20-1 Client info property values for type 4 connectivity to DB2

Property name	Maximum length	Default value	DB2 special register
ApplicationName	255	The string "db2jcc_application".	CURRENT CLIENT_APPLNAME
ClientAccountingInformation	255	A string of the form JCCversionclient-ip	CURRENT CLIENT_ACCTNG
ClientCorrelationToken	255	An LUWID (logical unit of work ID) that the data server generates	CURRENT CLIENT_CORR_TOKEN
ClientHostname	255	The string "db2jcc_local"	CURRENT CLIENT_WRKSTNNAME
ClientUser	128	The user ID that was specified when the connection was established	CURRENT CLIENT_USERID

JDBC 4.0 includes client information properties, which contain information about a connection to a data source. The `DatabaseMetaData.getClientInfoProperties` method returns a list of client info properties that the IBM Data Server Driver for JDBC and SQLJ supports.

In the IBM Data Server Driver for JDBC and SQLJ version 4.0 or later, the IBM Data Server Driver for JDBC and SQLJ-only methods are deprecated. You should use `java.sql.Connection.setClientInfo` instead.

With JDBC 4.0, Cast the `java.sql.Connection` object to a `com.ibm.db2.jcc.DB2Connection` is not necessary to set your client information. Example 20-1 shows snippet of Java code to set client information using JDBC 4.0.

Example 20-1 Sample coding for setting client information using JDBC 4.0

```
Connection con = conn.getConnection();

        con.setClientInfo("ApplicationName",
"MyAppName01234567890123456789012345678901234567890");
        con.setClientInfo("ClientAccountingInformation",
"AccInfo89012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789A1234567890123456789012345678901234567890123456789012345678901234567890123456789B123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890");
        con.setClientInfo("ClientHostname",
"MyClientComputer789012345678901234567890123456789012345678901234567890");
        con.setClientInfo("ClientUser",
"YasuhiroOhmori567890123456789012345678901234567890");
        con.setClientInfo("ClientCorrelationToken",
"MyClientToken456789012345678901234567890");
```

You can see the full client information as passed to DB2 in the **-DISPLAY THREAD** output as shown in Example 20-2. You can find the section V437 in the DSNV401I message with WORKSTATION, USERID, and APPLICATION information.

Example 20-2 -DISPLAY THREAD()*

```

-DB1A DIS THD(*)
DSNV401I  -DB1A DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DB1A ACTIVE THREADS - 599
NAME      ST A   REQ ID          AUTHID   PLAN      ASID TOKEN
SERVER    RA *    5  PROGNAME9999 DB2R6     DISTSERV 00A4    8
V437-WORKSTATION=MyClientComputer789012345678901234567890123456789012
34567890
USERID=Yasuhiro0hmori567890123456789012345678901234567890
APPLICATION NAME=MyAppName01234567890123456789012345678901234567
890
V442-CRTKN=MyClientToken456789012345678901234567890
V445-G944FF9C.C5BF.CC0DFE8E509E=8 ACCESSING DATA FOR
      ::9.68.255.156
  
```

Long client information passed from the application to the DB2 will be truncated in OMEGAMON PE accounting trace. Example 20-3 shows example output from accounting traces.

Example 20-3 Accounting identification block

1 LOCATION: DB1A GROUP: N/P MEMBER: N/P SUBSYSTEM: DB1A DB2 VERSION: V11	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R2M0) ACCOUNTING TRACE - LONG	PAGE: 1-34 REQUESTED FROM: NOT SPECIFIED TO: NOT SPECIFIED ACTUAL FROM: 13-09-03 19:46:43.79
--	--	---

----- IDENTIFICATION -----			
ACCT TSTAMP: 13-09-03 21:25:47.05	PLANNAME: MyAppNam	WLM SCL: DDFBAT	CICS NET: N/A
BEGIN TIME : 13-09-03 21:25:30.65	PROD TYP: JDBC DRIVER		CICS LUN: N/A
END TIME : 13-09-03 21:25:47.05	PROD VER: V4 R16M0	LWU NET: G9378992	CICS INS: N/A
REQUESTER : ::9.55.137.146	CORRNAME: db2jcc_a	LWU LUN: D447	
MAINPACK : SYSLH200	CORRNMBR: ppli	LWU INS: CBE919CC84BB	ENDUSER : Yasuhiro0hmori#1
PRMAUTH : DB2R6	CONNTYPE: DRDA	LWU SEQ: 2	TRANSACTION: MyAppName012345678901234567890#1
ORIGAUTH : DB2R6	CONNECT : SERVER		WSNAME : MyClientComputer#2

ELAPSED TIME DISTRIBUTION		CLASS 2 TIME DISTRIBUTION	
APPL		CPU	=====> 28%
DB2	=====> 98%	SECPU	=====> 41%
SUSP	=> 2%	NOTACC	=====> 29%
		SUSP	=> 2%

TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS	HIGHLIGHTS
-----	-----	-----	-----	-----	-----	-----	-----

Full client information will be separately reported in the end of the accounting records. Example 20-4 shows sample output setting longer value.

Example 20-4 Accounting - Truncated Values

----- IDENTIFICATION -----			
ACCT TSTAMP: 13-09-03 21:25:47.05	PLANNAME: MyAppNam	WLM SCL: DDFBAT	CICS NET: N/A
BEGIN TIME : 13-09-03 21:25:30.65	PROD TYP: JDBC DRIVER		CICS LUN: N/A
END TIME : 13-09-03 21:25:47.05	PROD VER: V4 R16M0	LWU NET: G9378992	CICS INS: N/A
REQUESTER : ::9.55.137.146	CORRNAME: db2jcc_a	LWU LUN: D447	
MAINPACK : SYSLH200	CORRNMBR: ppli	LWU INS: CBE919CC84BB	ENDUSER : Yasuhiro0hmori#1
PRMAUTH : DB2R6	CONNTYPE: DRDA	LWU SEQ: 2	TRANSACTION: MyAppName012345678901234567890#1
ORIGAUTH : DB2R6	CONNECT : SERVER		WSNAME : MyClientComputer#2

Rollup of parallelism information also applied to packages CPU cycles used by parallelism can be offloaded to zIIP when available (not the case in Figure 20-3).

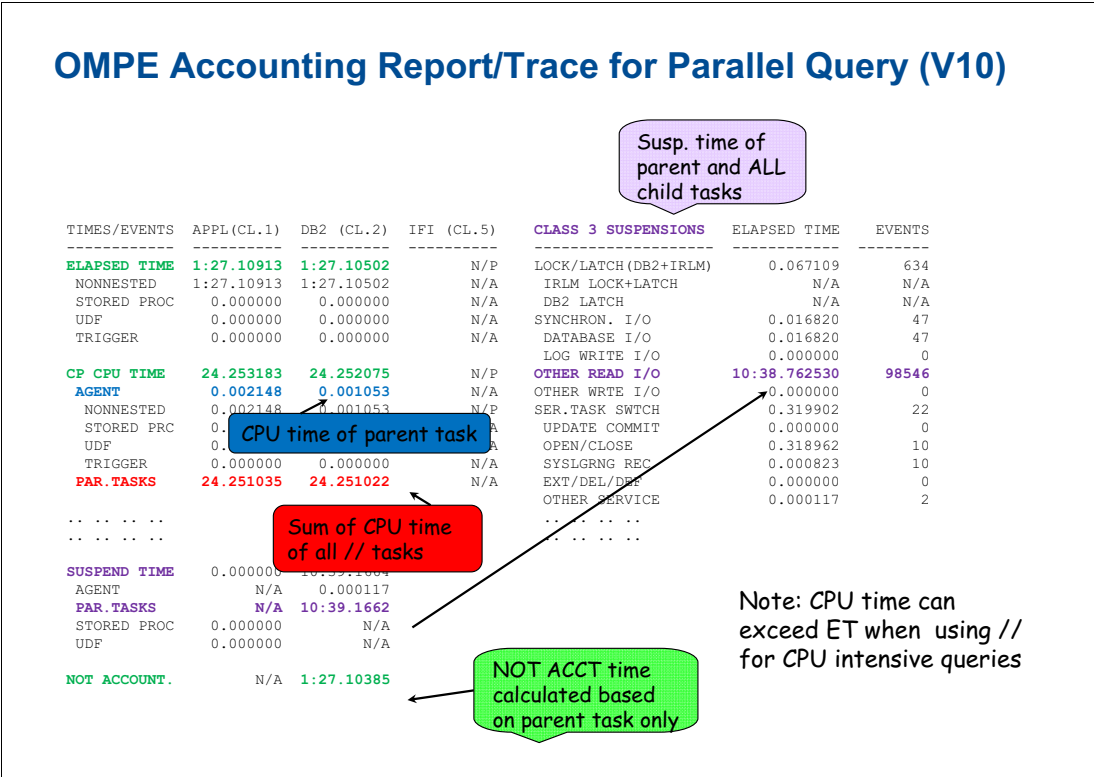


Figure 20-3 OMEGAMON PE Accounting Report/Trace for a DB2 10 parallel query

Note: CPU time can exceed elapsed time when using parallelism for CPU intensive queries.

DB2 version 11 contains a new bucket for parallel accounting called PQ SYNCHRONIZATION. PQ SYNCHRONIZATION contains the time the parent waits for child tasks to sync as well as the time the child waits for the parent to synch. In V10, these waits were recorded in NOT ACCOUNT time. Figure 20-4 shows an accounting report with the new field.

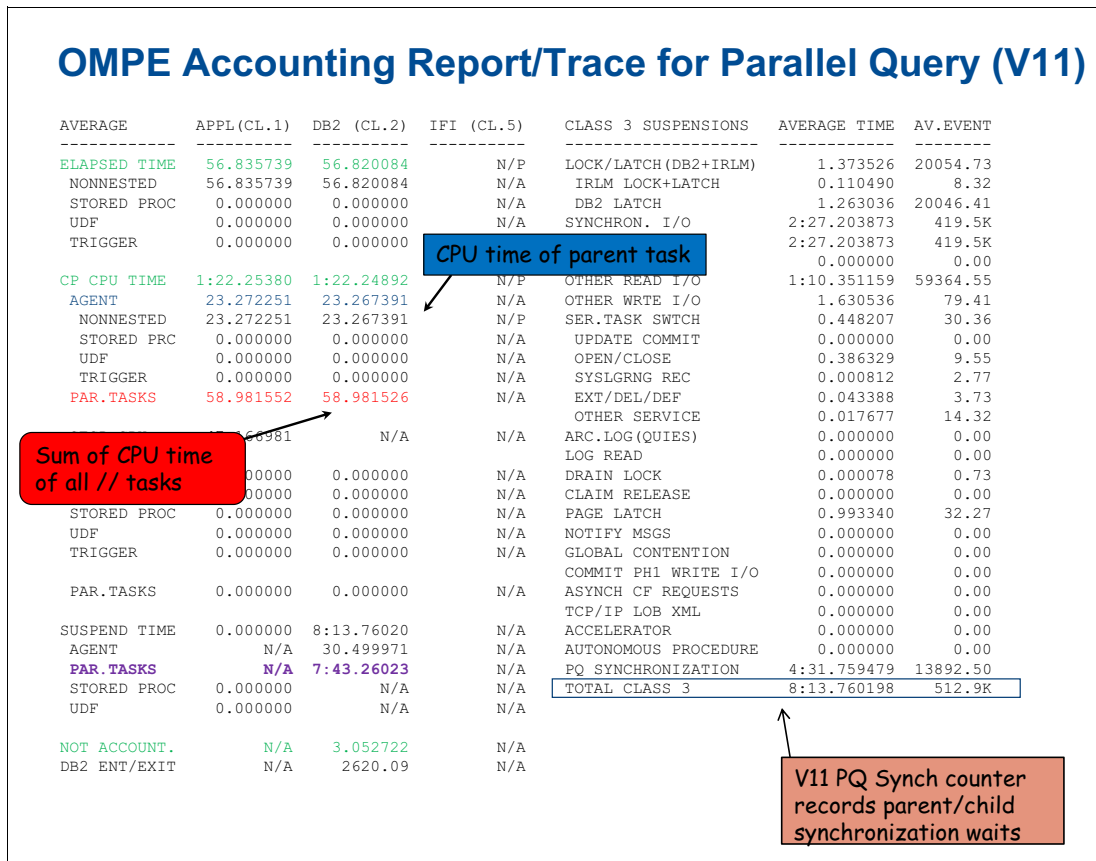


Figure 20-4 OMEGAMON PE Accounting Report/Trace for a V11 parallel Query

20.4 Capturing performance data with the statement cache

When DB2 prepares a dynamic SQL statement, it creates a control structure that is used when the statement is executed. When dynamic statement caching is in effect, DB2 stores the control structure associated with a prepared dynamic SQL statement in a storage pool. If that same statement is executed again, DB2 can use the cached control structure, avoiding the expense of re-preparing the statement.

The dynamic statement cache includes useful data points that help you evaluate the overall performance of your SQL statements. Follow these steps to externalize the statement cache statistics information for your performance analysis:

1. In your DSNZPARM, make sure **CACHE DYNAMIC SQL (CACHEDYN)** is set to **YES**.
2. Create **DSN_STATEMENT_CACHE_TABLE** to hold the statistics data by referring to sample job **DSNTESSC** in **HLQ.SDSNSAMP**. Use the definition of the **DSN_STATEMENT_CACHE_TABLE** as well as its LOB table space, AUX table, and indexes. If you are in CM mode, use the **DSNTESSC** member of the version of DB2 you migrated from.
3. Enable DB2 performance trace by issuing command **-START TRACE(P) CLASS(30) IFCID(318)**. IFCID 318 is a monitor trace instrumentation identifier; DB2 begins to collect statistics and accumulates them for the length of time when the trace is on. Stopping the trace resets all statistics. See Example 20-6.

Example 20-6 Start trace command for IFCID 318

```
-DB1A STA TRACE(P) CLASS(30) IFCID(318)
DSNW130I  -DB1A P TRACE STARTED, ASSIGNED TRACE NUMBER 02
DSN9022I  -DB1A DSNWVCM1 '-STA TRACE' NORMAL COMPLETION
```

4. Run the workload you want to monitor.
5. Issue statement **EXPLAIN STMTCACHE ALL** in a DSNTEP2 job to extract all statements from the global cache and to dump the statistics information to the DSN_STATEMENT_CACHE_TABLE.
6. Disable DB2 performance trace by issuing command **-STO TRACE(P) CLASS(30) IFCID(318)**

Note that the workload and the Explain should be run while the traces are still running.

Fields in the DSN_STATEMENT_CACHE_TABLE are described in *DB2 for z/OS Managing Performance*, SC19-4060-00 under the topic, “User tables that are supplied with DB2”. After the data is in the DSN_STATEMENT_CACHE_TABLE, you can use it in various ways to monitor the performance of your dynamic SQL at the statement level. For example, you can:

- Run selects against the table. For example, suppose you are looking for statements that use more than 1000 getpages, you can run the select in Example 20-7.

Example 20-7 Select statement against the DSN_STATEMENT_CACHE_TABLE

```
SELECT STMT_ID, PROGRAM_NAME, PRIMAUTH,
       STMT_TEXT
FROM SYSADM.DSN_STATEMENT_CACHE_TABLE WHERE
STAT_GPAG > 1000;
```

- If you are only looking for the worst performing SQL statements, you can put all of the statement table data into a spreadsheet and sort by the fields you are interested in, such as CPU, elapsed time, and getpages. In order to put the data into a spreadsheet:
 - Use Data Studio to select the data from the cache table and load it to a spreadsheet.
 - Use DSNTIAUL to unload the data into a delimited flat file, then FTP the file to your workstation and open it with a spreadsheet. Example 20-8 shows one example of DSNTIAUL control statements you can use to create the flat file.
 - Use other documented methods to load a spreadsheet with DB2 for z/OS data, as shown at this link:

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0606thakkar/index.html>

Example 20-8 DSNTIAUL control statements to load statement cache to delimited flat file

```
//DSNCACH EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD SYSOUT=*
//SYSREC00 DD DSN=DB2R7.DSNCACHE.UNLOAD,
//           BUFNO=20,UNIT=SYSDA,
//           DISP=(NEW,CATLG,DELETE),
//           SPACE=(CYL,(100,50),RLSE)
```

```

//SYSTSIN DD *
DSN SYSTEM(DB1A)
RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB11) -
      LIB('DB1AM.RUNLIB.LOAD') PARM('SQL')
END
//SYSIN      DD *
SELECT  CHAR(STMT_ID)                                , CHAR('!!')
      , CASE
          WHEN STMT_TOKEN IS NULL
          THEN CHAR(' ')
          ELSE SUBSTR(CHAR(STMT_TOKEN),1,20)
      END                                            , CHAR('!!')
      , CHAR(COLLID)                                , CHAR('!!')
      , SUBSTR(PROGRAM_NAME,1,40)                    , CHAR('!!')
      , INV_DROPALT                                  , CHAR('!!')
      , INV_REVOKE                                    , CHAR('!!')
      , INV_LRU                                       , CHAR('!!')
      , INV_RUNSTATS                                  , CHAR('!!')
      , CHAR(CACHED_TS)                              , CHAR('!!')
      , CHAR(USERS)                                   , CHAR('!!')
      , CHAR(COPIES)                                  , CHAR('!!')
      , CHAR(LINES)                                   , CHAR('!!')
      , SUBSTR(PRIMAUTH,1,40)                         , CHAR('!!')
      , SUBSTR(CURSQLID,1,40)                         , CHAR('!!')
      , SUBSTR(BIND_QUALIFIER,1,40)                   , CHAR('!!')
      , BIND_ISO                                       , CHAR('!!')
      , BIND_CDATA                                     , CHAR('!!')
      , BIND_DYNRL                                     , CHAR('!!')
      , BIND_DEGRE                                     , CHAR('!!')
      , BIND_SQLRL                                     , CHAR('!!')
      , BIND_CHOLD                                     , CHAR('!!')
      , CHAR(STAT_TS)                                  , CHAR('!!')
      , CHAR(STAT_EXEC)                                , CHAR('!!')
      , CHAR(STAT_GPAG)                                , CHAR('!!')
      , CHAR(STAT_SYNR)                                , CHAR('!!')
      , CHAR(STAT_WRIT)                                , CHAR('!!')
      , CHAR(STAT_EROW)                                , CHAR('!!')
      , CHAR(STAT_PROW)                                , CHAR('!!')
      , CHAR(STAT_SORT)                                , CHAR('!!')
      , CHAR(STAT_INDX)                                , CHAR('!!')
      , CHAR(STAT_RSCN)                                , CHAR('!!')
      , CHAR(STAT_RSCN)                                , CHAR('!!')
      , CHAR(STAT_PGRP)                                , CHAR('!!')
      , CHAR(STAT_ELAP)                                , CHAR('!!')
      , CHAR(STAT_CPU)                                 , CHAR('!!')
      , CHAR(STAT_SUS_SYNIO)                           , CHAR('!!')
      , CHAR(STAT_SUS_LOCK)                           , CHAR('!!')
      , CHAR(STAT_SUS_SWIT)                           , CHAR('!!')
      , CHAR(STAT_SUS_GLCK)                           , CHAR('!!')
      , CHAR(STAT_SUS_OTHR)                           , CHAR('!!')
      , CHAR(STAT_SUS_OTHW)                           , CHAR('!!')
      , CHAR(STAT_RIDLMT)                             , CHAR('!!')
      , CHAR(STAT_RIDSTOR)                             , CHAR('!!')
      , CHAR(EXPLAIN_TS)                              , CHAR('!!')
      , SUBSTR(SCHEMA,1,40)                            , CHAR('!!')

```



```

, BIND_RO_TYPE                                , CHAR('!')
, CHAR(BIND_RA_TOT)                            , CHAR('!')
, CHAR(GROUP_MEMBER)                          , CHAR('!')
, CHAR(STAT_EXECB)                            , CHAR('!')
, CHAR(STAT_GPAGB)                            , CHAR('!')
, CHAR(STAT_SYNRB)                            , CHAR('!')
, CHAR(STAT_WRITB)                            , CHAR('!')
, CHAR(STAT_EROWB)                            , CHAR('!')
, CHAR(STAT_PROWB)                            , CHAR('!')
, CHAR(STAT_SORTB)                            , CHAR('!')
, CHAR(STAT_INDXB)                            , CHAR('!')
, CHAR(STAT_RSCNB)                            , CHAR('!')
, CHAR(STAT_PGRPB)                            , CHAR('!')
, CHAR(STAT_RIDLIMTB)                         , CHAR('!')
, CHAR(STAT_RIDSTORB)                         , CHAR('!')
, LITERAL_REPL                                , CHAR('!')
, CHAR(STAT_SUS_LATCH)                        , CHAR('!')
, CHAR(STAT_SUS_PLATCH)                       , CHAR('!')
, CHAR(STAT_SUS_DRAIN)                        , CHAR('!')
, CHAR(STAT_SUS_CLAIM)                        , CHAR('!')
, CHAR(STAT_SUS_LOG)                          , CHAR('!')
, EXPANSION_REASON                            , CHAR('!')
, CHAR(VARCHAR(STMT_TEXT))                     , CHAR('!')
FROM SYSADM.DSN_STATEMENT_CACHE_TABLE;

```

20.5 Performance traces for SQL tuning

When the accounting and statistics traces do not provide enough detail to identify and manage poor performing SQL statements, performance traces can fill in the gaps. The drawback of performance traces is that they provide so much data that they should only be run for a very short time, and they should be filtered as much as possible.

Here is an example of a start trace command for performance trace with filters included, but remember that it will generate a lot of data and you should use filters as much as possible:

```

-START TRACE(P) CLASS(1,2,3) IFCID(xxx) DEST(xxx) TDATA(CPU,COR,TRA,DIST)
PLAN(myplan) PKGPROG(mypkg) AUTHID(authid1,authid2,authid3)

```

Identifying SQL: To identify which SQL in a transaction is an issue, you can use performance trace class(1,2,3) as above.

Getpages: When your SQL statement is generating too many getpages, IFCID 198 produces a record for every get page and every release page that the transaction issues. There is no performance class for IFCID 198 because it generates so much data. This IFCID can be especially useful when you are trying to diagnose which object the getpages are from. It will also tell you how many page hits versus misses there were and will identify the buffer pools.



Part 4

Appendixes

In this part of the book, we provide the following appendixes:

- ▶ Appendix A, “Production modeling” on page 291
- ▶ Appendix B, “IBM OMEGAMON XE for DB2 performance database” on page 297



A

Production modeling

In this appendix, we offer guidance on modeling asymmetric environments in order to provide consistent access paths across diverse systems.

In order to be able to model your production system in test, or obtain identical access paths from one data sharing member to another, it is not just a matter of copying over the RUNSTATS information about objects to the other environment.

We discuss the following topics:

- ▶ Functional requirements
- ▶ Simulating production
- ▶ Data sharing members on dissimilar hardware
- ▶ Generating input data for spreadsheets

A.1 Functional requirements

In order to properly determine the access path, the optimizer also considers CPU speed (model dependent, number of processors (CP+zIIP) for parallelism, buffer pool size, RID pool size, and SORT pool size. These details become important when you have different hardware or DB2 subsystem configurations between development and test, test and QA, QA and production, and even between different members of a data sharing group.

Inter-technology coexistence is often seen in large data sharing groups where one processor may be a generation back, and it is certainly relevant when any customers upgrade their hardware. If there is a coexistence period, then access paths for the same dynamic SQL may differ depending on what member (hardware) they are executed against. If applications are rebound during this period, the access paths may differ depending on what CEC they were running against. See Figure A-1.

Production modeling

- DB2 9 APAR PM26475 and DB2 10 APAR PM26973
 - Supports optimizer overrides for system settings
 - New DSNZPARMs
 - SIMULATED_CPU_SPEED
 - SIMULATED_CPU_COUNT
 - New SYSIBM.DSN_PROFILE_ATTRIBUTES
 - SORT_POOL_SIZE
 - MAX_RIDBLOCKS
 - For buffer pools
 - Same as the BP names listed in the DSNTIP1 panel
 - For example a KEYWORDS value of 'BP8K0' corresponds to BP BP8K0
 - NOTE: Overrides apply to EXPLAIN, BIND/REBIND and prepare

Figure A-1 APAR for production modeling

Tip: APAR PM26973 provided the two DSNZPARMs:

- ▶ SIMULATED_CPU_SPEED
- ▶ SIMULATED_CPU_COUNT

It also provided three profile attribute keywords:

- ▶ SORT_POOL_SIZE
- ▶ MAX_RIDBLOCKS
- ▶ Buffer pool name.

For the APAR, see the following website:

<http://www-01.ibm.com/support/docview.wss?uid=swg1PM26973>

The DB2 Information Center has examples of the queries that also convert the result from HEX into integer values, see the following website:

http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z10.doc.perf/src/tpc/db2z_modelproductionenvironment.htm

A.2 Simulating production

In order to set up your test environment similar to production, here are the sample steps for a DB2 10 subsystem:

1. Capture the production settings using the SQL statement in Example A-1 and using a unique QUERYNO value that does not exist in the cache. You also need to convert the hex value to an integer.

Example A-1 Selecting the values from the PLAN_TABLE

```
SET CURRENT DEGREE='ANY';
EXPLAIN ALL SET QUERYNO=1234 FOR
SELECT * FROM SYSIBM.SYSDUMMY1;

SELECT HEX(SUBSTR(IBM_SERVICE_DATA,25,2)) AS CPU_COUNT,
       HEX(SUBSTR(IBM_SERVICE_DATA,69,4)) AS CPU_SPEED,
       HEX(SUBSTR(IBM_SERVICE_DATA,13,4)) AS RIDPOOL,
       HEX(SUBSTR(IBM_SERVICE_DATA,9,4)) AS SORT_POOL_SIZE
FROM PLAN_TABLE WHERE QUERYNO=1234;
```

2. Ensure that the explain tables PLAN_TABLE and DSN_STATEMENT_TABLE exist.
3. Ensure that the profile monitoring is enabled using DSNTIJOS to create the necessary tables.
4. Execute the insert statement in Example A-2 to create a global profile for a single DB2 subsystem which is active when profile monitoring is on. Any unique PROFILEID can be used.

Example A-2 Inserting a PROFILEID

```
INSERT INTO SYSIBM.DSN_PROFILE_TABLE (PROFILEID)
VALUES (4713);
```

5. You need to insert 1 row for each of the global parameter values you want to override in the ID (4713), so 1 for each buffer pool size that is different as well as the CPU, RID, or SORT pools. See Example A-3.

Example A-3 Examples of individual overrides

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
  (PROFILEID,KEYWORDS,ATTRIBUTE1,ATTRIBUTE2)
VALUES
  (4713, 'BP8K0',NULL, 2500);
```

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
  (PROFILEID,KEYWORDS,ATTRIBUTE1,ATTRIBUTE2)
VALUES
  (4713, 'SORT_POOL_SIZE',NULL, 307200);
```

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
  (PROFILEID,KEYWORDS,ATTRIBUTE1,ATTRIBUTE2)
VALUES
  (4713, 'MAX_RIDBLOCKS',NULL, 300);
```

6. Using the CPU_SPEED and CPU_COUNT values from step 1, set parameters DSN6SPRM.SIMULATED_CPU_SPEED and DSN6SPRM.SIMULATED_CPU_COUNT to match the production system. Note that the CPU_COUNT will only be available if parallelism is chosen, so you may not be interesting in this field. These are online changeable and will take effect outside of the activation of the profile.
7. Issue the **-START PROFILE** command to activate the rows that you inserted into the profile table.
8. Execute an EXPLAIN. In DSN_STATEMNT_TABLE the column REASON contains the value 'PROFILEID 4713' appended to the existing REASON value for that statement. The column REASON has this value when a subsystem parameter exists in SYSIBM.DSN_PROFILE_ATTRIBUTES and is for an active global profile in SYSIBM.DSN_PROFILE_TABLE where PROFILE_ENABLED = 'Y'.

A.3 Data sharing members on dissimilar hardware

In the presence of different hardware across data sharing group members, do these steps:

1. Issue the SELECT of the CPU speed from each member as shown in Example A-4. Of course those members running on LPARs of the same CEC are equal, but just to be on the safe side.

Example A-4 EXPLAIN a query and then select the CPU speed from each member

```
EXPLAIN ALL SET QUERYNO=1234 FOR
SELECT * FROM SYSIBM.SYSDUMMY1;
```

```
SELECT HEX(SUBSTR(IBM_SERVICE_DATA,69,4)) AS CPU_SPEED
FROM PLAN_TABLE WHERE QUERYNO=1234;
```

2. Convert the CPU speed from HEX to decimal via a calculator or the extended version of the query listed in the Information Center.
3. Choose the highest value returned from these queries as the baseline to set for other members.

4. Set SIMULATE_CPU_SPEED for each of the other members and recycle that member.
5. Reissue the SQL for the two query numbers to ensure that the update was successful. See Example A-5.

Example A-5 EXPLAIN another statement and compare the before and after CPU_SPEEDs

```
EXPLAIN ALL SET QUERYNO=1235 FOR  
SELECT * FROM SYSIBM.SYSDUMMY1;
```

```
SELECT QUERYNO,HEX(SUBSTR(IBM_SERVICE_DATA,69,4)) AS CPU_SPEED  
FROM PLAN_TABLE WHERE QUERYNO IN (1234, 1235);
```

A.4 Generating input data for spreadsheets

You can also use the OMPE File data sets to generate CSV (comma-separated value) input-data. This CSV data can then be transferred to workstations and imported into spreadsheets to improve DB2 performance analysis using graphical representations or pivot tables. See “Chapter 25. Generating input data for spreadsheets” at the following website:

<http://www-01.ibm.com/support/docview.wss?uid=swg27021212&aid=1>



IBM OMEGAMON XE for DB2 performance database

OMEGAMON XE for DB2 provides a performance database (PDB), which you can use to store historical information in DB2 tables. Using these tables can be useful for problem determination, application profiling, KPI monitoring, and capacity planning.

In this appendix, we introduce the OMEGAMON PDB, and outline how to create the PDB database, and extract, transform, and load (ETL) DB2 trace information into the PDB tables.

The appendix covers the following topics:

- ▶ Introduction
- ▶ Creating the performance database
- ▶ Extracting, transforming, and loading accounting and statistics data
- ▶ Additional information

B.1 Introduction

The PDB consists of a set of tables that you can populate with information from DB2 statistics, accounting, performance, locking, and audit traces. The population process is also referred to as *extract, transform, and load* (ETL). We provide an overview of the PDB ETL process in Figure B-1.

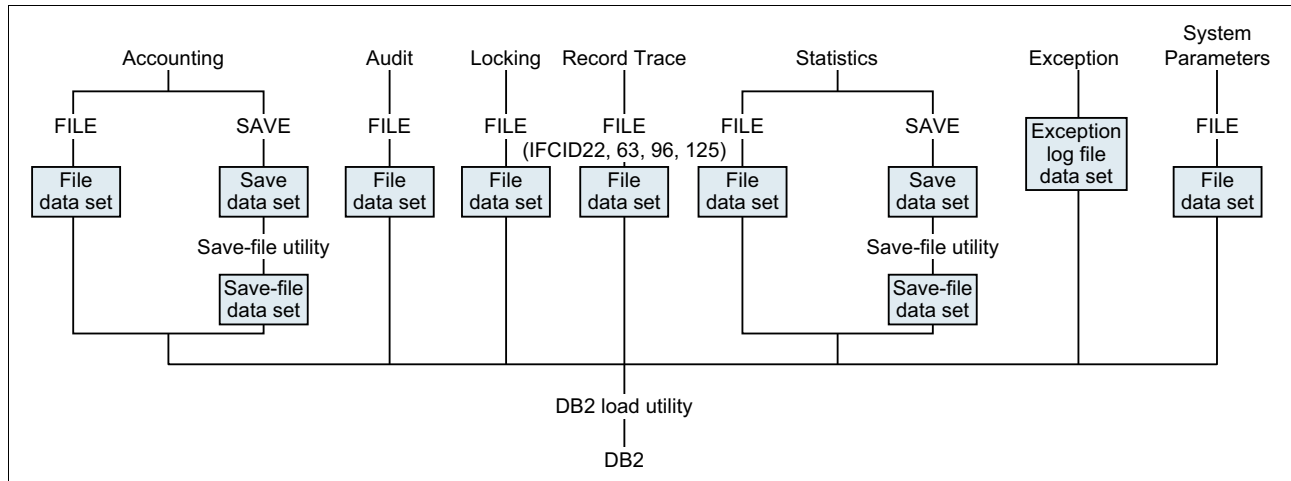


Figure B-1 OMEGAMON PDB ETL overview

As indicated in Figure B-1, ETL processes non-aggregated (FILE format) and aggregated (SAVE format) information.

- Aggregated information:

Several records are summarized by specific identifiers. In a report, each entry represents aggregated data. You run the **SAVE** subcommand to generate a VSAM data set that contains the aggregated data. When the data is saved, you use the Save-File utility to generate a DB2-loadable data set. As you might have noticed in Figure B-1, this format is supported only for statistics and accounting trace information. This option is useful if you must process huge volumes of accounting information.

- Non-aggregated information:

For non-aggregated data, each record is listed in the order of occurrence. In a trace, each entry represents non-aggregated data. You run the **FILE** subcommand to generate a data set that contains non-aggregated data. This format is supported for all DB2 trace information. Analyzing non-aggregated accounting information can be useful if you want to use the report capabilities of SQL to drill down on thread level accounting information. In our scenario, the volume of DB2 trace information is not expected to be large. We therefore decided to load the PDB tables with non-aggregated information.

With PDB ETL, you can process DB2 trace data of the following input formats:

- System Measurement Facility (SMF) record types 100 (statistics), 101 (accounting), and 102 (performance and audit).
- Generalized Trace Facility (GTF).
- OMEGAMON PE ISPF interface (collect report data).
- Batch program FPEZCRD. For an example of how to run program FPEZCRD in batch, refer the JCL sample that is provided in the RKO2SAMP library, member FPEZCRDJ.
- Near term history sequential data sets.

In our DB2 environment, we processed DB2 traces that we collected through SMF and GTF.

B.1.1 Performance database structure

The PDB database design is provided by OMEGAMON and comes with a set of tables to store DB2 trace data of the following information categories:

- ▶ Accounting
- ▶ Audit
- ▶ Exceptions
- ▶ Locking
- ▶ Record trace
- ▶ Statistics
- ▶ System parameters

For this book, we focused on using non-aggregated accounting and statistics information. If you need details about using the PDB for the other information categories, see Chapter 5, “Advanced reporting concepts. The Performance Database and the Performance Warehouse”, in *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Reporting User's Guide Version 5.1.0*, SH12-6927.

Accounting tables

Figure B-2 shows the accounting table categories that are provided by the performance database. PDB stores each data type in its own DB2 table.

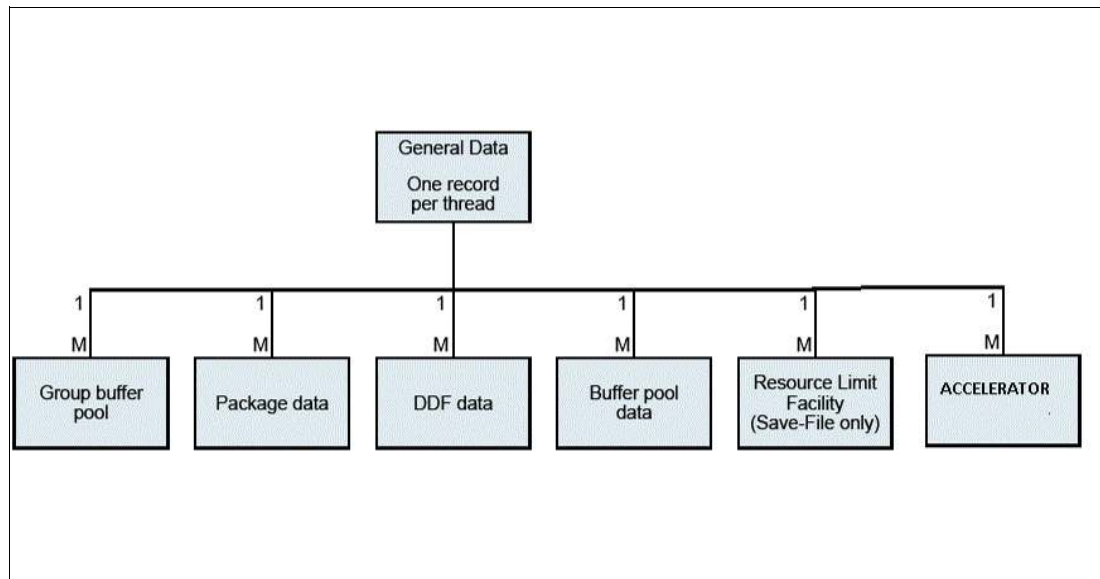


Figure B-2 PDB structure accounting tables

OMEGAMON PE provides two sets of accounting tables:

- ▶ FILE accounting tables, which detailed information so that you can use SQL to query accounting information on a thread level.
- ▶ SAVE accounting tables, which store aggregated data so that you can use SQL to query summarized accounting information of the time interval boundary.

FILE accounting tables

Each table type that is shown in Figure B-2 stores the following information:

- ▶ General data: General accounting information (one row per thread)
- ▶ Group buffer pool: For each thread, one row per group buffer pool that is being used
- ▶ Package data: For each thread, one row per package that is being used

- Buffer pool: For each thread, one row per buffer pool that is being used
- Resource limit facility: One row per resource limit type that is encountered

SAVE accounting tables

Each table type that is shown in Figure B-2 on page 299 stores the following aggregated information:

- General data: General accounting information, one row per aggregation interval
- Group buffer pool: For each aggregation interval, one row per group buffer pool that is being used
- Package data: For each aggregation interval, one row per package that is being used
- Buffer pool: For each aggregation interval, one row per buffer pool that is being used

Accounting table DDL and load statements

OMEGAMON PE provides sample create table DDL, load utility control statement templates, and table metadata descriptions in the RKO2SAMP library members shown in Table B-1 and Table B-2. We used these templates to create and load these accounting tables.

Table B-1 FILE accounting table DDL and load statements

Table name	Type	RKO2SAMP create table DDL	RKO2SAMP load utility statements	RKO2SAMP table metadata documentation
DB2PMFACCT_BUFFER	Buffer pool data	DGOACFBU	DOGALFBU	DGOABFBU
DB2PMFACCT_GENERAL	General data	DGOACFGE	DGOALFGE	DGOABFGE
DB2PMFACCT_GBUFFER	Group buffer pool	DGOACFGP	DGOALFGP	DGOABFGP
DB2PMFACCT_PROGRAM	Package data	DGOACFPK	DGOALFPK	DGOABFPK
DB2PMFACCT_DDF	DDF data	DGOACDFD	DGOALFDF	DGOABFDF

Table B-2 SAVE accounting table DDL and load statements

Table name	Type	RKO2SAMP create table DDL	RKO2SAMP load utility statements	RKO2SAMP table metadata documentation
DB2PMSACCT_BUFFER	Buffer pool data	DGOACSBU	DOGALSBU	DOGABSBU
DB2PMSACCT_GENERAL	General data	DGOACSGE	DOGALSGE	DOGABSGE
DB2PMFACCT_GBUFFER	Group buffer pool	DGOACSGP	DOGALSGP	DOGABSGP
DB2PMFACCT_PROGRAM	Package data	DGOACSPK	DOGALSPK	DOGABSPK

Table name	Type	RKO2SAMP create table DDL	RKO2SAMP load utility statements	RKO2SAMP table metadata documentation
DB2PMFACCT_DDF	DDF data	DGOACSDF	DOGALSDF	DOGABSDF

Statistics tables DDL and load statements

Figure B-3 shows the structure of each of the statistics tables in the performance database. PDB stored each data type in its own DB2 table.

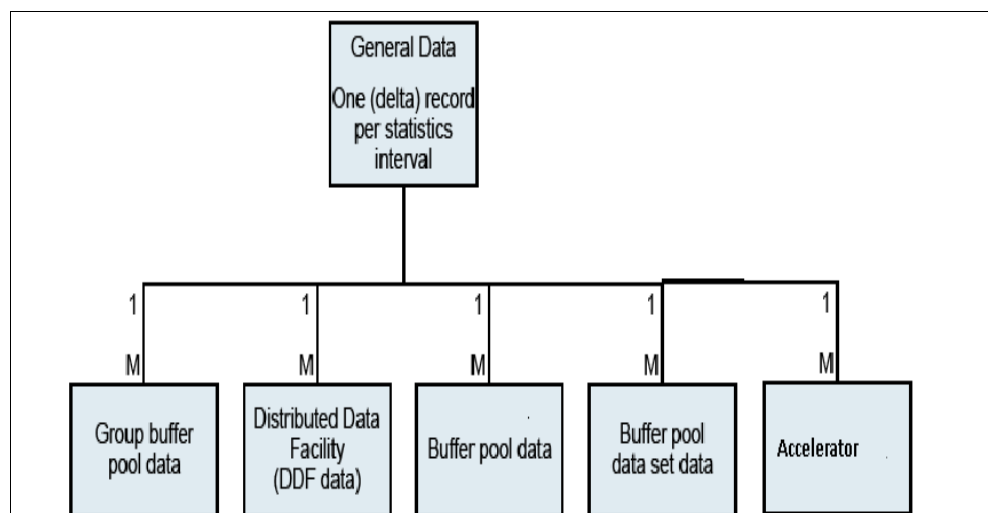


Figure B-3 PDB structure statistics tables

In our environment, we generate loadable input records in the FILE data format. In that format, each table type that is shown in Figure B-3 stores the following information:

- ▶ General data: One row for each Statistics delta record, containing data from IFCID 1 and 2. A delta record describes the activity between two consecutive statistics record pairs.
- ▶ Group buffer pool data: One row per group buffer pool that is active at the start of the corresponding delta record.
- ▶ DDF data: One row per remote location that is participating in distributed activity by using the DB2 private protocol and one row for all remote locations that used DRDA.
- ▶ Buffer pool data: One row per buffer pool that is active at the start of the corresponding delta record.
- ▶ Buffer pool data set data: One row for each open data set that has an I/O event rate at least one event per second during the reporting interval. To obtain that statistics trace information, you must activate statistics trace class 9.

OMEGAMON provides sample create table DDL, load utility control statement templates, and table metadata descriptions in the RKO2SAMP library members that are shown in Table B-3. We used these templates to create and load these statistics tables.

Table B-3 Statistics table DDL and load statements

Table name	Type	RK02SAMP create Table DDL	RK02SAMP load utility statements	RK02SAMP table metadata documentation
DB2PM_STAT_GENERAL	General data	DGOSCGEN	DGOSLGEN	DGOSBGEN
DB2PM_STAT_GBUFFER	Group buffer pool data	DGOSCGBP	DGOSLGBP	DGOSBGBP
DDB2PM_STAT_DDF	DDF data	DGOSCDDF	DGOSLDDF	DGOSBDDF
DB2PM_STAT_BUFFER	Buffer pool data	DGOSCBUF	DGOSLBUF	DGOSBBUF

B.2 Creating the performance database

We used the create table DDL RK02SAMP library members that are described in B.1.1, “Performance database structure” on page 299 to create the PDB accounting and statistics tables. To create the PDB, we performed the following activities:

- ▶ Create a DB2 for z/OS database to store the PDB tables.
- ▶ Customize PDB create table DDL.
- ▶ Create PDB tables.

B.2.1 Creating a DB2 z/OS database

We ran the SQL shown in Example B-1 to create the DB2 for z/OS database that we used to create the PDB tables. In our PDB environment, table spaces use buffer pool BP1, and index spaces use BP2.

Example B-1 PDB create DB2 z/OS database

```
CREATE DATABASE PMPDB
  BUFFERPOOL BP1
  INDEXBP BP2
  CCSID EBCDIC
  STOGROUP SYSDEFLT;
```

B.2.2 Customizing the PDB create table DDL

The OMEGAMON -provided PDB create table DDL statements require customization, as OMEGAMON does not provide an interface for providing PDB table qualifier, database, and table space names. In addition, the PDB provided database design does not provide create table space DDL and does not provide for indexes that are required to ensure uniqueness of data and to support query performance. To perform this customization, we performed the following tasks:

- ▶ Generate a create table DDL data set that contains all DDL statements.
- ▶ Modify a create table DDL to reflect the PDB database name and table qualifier.

Generating a create table DDL data set

We ran the JCL that is shown in Example B-2 to merge the accounting and statistics create DDL statements that are shown in Table B-1 on page 300, Table B-2 on page 300, and Table B-3 on page 302 in to a data set. For application profiling, we run queries on aggregated accounting information we created in the OMEGAMON PE accounting tables that are described in “SAVE accounting tables” on page 300.

Example B-2 PDB generate create table DDL data set

```
//S1GEN EXEC PGM=IEBGENER
//SYSUT1 DD *
SET CURRENT SCHEMA = 'PDB';
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACFBU)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACFDF)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACFGE)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACFGP)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACFPK)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACSBU)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACSDF)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACSGE)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACSGP)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACSPK)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOACSRF)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOSCBUF)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOSCCDF)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOSCGBP)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOSCGEN)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOSCSET)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOWCSFP)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOWC106)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOWC201)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOWC202)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOWC230)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DGOWC256)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
```

```
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCBD)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCBD)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCCHG)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCNT)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCDDL)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCML)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCFAI)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCSQL)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
// DD DISP=SHR,DSN=<omh1q>.RK02SAMP(DG0XCUTI)
// DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOL0)
//SYSUT2 DD DISP=SHR,DSN=DB2R3.PM.CNTL($04DDLTB)
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
```

Customizing a create table DDL

Next, we customize the create table DDL data set that we generated in Example B-2 on page 303. You might notice that we set the current schema to control the table qualifier and that we inserted a semicolon to separate the create table statements for SQL batch processing. We ran the ISPF edit command that is illustrated in Figure B-4 to modify the DDL to use the database that we created in B.2.1, “Creating a DB2 z/OS database” on page 302 for table creation.

File	Edit	Edit_Settings	Menu	Utilities	Compilers	Test	Help

EDIT	DB2R3.PM.CNTL(\$03DDL) - 01.00					Columns	0010 0072
Command ==> c 'IN DB2PM.' 'IN PMPDB.' ALL				Scroll ==> CSR			
***** ***** Top of Data *****							
000001	SET CURRENT SCHEMA = 'PDB';						
000002	--**Start of Specifications*****						
000003	--*						
000004	--* MODULE-NAME = DG0SCBUF						*
000005	--* DESCRIPTIVE-NAME = SQL for creating Statistics Buffer Pool Table						*

Figure B-4 Customize a create table DDL

Creating table spaces

The create table DDL statements reference the following table spaces in the table space clause:

- ▶ PMPDB.TSPAFCBU
- ▶ PMPDB.TSPAFCDF
- ▶ PMPDB.TSPAFCGE
- ▶ PMPDB.TSPAFCGP
- ▶ PMPDB.TSPAFCPK
- ▶ PMDB.TSPASBU
- ▶ PMDB.TSPASDF

- ▶ PMDB.TSPASGE
- ▶ PMDB.TSPASGP
- ▶ PMDB.TSPASPK
- ▶ PMPDB.TSPSBUF
- ▶ PMPDB.TSPSDDF
- ▶ PMPDB.TSPSGBP
- ▶ PMPDB.TSPSGEN
- ▶ PMPDB.TSPSSET

As these table spaces do not yet exist, we used the create table space DDL template that is shown in Example B-3 to create these table spaces. The template supports table space compression and uses the primary and secondary space quantity sliding scale feature to take advantage of autonomic space management.

Example B-3 Create table space template

```
CREATE TABLESPACE <tsname>
  IN PMPDB
  USING STOGROUP SYSDEFLT
  PRIQTY -1 SECQTY -1
  ERASE NO
  FREEPAGE 0 PCTFREE 5
  GBPCACHE CHANGED
  TRACKMOD YES
  LOGGED
  SEGSIZE 64
  BUFFERPOOL BP1
  LOCKSIZE ANY
  LOCKMAX SYSTEM
  CLOSE YES
  COMPRESS YES
  CCSID      EBCDIC
  DEFINE YES
  MAXROWS 255;
```

B.2.3 Creating the PDB accounting and statistics tables

Now, the DB2 for z/OS database PMPDB and the table spaces that are required for the tables are created and a generated a data set with customized create table DDL statements exists. Next, we run the batch JCL that is shown in Example B-4 to run the create table DDL statements that we customized in “Customizing a create table DDL” on page 304.

Example B-4 Batch JCL PDB accounting and statistics table creation

```
//S10TEP2 EXEC PGM=IKJEFT1B,DYNAMNBR=20,TIME=1440
//STEPLIB DD DISP=SHR,DSN=DB0ZT.SDSNEXIT
//          DD DISP=SHR,DSN=DB0ZT.SDSNLOAD
//          DD DISP=SHR,DSN=DB0ZT.RUNLIB.LOAD
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
           DSN SYSTEM(DOZG)
           RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP10)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD DISP=SHR,DSN=DB2R3.PM.CNTL($04DDLTB)
```

B.3 Extracting, transforming, and loading accounting and statistics data

Next, we extract, transform, and load (ETL) DB2 accounting and statistics trace information in to the PDB tables that we created in B.2.3, “Creating the PDB accounting and statistics tables” on page 305. The ETL process consists of the following processing steps:

1. Extract and transform DB2 trace data into an OMEGAMON Performance Expert (OMEGAMON PE) FILE formatted data set
2. Load the OMEGAMON PE FILE formatted data set into DB2 tables.
3. Extract transform and DB2 trace date into an OMEGAMON Performance Expert SAVE formatted data set.
4. Load the OMEGAMON PE SAVE data into DB2 tables.

B.3.1 Extracting and transforming DB2 trace data into the FILE format

We ran the batch JCL that is shown in Example B-5 to extract and to transform SMF DB2 accounting and statistics data into the OMEGAMON XE for DB2 PE FILE format. We obtained the accounting and statistics data in a sequential data set that we later use for the DB2 LOAD utility to load the data into DB2 Performance Database accounting and statistics tables.

Example B-5 OMEGAMON PE extract and transform DB2 trace data into FILE format

```
/* -----  
/*DOC Extract and transform accounting and statistics trace data  
/*DOC into Omegamon PE FILE format  
/* -----  
//DB2PM1 EXEC PGM=DB2PM,REGION=OM  
//STEPLIB DD DISP=SHR,DSN=<omhlq>.RKANMOD  
//INPUTDD DD DISP=SHR,DSN=SMF.DUMP.G0033V00  
//STFILDD1 DD DISP=(NEW,CATLG,DELETE),DSN=DB2R3.PM.STAT.FILE,  
//          SPACE=(CYL,(050,100),RLSE),  
//          UNIT=SYSDA,  
//          DATACLAS=COMP /*trigger DFSMS compression */  
//ACFILDD1 DD DISP=(NEW,CATLG,DELETE),DSN=DB2R3.PM.ACCT.FILE,  
//          SPACE=(CYL,(050,100),RLSE),  
//          UNIT=SYSDA,  
//          DATACLAS=COMP /*trigger DFSMS compression */  
//JOBSUMDD DD SYSOUT=A  
//DPMLOG DD SYSOUT=A  
//SYSOUT DD SYSOUT=A  
//SYSIN DD *  
GLOBAL  
    INCLUDE(SSID(DB1S)) TIMEZONE(-1)  
STATISTICS  
    FILE DDNAME(STFILDD1)  
ACCOUNTING  
    FILE DDNAME(ACFILDD1)  
EXEC
```

B.3.2 Extracting and transforming DB2 trace data into the SAVE format

We ran the batch JCL that is shown in Example B-6 to extract and to transform SMF DB2 accounting data into OMEGAMON XE for DB2 PE accounting SAVE format. We obtained the accounting data in a sequential data set, which we later use as input for the DB2 LOAD utility to load the data into DB2 Performance Database save accounting tables.

Example B-6 Extract and transform accounting SAVE format

```
//IDC01 EXEC PGM=IDCAMS
/* =====
/* Def Cluster source: RK02SAMP(DG0PJAMI)
/* =====
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE (DB2SMF.WASRB.ACCTLOAD) NONVSAM
SET MAXCC = 0
DELETE (DB2SMF.WASRB.ACCTSAVE ) CLUSTER
SET MAXCC = 0
DEFINE CLUSTER -
  (NAME(DB2SMF.WASRB.ACCTSAVE ) -
    CYL(100,40) -
    BUFFERSPACE(40960) -
    KEYS(255 0) -
    REUSE -
    RECORDSIZE(2800 4600) -
  ) -
  DATA (CISZ(8192)) -
  INDEX (CISZ(4096))
//SAVE02 EXEC PGM=DB2PM,REGION=0M
//STEPLIB DD DISP=SHR,DSN=OMEGA5RT.SC63.RKANMOD
//INPUTDD DD DISP=SHR,DSN=DB2SMF.WASRB.SC63.T4.SMFDB2
//ACSAVDD DD DISP=SHR,DSN=DB2SMF.WASRB.ACCTSAVE
//DPMLOG DD SYSOUT=A
//JOBSUMDD DD SYSOUT=A
//SYSOUT DD SYSOUT=A
//SYSIN DD *
GLOBAL
  INCLUDE(SUBSYSTEMID(D0Z*))
  TIMEZONE(+4)
ACCOUNTING
  /* 1 minute interval */
  REDUCE INTERVAL(1) BOUNDARY(60)
  SAVE
EXEC
//CONVO3 EXEC PGM=DG0PMICO,PARM=CONVERT,COND=(0,NE)
//STEPLIB DD DISP=SHR,DSN=OMEGA5RT.SC63.RKANMOD
//SYSPRINT DD SYSOUT=*
//INPUT DD DSN=DB2SMF.WASRB.ACCTSAVE,DISP=SHR
//OUTPUT DD DSN=DB2SMF.WASRB.ACCTLOAD,
//          DISP=(NEW,CATLG,DELETE),
//          SPACE=(CYL,(200,10),RLSE),
//          UNIT=SYSDA,
//          DCB=(RECFM=VB,LRECL=9072,BLKSIZE=9076)
```

B.3.3 Preparing a load job

Loading data into DB2 tables requires that a DB2 load utility batch JCL be available for batch job submission. To prepare the required batch JCL, we performed the following tasks:

- Consolidate and customize load utility control statements for loading PDB accounting and statistics data.
- Provide batch JCL for DB2 load utility job submission.

Load utility control statements

We ran the batch JCL that is shown in Example B-7 and Example B-8 to merge the load utility control statements that we referenced in “Accounting table DDL and load statements” on page 300 and in “Statistics tables DDL and load statements” on page 301 into a consolidated data set.

Example B-7 Merge statistics and accounting file load utility control statements

```
//S1GEN    EXEC PGM=IEBGENER
//SYSUT1   DD *
--OPTIONS PREVIEW
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOSLBUF)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOSLDDF)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOSLGBP)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOSLGEN)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOSLSET)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOALFBU)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOALFDF)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOALFGE)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOALFGP)
//         DD DISP=SHR,DSN=<omh1q>.TK02SAMP(DGOALFPK)
//SYSUT2   DD DISP=SHR,DSN=DB2R3.PM.CNTL($08LOATB)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
```

Example B-8 Merge accounting save load utility control statements

```
//COPY1    EXEC PGM=IEBGENER,DYNAMNBR=20,TIME=1440
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT2   DD DISP=SHR,DSN=DB2R3.SG.PM.LOAD(LOADACCS)
//SYSUT1   DD *
-- OPTIONS PREVIEW
// DD DISP=SHR,DSN=<LOADHLQ>.RK02SAMP(DGOALSBU)
// DD DISP=SHR,DSN=<LOADHLQ>.RK02SAMP(DGOALSDF)
// DD DISP=SHR,DSN=<LOADHLQ>.RK02SAMP(DGOALSGE)
// DD DISP=SHR,DSN=<LOADHLQ>.RK02SAMP(DGOALSGP)
// DD DISP=SHR,DSN=<LOADHLQ>.RK02SAMP(DGOALSPK)
// DD DISP=SHR,DSN=<LOADHLQ>.RK02SAMP(DGOALSRF)
```

We then modified the generated data set to reflect the table qualifier and the appropriate input DD statement and implemented the load utility options that we needed to use. Here are the load options that we use:

- ▶ RESUME YES
- ▶ LOG NO
- ▶ KEEPDICTIONARY
- ▶ NOCOPYPEND

B.3.4 Loading accounting and statistics tables

We use the DB2 load utility to load the data that is referred to in B.3.1, “Extracting and transforming DB2 trace data into the FILE format” on page 306 and in B.3.2, “Extracting and transforming DB2 trace data into the SAVE format” on page 307 into the PDB accounting and statistics tables.

B.3.5 Maintaining PDB tables

Your DB2 installation regularly performs Image Copy, Runstats, and Reorg on your tables to comply with your recovery requirements and to support good query performance.

Image copy

We ran the batch JCL that is shown in Example B-9 to perform image copy on PDB accounting and statistics tables.

Example B-9 Image copy batch JCL

```
//COPY EXEC DSNUPROC,SYSTEM=DB1S,
//          LIB='SYS1.DSNDB1S.SDSNLOAD',
//          UID='PDBCOPY'
//DSNUPROC.SYSIN DD *
--OPTIONS PREVIEW
  TEMPLATE TPDB DSN DB1SIC.IC.&DB..&TS..D&DATE..T&TIME.
  DATACLAS COMP
  LISTDEF LPDB INCLUDE TABLE PDB.*
  COPY LIST LPDB COPYDDN(TPDB) CHangelIMIT(0) PARALLEL
```

Runstats

Because we configured the administrative scheduler to perform autonomic statistics maintenance on non-catalog table spaces, there was no need to plan any further Runstats activity.

Reorg

We ran the batch JCL that is shown in Example B-10 to perform Reorg on PDB accounting and statistics tables.

Example B-10 Reorg batch JCL

```
//REORG1 EXEC DSNUPROC,SYSTEM=DB1S,
//          UID='PDBREO'
//DSNUPROC.SYSIN DD *
--OPTIONS PREVIEW
  LISTDEF LPDB INCLUDE TABLE PDB.*
  TEMPLATE TCOPY DSN DB1SIC.IC.&DB..&TS..D&DATE..T&TIME.
```

```

          DATACLAS COMP
TEMPLATE TSYSUT1 DSN(DB1SIC.&DB..&TS..&UTILID..SYSUT1)
          DISP(NEW,DELETE,KEEP)
          DATACLAS COMP
TEMPLATE TSORTOUT DSN(DB1SIC.&DB..&TS..&UTILID..SORTOUT)
          DISP(NEW,DELETE,KEEP)
          DATACLAS COMP
TEMPLATE TPUNCH   DSN(DB1SIC.&DB..&TS..&UTILID..PUNCH   )
          DISP(NEW,DELETE,KEEP)
          DATACLAS COMP
TEMPLATE TSYSREC  DISP(NEW,DELETE,KEEP)
          DSN(DB1SIC.&DB..&TS..&UTILID..SYSREC)
          DATACLAS COMP

REORG TABLESPACE LIST LPDB
LOG NO
SHRLEVEL REFERENCE
SORTDATA
SORTDEVT SYSDA
SORTNUM 4
UNLDDN TSYSREC
WORKDDN(TSYSUT1,TSORTOUT)
STATISTICS
COPYDDN(TCOPY)
PUNCHDDN(TPUNCH)

```

B.4 Sample query for application profiling

We created the DB2 SQL table UDF that is shown in Example B-11 to provide an interface for querying the DB2PMSACCT_GENERAL and DB2PMSACCT_BUFFER PDB tables for application profiling. The UDF receives two input parameters and joins DB2 general and buffer pool accounting information. The result is filtered by the DB2 client application information and the connection type (RRS or DRDA) to provide profiling information for a particular clientApplicationInformation for JDBC type 2 (connection type RRS) or for JDBC type 4 (connection type DRDA) applications.

Example B-11 OMEGAMON PE SQL table UDF

```

CREATE FUNCTION ACCOUNTING
(CLIENTAPPLICATION VARCHAR(128),
CONNTYPE           CHAR(8)      )
RETURNS TABLE (
    "DateTime"          VARCHAR(16)
  , "ClientApplication" VARCHAR(40)
  , "Elapsed"           DECIMAL(9,2)
  , "TotCPU"            DECIMAL(9,2)
  , "TotzIIP"           DECIMAL(9,2)
  , "DB2CPU"            DECIMAL(9,2)
  , "DB2zIIP"           DECIMAL(9,2)
  , "Commit"            INTEGER
  , "SQL"               INTEGER
  , "Locks"             INTEGER
  , "RowsFetched"        INTEGER
  , "RowsInserted"      INTEGER

```



```

, "RowsUpdated"          INTEGER
, "RowsDeleted"         INTEGER
, "GetPage"             INTEGER
, "AVG-Time"            DECIMAL(15, 6)
, "AVG-CPU"             DECIMAL(15, 6)
, "Time/SQL"            DECIMAL(15, 6)
, "CPU/SQL"             DECIMAL(15, 6)
, "AVG-SQL"             DECIMAL(15, 6)
, "LOCK/Tran"           DECIMAL(15, 6)
, "LOCK/SQL"            DECIMAL(15, 6)
, "GETP/Tran"           DECIMAL(15, 6)
, "GETP/SQL"            DECIMAL(15, 6)
)
LANGUAGE SQL READS SQL DATA NO EXTERNAL ACTION
DETERMINISTIC
RETURN
WITH
Q1 AS
(SELECT
  substr(char(INTERVAL_TIME),1,16      ) AS DATETIME
, CLIENT_TRANSACTION
, DECIMAL(CLASS1_ELAPSED,9,2           ) AS ELAPSED
, DECIMAL(CLASS1_CPU_NNESTED+CLASS1_CPU_STPROC+CLASS1_CPU_UDF
+CLASS1_IIP_CPU,9,2                   ) AS CPU
, DECIMAL(CLASS1_IIP_CPU,9,2           ) AS ZIIP
, DECIMAL(CLASS2_CPU_NNESTED+CLASS2_CPU_STPROC+CLASS2_CPU_UDF
+CLASS2_IIP_CPU,9,2                   ) AS DB2CPU
, DECIMAL(CLASS2_IIP_CPU,9,2           ) AS DB2ZIIP
, DECIMAL(COMMIT,9,2                   ) AS COMMIT
, DECIMAL(SELECT+INSERT+UPDATE+DELETE+FETCH+MERGE,9,2) AS SQL
, DECIMAL(LOCK_REQ,9,2                 ) AS LOCKS
, INTEGER(ROWS_FETCHED                 ) AS ROWS_FETCHED
, INTEGER(ROWS_INSERTED                ) AS ROWS_INSERTED
, INTEGER(ROWS_UPDATED                 ) AS ROWS_UPDATED
, INTEGER(ROWS_DELETED                 ) AS ROWS_DELETED
FROM DB2PMSACCT_GENERAL
WHERE CONNECT_TYPE = ACCOUNTING.CONNNTYPE
AND CLIENT_TRANSACTION = ACCOUNTING.CLIENTAPPLICATION
AND COMMIT > 0 ),
Q2 AS
(SELECT
  substr(char(INTERVAL_TIME),1,16      ) AS DATETIME
, CLIENT_TRANSACTION
, decimal(SUM(BP_GETPAGES),9,2         ) AS GETPAGE
FROM DB2PMSACCT_BUFFER
WHERE CONNECT_TYPE = ACCOUNTING.CONNNTYPE
AND CLIENT_TRANSACTION = CLIENTAPPLICATION
GROUP BY substr(char(INTERVAL_TIME),1,16), CLIENT_TRANSACTION ),
Q3 AS
(SELECT Q1.*, Q2.GETPAGE FROM Q1, Q2 WHERE
  (Q1.DATETIME,Q1.CLIENT_TRANSACTION) =
  (Q2.DATETIME,Q2.CLIENT_TRANSACTION) AND Q1.SQL > 0),
Q4 AS
(SELECT Q3.*,
  ELAPSED/COMMIT as "AVG-Time",

```

```

        CPU/COMMIT as "AVG-CPU",
    ELAPSED/SQL    as "Time/SQL",
        CPU/SQL    as "CPU/SQL",
        SQL/COMMIT as "AVG-SQL",
    LOCKS/COMMIT  as "LOCK/Tran",
    LOCKS/SQL     as "LOCK/SQL",
    GETPAGE/COMMIT as "GETP/Tran",
    GETPAGE/SQL   as "GETP/SQL"
FROM Q3)
SELECT * FROM Q4

```

For each interval, the UDF returns the following information:

- ▶ DateTime: Interval date and time
- ▶ ClientApplication: Client application name
- ▶ Elapsed: Total elapsed time
- ▶ TotCPU: Total CPU time, including the time that was processed on a zIIP processor
- ▶ TotzIIP: Total zIIP processor time
- ▶ DB2CPU: DB2 part of the total CPU time
- ▶ DB2zIIP: DB2 part of the zIIP processor time
- ▶ Commit: Total number of commits
- ▶ SQL: Total number of SQL **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **FETCH**, and **MERGE** statements
- ▶ Locks: Total number of lock requests
- ▶ RowsFetched: Number of rows that were fetched
- ▶ RowsInserted: Number of rows that were inserted
- ▶ RowsUpdated: Number of rows that were updated
- ▶ RowsDeleted: Number of rows that were deleted
- ▶ GetPage: Number of getpage requests
- ▶ AVG-Time: Average elapsed time
- ▶ AVG-CPU: Average CPU time, including zIIP time
- ▶ Time/SQL: Average elapsed time per SQL
- ▶ CPU/SQL: Average CPU time per SQL
- ▶ AVG-SQL: Average number of SQL per commit
- ▶ LOCK/transaction: Average number of lock requests per commit
- ▶ LOCK/SQL: Average number of locks per SQL
- ▶ GETP/transaction: Average number of getpage requests per commit
- ▶ GETP/SQL: Average number of getpage requests per SQL

B.5 Using the UDF for application profiling

We used the query that is shown in Example B-12 to start the UDF for JDBC type 2 (connection type RRS) application profiling.

Example B-12 Starting UDF for JBC driver Type 4

```

select * from
    table(accounting('TraderClientApplication','RRS')) a
    order by "DateTime"      ;

```

DateTime	ClientApplication	Elapsed	To
2012-08-14-22.39	TraderClientApplication	11.96	
2012-08-14-22.41	TraderClientApplication	2417.46	3

B.6 Additional information

For more information about using the OMEGAMON Performance Expert PDB, see the following resources:

- ▶ Chapter 5, “Advanced reporting concepts”, in *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, Reporting User's Guide Version 5.1.0*, SH12-6927
- ▶ *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-72244

B.7 MEMU2 versus Performance Database fields

Table B-4 is a reference for the actual IFCID values, the fields in the Performance Database, and the fields which are used in the MEMU2 REXX exec. It could be used with the OMEGAMON PE Spreadsheet Generator function to create a .csv file which would mimic the output from MEMU.

Table B-4 IFCIDs in OMEGAMON PE and MEMU2

#	IFCID FIELD	PM FIELD	PDB COLUMN NAME	MEMU2 Description
1	-	DB2REL	DB2_REL	SSID
2	QWHSSSID	QWHSSSID	SUBSYSTEM_ID	Date
3	-	SDENDREC	END_REC_TSTAMP	Local time
4	QW0225AN	-	-	-
5	QW0225RG	QW0225RG	EXT_REG_SIZE_MAX	ASID DBM1 MVS ext region size (MB)
6	QW0225LO	QW0225LO	BIT_LOW_PRIVATE_24	ASID DBM1 MVS 24-bit low priv (MB)
7	QW0225HI	QW0225HI	BIT_HGH_PRIVATE_24	ASID DBM1 MVS 24-bit high priv (MB)
8	QW0225EL	QW0225EL	BIT_EXT_LOW_PRI_31	ASID DBM1 MVS 31-bit ext low priv (MB)
9	QW0225EH	QW0225EH	BIT_EXT_HGH_PRI_31	ASID DBM1 MVS 31-bit ext high priv (MB)
10	QW0225TP	QW0225TP	BIT_HGH_ADDR_PR_24	ASID DBM1 MVS 24-bit priv high address
11	QW0225EP	QW0225EP	BIT_HGH_ADDR_PR_31	ASID DBM1 MVS 31-bit priv high address
12	QW0225CR	QW0225CR	TOT_STORAGE_RESERV	ASID DBM1 Reserved for must complete (MB)
13	QW0225MV	QW0225MV	TOT_AMOUNT_FOR_MVS	ASID DBM1 Reserved for MVS (MB)
14	QW0225SO	QW0225SO	WARN_CUSH_TO_CONTR	ASID DBM1 Cushion warning (MB)
15	QW0225GS	QW0225GS	TOT_GETM_STCK_STOR	ASID DBM1 31-bit priv getmained stack (MB)
16	QW0225SU	QW0225SU	IN_USE_STACK_STOR	ASID DBM1 31-bit priv stack in use (MB)
17	QW0225VR	QW0225VR	TOT_VAR_STORAGE	ASID DBM1 31-bit priv variable pool (MB)
18	QW0225FX	QW0225FX	TOT_FIXED_STORAGE	ASID DBM1 31-bit priv fixed pool (MB)

#	IFCID FIELD	PM FIELD	PDB COLUMN NAME	MEMU2 Description
19	QW0225GM	QW0225GM	TOT_GETM_STORAGE	ASID DBM1 31-bit priv getmaind (MB)
20	QW0225AV	QW0225AV	AVAILABLE_STORAGE	ASID DBM1 31-bit priv available (MB)
21	QW0225SS	QW0225SS	AGENT_STACK_STOR	ASID DBM1 31-bit priv sys agent stack in use (MB)
22	QW0225VA	QW0225VA	A2GB_VAR_STORAGE	ASID DBM1 64-bit priv variable pool (MB)
23	QW0225FA	QW0225FA	A2GB_FIXED_STORAGE	ASID DBM1 64-bit priv fixed pool (MB)
24	QW0225GA	QW0225GA	A2GB_GETM_STORAGE	ASID DBM1 64-bit priv getmaind (MB)
25	QW0225SM	QW0225SM	A2GB_STOR_MGR_STOR	ASID DBM1 64-bit priv for SM ctrl str (MB)
26	QW0225RL	QW0225RL	REAL_STORAGE_FRAME	ASID DBM1 REAL in use for 31 and 64-bit priv (MB)
27	QW0225AX	QW0225AX	AUX_STORAGE_SLOT	ASID DBM1 AUX in use for 31 and 64-bit priv (MB)
28	QW0225HVPagesInReal	SW225VPR	A2GB_REAL_FRAME	ASID DBM1 REAL in use for 64-bit priv (MB)
29	QW0225HVAuxSlots	SW225VAS	A2GB_AUX_SLOT	ASID DBM1 AUX in use for 64-bit priv (MB)
30	QW0225HVGPageInReal	SW225GPR	A2GB_HWM_REAL_FRM	ASID DBM1 HWM REAL for 64-bit priv (MB)
31	QW0225HVGAuxSlots	SW225GAS	A2GB_HWM_AUX_SLOT	ASID DBM1 HWM AUX for 64-bit priv (MB)
32	QW0225PriStg_Real	SW225PSR	A2GB_REAL_FRAME_TS	ASID DBM1 REAL in use for 64-bit priv w/o BP (MB)
33	QW0225PriStg_Aux	SW225PSA	A2GB_AUX_SLOT_TS	ASID DBM1 AUX in use for 64-bit priv w/o BP (MB)
34	QW0225CTLTP	S225CTLTP	QW0225CTLTP	ASID DBM1 CTLTP (S)
35	QW0225CTLS	S225CTLS	QW0225CTLS	ASID DBM1 CTLS (S)
36	QW0225AN	SW0225NM	DIST_ADDRESS_SPACE	-
37	QW0225RG	QW0225RG	DIST_EXT_REG_SIZE	ASID DIST MVS ext region size (MB)
38	QW0225LO	QW0225LO	DIST_LOW_PRIV_24	ASID DIST MVS 24-bit low priv (MB)
39	QW0225HI	QW0225HI	DIST_HIGH_PRIV_24	ASID DIST MVS 24-bit high priv (MB)
40	QW0225EL	QW0225EL	DIST_EXT_LOW_PR_31	ASID DIST MVS 31-bit ext low priv (MB)
41	QW0225EH	QW0225EH	DIST_EXT_HGH_PR_31	ASID DIST MVS 31-bit ext high priv (MB)
42	QW0225TP	QW0225TP	DIST_HGH_ADR_PR_24	ASID DIST MVS 24-bit priv high address
43	QW0225EP	QW0225EP	DIST_HGH_ADR_PR_31	ASID DIST MVS 31-bit priv high address
44	QW0225CR	QW0225CR	DIST_STOR_RESERV	ASID DIST Reserved for must complete (MB)
45	QW0225MV	QW0225MV	DIST_AMOUN_FOR_MVS	ASID DIST Reserved for MVS (MB)
46	QW0225SO	QW0225SO	DIST_CUSH_TO_CONTR	ASID DIST Cushion warning (MB)

#	IFCID FIELD	PM FIELD	PDB COLUMN NAME	MEMU2 Description
47	QW0225GS	QW0225GS	DIST_GETM_STK_STOR	ASID DIST 31-bit priv getmaind stack (MB)
48	QW0225SU	QW0225SU	DIST_USE_STK_STOR	ASID DIST 31-bit priv stack in use (MB)
49	QW0225VR	QW0225VR	DIST_VAR_STORAGE	ASID DIST 31-bit priv variable pool (MB)
50	QW0225FX	QW0225FX	DIST_FIXED_STORAGE	ASID DIST 31-bit priv fixed pool (MB)
51	QW0225GM	QW0225GM	DIST_GETM_STORAGE	ASID DIST 31-bit priv getmaind (MB)
52	QW0225AV	QW0225AV	DIST_AVAIL_STORAGE	ASID DIST 31-bit priv available (MB)
53	QW0225SS	QW0225SS	DIST_AG_STACK_STOR	ASID DIST 31-bit priv sys agent stack in use (MB)
54	QW0225VA	QW0225VA	A2GB_DIST_VAR_STOR	ASID DIST 64-bit priv variable pool (MB)
55	QW0225FA	QW0225FA	A2GB_DIST_FIX_STOR	ASID DIST 64-bit priv fixed pool (MB)
56	QW0225GA	QW0225GA	A2GB_DIST_GETM_STO	ASID DIST 64-bit priv getmaind (MB)
57	QW0225SM	QW0225SM	A2GB_DIST_STOR_MGR	ASID DIST 64-bit priv for SM ctrl str (MB)
58	QW0225RL	QW0225RL	DIST_REAL_FRAME	ASID DIST REAL in use for 31 and 64-bit priv (MB)
59	QW0225AX	QW0225AX	DIST_AUX_SLOT	ASID DIST AUX in use for 31 and 64-bit priv (MB)
60	QW0225HVPagesInReal	SW225VPR	A2GB_DIST_REAL_FRM	ASID DIST REAL in use for 64-bit priv (MB)
61	QW0225HVAuxSlots	SW225VAS	A2GB_DIST_AUX_SLOT	ASID DIST AUX in use for 64-bit priv (MB)
62	QW0225HVGPageInReal	SW225GPR	A2GB_HWM_DIST_REAL	ASID DIST HWM REAL for 64-bit priv (MB)
63	QW0225HVGAuxSlots	SW225GAS	A2GB_HWM_DIST_AUX	ASID DIST HWM AUX for 64-bit priv (MB)
64	QW0225PriStg_Real	SW225PSR	A2GB_DIST_REALF_TS	ASID DIST REAL in use for 64-bit priv w/o BP (MB)
65	QW0225PriStg_Aux	SW225PSA	A2GB_DIST_AUXS_TS	ASID DIST AUX in use for 64-bit priv w/o BP (MB)
66	QW0225CTLTP	S225CTLTP	DIST_QW0225CTLTP	ASID DIST CTLTP (S)
67	QW0225CTLS	S225CTLS	DIST_QW0225CTLS	ASID DIST CTLS (S)
68	QW0225AT	QW0225AT	ACT_ALLIED_THREADS	Active threads
69	QW0225DB	QW0225DB	ACTIV_DISCON_DBATS	Active and disc DBATs
70	QW0225CE	QW0225CE	NUMB_CAST_ENGINES	Castout engines
71	QW0225DW	QW0225DW	NUMB_DEFF_WRT_ENG	Deferred write engines
72	QW0225GW	QW0225GW	NUMB_GBP_WRT_ENG	GBP write engines
73	QW0225PF	QW0225PF	NUMB_PREFETCH_ENG	Prefetch engines
74	QW0225PL	QW0225PL	NUMB_P_LOCK_EXIT_E	P-lock/ notify exit engines
75	QW0225PT	QW0225PT	PAR_CHILD_THREADS	Active parallel child threads
76	QW0225EC	QW0225EC	EXTENDED_CSA_SIZE	MVS ECSA size (MB)

#	IFCID FIELD	PM FIELD	PDB COLUMN NAME	MEMU2 Description
77	QW0225FC	QW0225FC	COMMON_FIXED_STOR	31-bit common fixed pool (MB)
78	QW0225VC	QW0225VC	COMMON_VAR_STOR	31-bit common variable pool (MB)
79	QW0225GC	QW0225GC	COMMON_GETM_STOR	31-bit common getmaind (MB)
80	QW0225FCG	SW225FCG	A2GB_COMMON_FIXED	64-bit common fixed pool (MB)
81	QW0225VCG	SW225VCG	A2GB_COMMON_VAR	64-bit common variable pool (MB)
82	QW0225GCG	SW225GCG	A2GB_COMMON_GETM	64-bit common getmaind (MB)
83	QW0225SMC	SW225SMC	A2GB_COMM_STOR_MGR	64-bit common for SM ctrl str (MB)
84	QW0225SV	QW0225SV	A2GB_SHARED_VAR	64-bit shared variable pool (MB)
85	QW0225SF	QW0225SF	A2GB_SHARED_FIXED	64-bit shared fixed pool (MB)
86	QW0225SG	QW0225SG	A2GB_SHARED_GETM	64-bit shared getmaind (MB)
87	QW0225SMS	SW225SMS	A2GB_SHR_STOR_MGR	64-bit shared for SM ctrl str (MB)
88	QW0225GSG_SYS	SW225GSY	A2GB_SHR_AG_SYS_ST	64-bit shared system agent stack (MB)
89	QW0225SUG_SYS	SW225SSY	A2GB_SHR_AG_SYS_IU	64-bit shared system agent stack in use (MB)
90	QW0225GSG	SW225GSG	A2GB_SHR_AG_NSY_ST	64-bit shared non-system agent stack (MB)
91	QW0225SUG	SW225SUG	A2GB_SHR_AG_NSY_IU	64-bit shared non-system agent stack in use (MB)
92	QW0225CTGP	S225CTGP	QW0225CTGP	CTGP (S)
93	QW0225DISC	S225DISC	QW0225DISC	DISC (S)
94	QW0225SHRNMO MB	SW225SMO	SHR_MEMORY_OBJECTS	Shared memory objects alloc - LPAR
95	QW0225SHRPAGE S	SW225SPG	A2GB_SHR_MEM_PAGES	64-bit shared alloc - LPAR (MB)
96	QW0225SHRGRAY TES	SW225SGB	A2GB_HWM_SHR_STOR	64-bit shared alloc - HWM - LPAR (MB)
97	QW0225SHRINRE AL	SW225SRL	A2GB_SHR_PGS_BACKD	REAL in use for 64-bit shared - LPAR (MB)
98	QW0225SHRAUXS LOTS	SW225SAX	A2GB_SHR_AUX_SLOTS	AUX in use for 64-bit shared - LPAR (MB)
99	QW0225SHRPAGE INS	SW225SPI	A2GB_SHR_PGS_PGD_I	64-bit shared pages paged in - LPAR
100	QW0225SHRPAGE OUTS	SW225SPO	A2GB_SHR_PGS_PGD_O	64-bit shared pages paged out - LPAR
101	QW0225ShrStg_R eal	SW225SSR	A2GB_SHR_REALF_TS	REAL in use for 64-bit shared (MB)
102	QW0225ShrStg_Au x	SW225SSA	A2GB_SHR_AUXS_TS	AUX in use for 64-bit shared (MB)
103	QW0225ShrStkStg _Real	SW225KSR	A2GB_SHR_REALF_STK	REAL in use for 64-bit shared stack (MB)
104	QW0225ShrStkStg _Aux	SW225KSA	A2GB_SHR_AUXS_STK	AUX in use for 64-bit shared stack (MB)

#	IFCID FIELD	PM FIELD	PDB COLUMN NAME	MEMU2 Description
105	QW0225ComStg_Real	SW225CSR	A2GB_COMMON_REALF	REAL in use for 64-bit common (MB)
106	QW0225ComStg_Aux	SW225CSA	A2GB_COMMON_AUXS	AUX in use for 64-bit common (MB)
107	QW0225_WARN	S225WARN	QW0225_WARN	WARN (S)
108	QW0225_REALAVAIL	S225RLAV	QW0225_REALAVAIL	REALAVAIL (MB) (S)
109	QW0225_REALAVAILLO	S225RLLO	QW0225_REALAVAILLO	REALAVAILLO (MB) (S)
110	QW0225_REALAVAILLOK	S225RLOK	QW0225_REALAVAILLOK	REALAVAILLOK (MB) (S)
111	QW0225_ESQAS	S225ESQS	QW0225_ESQAS	ESQAS (MB) (S)
112	QW0225_ESQA_Alloc	S225ESQA	QW0225_ESQA_ALLOC	ESQA Alloc (MB) (S)
113	QW0225_ESQA_HWM	S225ESQH	QW0225_ESQA_HWM	ESQA HWM (MB) (S)
114	QW0225_ECSCA_Alloc	S225ECSCA	QW0225_ECSCA_ALLOC	ECSCA Alloc (MB) (S)
115	QW0225_ECSCA_HWM	S225ECSCA	QW0225_ECSCA_HWM	ECSCA HWM (MB) (S)
116	QW0225_ECSCA_Conv	S225ECSC	QW0225_ECSCA_CONV	ECSCA Conv (MB) (S)
117	QW0225SC	QW0225SC	LOC_DYN_CACHE_POOL	xPROC for dynamic SQL - Total (MB)
118	QW0225LS	QW0225LS	IN_USE_LOCAL_CACHE	xPROC for dynamic SQL - In use (MB)
119	QW0225SX	QW0225SX	LOC_STATIC_CACHE	xPROC for static SQL - Total (MB)
120	QISEKSPA	QISEKSPA	XPROC_STORAGE	xPROC for static SQL - In use (MB)
121	QW0225HS	QW0225HS	HWM_STOR_THREAD	xPROC for dynamic SQL - HWM (MB)
122	QW0225LC	QW0225LC	STMT_IN_LOC_CACHE	Stmt in 64-bit agent local pools
123	QW0225HC	QW0225HC	HWM_STOR_STMT_CNT	Stmt in 64-bit agent local pools - HWM
124	QW0225L2	QW0225L2	A2GB_IN_USE_LOC_C	Stmt cache in 64-bit agent local pools (MB)
125	QW0225H2	QW0225H2	A2GB_HWM_STOR_THR	HWM for alloc stmt cache storage (MB)
126	QW0225HT	QW0225HT	HWM_STOR_TIME	Time at HWM (UTC)
127	QW0225S2	QW0225S2	A2GB_LOC_DYN_CACHE	64-bit stmt cache blocks (MB)
128	QW0225F1	QW0225F1	QW0225F1	QW0225F1 (S)
129	QW0225F2	QW0225F2	QW0225F2	QW0225F2 (S)
130	QW0225AL	QW0225AL	TOT_AGENT_LOC_STRG	31-bit agent local (MB)
131	QW0225AS	QW0225AS	TOT_AGENT_SYS_STRG	31-bit system agent (MB)
132	QW0225ALG	SW225ALG	A2GB_SHR_AG_LOCAL	64-bit agent local (MB)
133	QW0225ASG	SW225ASG	A2GB_SHR_AG_SYSTEM	64-bit system agent (MB)

#	IFCID FIELD	PM FIELD	PDB COLUMN NAME	MEMU2 Description
134	QW0225BB	QW0225BB	BUFFER_MGR_STORAGE	BM control blocks (MB)
135	QW0225RP	QW0225RP	RID_POOL	RID pool (MB)
136	QW0225CD	QW0225CD	COMPR_DICTIONARY	Comp dict (MB)
137	QDBPVPSZ	-	-	Virtual buffer pools - Defined (MB)
138	QBSTVPL	SVPOOL1	VIRTUAL_BUFFERPOOL	Virtual buffer pools - Allocated (MB)
139	-	SBSTVPL2	VIRT_POOL_CNTL_BLK	VPOOL control blocks (MB)
140	QISEDDBD	QISEDDBD	DBD_PAGES	EDM DBD pool in use (MB)
141	QISEDYNP	QISEDYNP	PGS_USED_STMT_POOL	EDM stmt pool in use (MB)
142	QISESKCT	QISESKCT	SKCT_PAGES	EDM skel pool in use for SKCT (MB)
143	QISESKPT	QISESKPT	SKPT_PAGES	EDM skel pool in use for SKPT (MB)
144	QISESQCB	QISESQCB	PLAN_STORAGE	31-bit priv alloc to plans (MB)
145	QISESQKB	QISESQKB	PKG_STORAGE	31-bit priv alloc to packages (MB)
146	QISESQCA	QISESQCA	A2GB_PLAN_STORAGE	64-bit storage alloc to plans (MB)
147	QISESQKA	QISESQKA	A2GB_PKG_STORAGE	64-bit storage alloc to packages (MB)
148	QSSTCONT	QSSTCONT	CONTRACTION_ISSUED	Full system contraction
149	QSSTCRIT	QSSTCRIT	SHORTAGES_RAISED	Storage critical
150	QSSTABND	QSSTABND	ABENDS_ISSUED	Abends due to insufficient storage

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only:

- ▶ *DB2 11 for z/OS Technical Overview*, SG24-8180
- ▶ *DB2 for z/OS and WebSphere Integration for Enterprise Java Applications*, SG24-8074
- ▶ *DB2 9 for z/OS: Distributed Functions*, SG24-6952
- ▶ *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322
- ▶ *DB2 10 for z/OS Technical Overview*, SG24-7892
- ▶ *DB2 PM Usage Guide*, SG24-2584
- ▶ *A Deep Blue View of DB2 Performance IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-7224
- ▶ *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725
- ▶ *DB2 for z/OS and WebSphere Integration for Enterprise Java Applications*, SG24-8074
- ▶ *System Programmer's Guide to: Workload Manager*, SG24-6472-03
- ▶ *Effective zSeries Performance Monitoring Using Resource Measurement Facility*, SG24-6465
- ▶ *DB2 9 for z/OS: Buffer Pool Monitoring and Tuning*, REDP-4604
- ▶ *OS/390 Workload Manager Implementation and Exploitation*, SG24-5326

You can search for, view, download or order these documents and other Redbooks publications, Redpaper publications, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- ▶ *z/OS V1R13.0 DFSMS: Managing Catalogs*, SC26-7409-11
- ▶ *DB2 11 for z/OS Administration Guide*, SC19-4050
- ▶ *DB2 11 for z/OS Application Programming and SQL Guide*, SC19-4051
- ▶ *DB2 11 for z/OS Application Programming Guide and Reference for Java*, SC19-4052
- ▶ *DB2 11 for z/OS Codes*, GC19-4053
- ▶ *DB2 11 for z/OS Command Reference*, SC19-4054

- ▶ *DB2 11 for z/OS Data Sharing: Planning and Administration*, SC19-4055
- ▶ *DB2 11 for z/OS Installation and Migration*, SC19-4056
- ▶ *DB2 11 for z/OS Internationalization Guide*, SC19-4057
- ▶ *Introduction to DB2 for z/OS*, SC19-4058
- ▶ *DB2 11 for z/OS DB2 11 for z/OS IRLM Messages and Codes for IMS and DB2 for z/OS*, GC19-2666
- ▶ *DB2 11 for z/OS Managing Performance*, SC19-4060
- ▶ *DB2 11 for z/OS Managing Security*, SC19-4061
- ▶ *DB2 11 for z/OS Messages*, GC19-4062
- ▶ *DB2 11 for z/OS ODBC Guide and Reference*, SC19-4063
- ▶ *DB2 11 for z/OS pureXML Guide*, SC19-4064
- ▶ *DB2 11 for z/OS RACF Access Control Module Guide*, SC19-4065
- ▶ *DB2 11 for z/OS SQL Reference*, SC19-4066
- ▶ *DB2 11 for z/OS Utility Guide and Reference*, SC19-4067
- ▶ *DB2 11 for z/OS What's New?*, GC19-4068
- ▶ *DB2 11 for z/OS Diagnosis Guide and Reference*, LY37-3222
- ▶ *z/OS Resource Measurement Facility User's Guide Version 2 Release 1*, SC34-2664
- ▶ *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, Reporting User's Guide*, SH12-6927

Online resources

These websites are also relevant as further information sources:

- ▶ Capacity Planning for zAAP and zIIP Specialty Engines :
<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD103548>
- ▶ CFSizer:
<http://www-947.ibm.com/systems/support/z/cfsizer/>
- ▶ What is DB2 Accounting Class 2 Not Accounted Time?
<http://www.ibm.com/support/docview.wss?uid=swg21045823>

Help from IBM

IBM Support and downloads:

ibm.com/support

IBM Global Services:

ibm.com/services

Index

Numerics

125 317
22 64, 66, 96, 146, 171, 219, 254, 312
63 64, 66, 86, 175, 247, 315
96 64, 66, 98, 174–175, 312, 316

A

access 8, 12, 15, 20, 33, 41, 46, 58, 67, 74, 79, 87, 90, 110, 113, 125, 130, 155, 167, 180, 190, 202–203, 218, 232, 244, 291
access path 38, 73–74, 99, 131, 177, 224, 232, 235, 254, 292
ACCOUNTING 218, 281, 306–307
Accounting 14, 93, 204, 206, 218, 246, 251, 266, 271, 278, 282, 299
accounting report 50, 55–56, 106, 137, 218, 225, 253, 258–259, 279
ALTER BUFFERPOOL 141–142
APAR 23, 41, 56, 79, 97, 114, 141, 170, 190, 239, 292
application 5, 11, 16, 24, 32–33, 55, 57, 61–62, 78, 84, 89, 119, 122, 130, 153, 180, 201, 218, 231, 233, 243–244, 259, 268, 279
argument 110
Assign 107
attribute 253, 292
Audit 15, 299
authorization ID 75
AVG 34, 158, 219, 233, 311

B

base table 147
batch job 308
BETWEEN 72
bind option 70, 157
BIT 93, 187
BLKSIZE 307
BLWLINTHD 34
BLWLTRPCT 34
BSDS 58
buffer pool 7, 14, 27, 59, 79, 94, 112, 125–126, 170, 180, 187, 234, 243–244, 292
 activity 173
 BP0 126
 change 80, 239
 complete scans 80
 directory 126, 255
 manager 174
 name 187
 scan 80
 setting 59, 130, 142, 174
 size 172, 187, 294
 space 103, 128, 187, 235, 254–255
 storage 126, 140, 172, 187

 support 140, 173
Buffer pools 235
buffer pools 103, 125, 170, 180, 211, 228, 232, 247, 254, 287

C

CACHE 64–65, 143–144, 239, 252, 284
catalog table 128
CCSID 21, 302, 305
CF 45, 90, 143, 145, 219, 233, 247, 259
CICS 8, 30, 39–40, 57, 180–181, 199, 202, 205–206, 218, 281
class 14, 22, 30, 80, 90, 112, 118, 134, 180, 204, 206, 219, 221, 235, 244, 247, 257–258, 266, 278
Class 2 55–56, 138, 212–213, 265, 267, 278
CLI 71, 157
CLUSTER 103, 249, 253, 260, 307
CM 284
CMTSTAT 155
COLLID 72, 165, 286
COMMIT 311
Commit 310
components 29, 46, 98, 143, 179, 260–261, 279
compression 23, 88
CONDBAT 155
condition 85, 95, 145, 156
CONNECT 82, 220, 281
COPY 21, 244, 309
cost 4, 6, 9, 15, 24, 58, 67, 80, 83, 94, 110, 113, 126, 138, 156, 162, 181, 186, 211, 236, 244, 262
COUNT 65, 126, 141, 219
CPU 311
CPU overhead 93, 122
CPU time 53–54, 96, 122, 128, 136, 154, 180, 210, 216, 221–222, 249, 266, 279
CREATE 302, 305, 310
cross invalidation 146–147
CS 94, 246
CSV 10
CT 64, 108, 188
CTHREAD 105, 180, 193–194
CURRENT EXPLAIN MODE 73
CURRENT SCHEMA 303–304
CURRENT SQLID 72, 74, 220

D

data 3, 14–15, 29, 43, 56, 58, 67, 71, 77, 83, 89, 113, 119, 126, 153, 180–181, 203, 218, 231–232, 244, 259, 265, 279, 282, 292
data page 94, 146, 249
data set 47, 58, 79, 88, 94, 148, 187, 191, 231–232, 247, 252, 259, 282, 298
 maximum number 79, 191
data sharing 9, 18, 32, 78, 84–85, 90, 137, 146, 148, 159,

- 238–239, 243, 247, 250, 252, 261, 291–292, 294
- data space 259
- database access thread 162
- DATE 65, 309
- DB2 xxi, 1, 3, 13, 29–30, 54, 62, 78, 83–84, 89, 109, 125, 130, 153, 156, 167, 179–180, 199, 201, 217–218, 231–232, 244, 258, 265, 278, 284, 292–293, 297
 - Statistics trace 14
 - tools xxi
- DB2 10 xxi, 8, 17, 23, 35, 54, 62–63, 80, 85, 91, 111–112, 128, 134, 156, 170, 179–180, 220, 239, 249, 251, 293
 - address 54–55, 112, 180–181
 - base 142, 192
 - break 54
 - change 73, 135, 194
 - data 23, 85, 104, 140, 157, 181
 - environment 65, 157
 - function 190
 - group 122
 - make 54
 - NFM 71
 - page fix 180
 - running 54, 56, 96, 121
 - SQL 71, 73, 122, 172, 188
 - table spaces 103
 - track 181–182
 - use 23, 75, 103, 134, 140, 157, 170, 181
- DB2 8 249
- DB2 9 xxi–xxii, 33, 36, 55–56, 62, 88, 91, 111–112, 126, 128, 159, 168, 180, 187, 239
 - DB2 10 56, 58, 64, 112, 122, 170, 187, 239
 - dynamic prefetch 59, 135
 - package 128, 188
 - system 58, 93, 112, 139
- DB2 client 310
- DB2 Connect xxii, 157
- DB2 for z/OS 302, 305
- DB2 member 32, 74, 79, 99, 146, 148, 159, 247
- DB2 Performance Expert 299, 313, 320
- DB2 Performance Monitor 8
- DB2 Query Monitor 6
- DB2 SQL Performance Analyzer 6, 8
- DB2 subsystem 3, 9, 13, 29, 32, 34, 51, 91, 154, 184, 187, 244, 293
- DB2 system 32, 78, 91, 111, 239, 259
- DB2 V8 36, 85, 111, 135, 137
 - system 111
- DB2 V9 62
- DBAT 33, 36, 110, 155, 180, 278
- DBD 62–63, 88, 90, 318
- DBD01 67
- DBET 143
- DBID/OBID 261–262
- DBM1 address space 59, 69, 113, 121, 138, 155, 181, 196
- DBNAME 127
- DD SYSOUT 304–305
- DDF 14, 31, 33–34, 54–56, 104, 110, 112, 153, 156, 208–209, 211, 218, 250–251, 259, 277–278, 300–302

- DDF command 157
- DDL 20, 78, 88, 156, 245
- DEADLOCK 91, 244
- decimal 56, 294
- decomposition 189
- default value 23, 96
- deferred write 55, 104, 121, 129–130, 192, 241
- Deflate algorithm 24
- DEGREE 220, 293
- DELETE 21, 58, 64, 94, 220, 260, 269, 285, 306–307
- delete 46, 85, 104, 128, 137, 239, 259
- DFSMS 88, 262, 306
- disk I/O 4
- DISPLAY THREAD 36, 194, 281
- DIST 19, 32, 53, 55, 105, 122, 182–183, 287, 314
- Distributed xxii, 159, 218
- DRDA xxii, 56, 58, 110, 113, 122, 153–154, 156, 207, 259, 301, 310
- DSN6SPRM 294
- DSNAME 11
- DSNJ031I 97
- DSNTEP2 22, 285, 305
- DSNTIPN 23
- DSNTSMFD 23
- DSNZPARM 18, 23, 59, 65, 69, 78, 80, 85, 96–97, 155–156, 176, 282, 284
 - CACHEDYN 69
 - CLOSET 78
 - DSMAX 79
 - EDMPOOL 65
 - MAXKEEPD 70
 - MAXTEMPS 176
 - OUTBUFF 84
 - PCLOSEN 59, 78
 - PCLOSET 59
 - REAL_STORAGE_MANAGEMENT 59
 - RID POOL SIZE 178
 - SMFCOMP 23
 - WFSTGUSE_AGENT_THRESHOLD 176
 - WFSTGUSE_SYSTEM_THRESHOLD 176
- DWQT 129
- dynamic SQL 9, 61, 154, 174, 190, 203, 219, 284, 292
- dynamic SQL statement 67, 284
- Dynamic statement cache 105, 251

E

- EBCDIC 302, 305
- EDM pool 61–62, 105, 251
- EDMPOOL 63, 191, 193
- efficiency 7, 24, 96, 129, 132, 162, 171, 221
- element 202
- environment xxi, 5, 30, 32, 40, 56, 59, 78, 88, 91, 114, 122, 143, 154, 174, 218, 239, 247, 267, 291, 298, 301–302
- ETL 297
- EVALUNC 94
- EVENTS 226, 233, 258, 266, 281
- Exceptions 299
- EXEC 21, 82, 218, 285, 303, 305
- EXPLAIN 12, 72–73, 219, 285, 293–294

QUERYNO 294
Explain 224, 235, 285
EXPLAIN STMTCACHE ALL 74
EXPLAIN STMTCACHE STMTID 74
expression 168
extract 297–298, 306

F

FETCH 10, 164, 220, 269, 311–312
fetch 9–10, 108, 136, 164, 213
FICON 49
FILE 171, 176, 298–299
FILE subcommand 298
function 3, 6, 23–24, 71, 94, 109, 142, 156, 192, 219, 272

G

GB bar 62, 79, 90
 virtual storage 65
Generalized Trace Facility 298
getpage 130, 254
GLOBAL 64, 73, 90, 158, 176, 219, 233, 247, 249, 259,
306–307
GRECP 86, 147
GROUP 18, 36, 103, 127, 146, 164, 218, 220, 256, 281,
311
GTF 17, 19, 105, 298

H

handle 78, 154, 173, 192, 262
History 111
host variables 9
Hybrid join 177

I

I/O 4, 10, 24, 30, 55, 58, 66, 81, 84, 107, 121, 125–126,
171, 180–181, 219, 231–232, 245, 253, 258, 266, 301
IBM Tivoli OMEGAMON XE for DB2 Performance Expert
on z/OS 7, 313, 320
IDTHTOIN 155
IEAOPTxx 119
IFCID 301
IFCID 199 134
IFCID 225 8, 64, 140, 181, 187, 189
IFCID 239 265
IFCID 3 265
IFCID 306 86
IFCID 359 106, 235
IFCIDs 14, 105, 174, 192, 253, 260, 263, 274
IFCIF 369 56
IFI 85–86, 192, 219, 233, 281
IIPHONORPRIORITY 114
image copy 86, 103
IMMEDWRITE 239
IMPLICIT 73
IMS 30, 32, 39, 57, 181, 199, 202, 205–206, 218
index xxii, 12, 30, 78, 84, 94, 110–111, 126, 168, 224,
235, 239, 244, 247, 259, 285
index page

split 106, 235, 249
information 297–298
INSERT 64, 104, 107, 220, 269, 293–294, 311–312
insert 58, 85, 95, 137, 213, 235, 252–253, 259, 293
installation 12, 23, 173
INSTANCE 82
IRLM 32, 36–37, 53, 55–56, 89–90, 93, 112–113, 128,
189, 219, 233, 246–247, 250, 258, 266–267
IS 19, 72, 92, 144, 148, 157, 286
ISPF 9, 298, 304
IX 99, 127, 239, 248

J

Java 159, 165
JCC 71
JCL 298, 303, 305
JDBC 73, 113, 154, 157, 310, 312

K

KB 10, 63, 85, 126, 170, 181
KB page 150, 170
keyword 24, 37, 71

L

LENGTH 136
list 3, 8, 37, 42, 59, 104, 112, 133, 135, 167, 182, 190,
224, 229, 235, 250, 280
list prefetch 133, 136, 240
LOAD 9, 21–22, 107, 286, 305–306
load 297
LOB 90, 126, 156, 208, 219, 233, 259, 267, 284
LOB object 128, 147
LOB table 126
LOBs 126, 128
LOCK 65, 91, 219, 233, 246, 258, 266
LOCKING 96, 246, 251
Locking 90, 246–247, 299
locking 83, 89, 128, 211, 221, 244–245, 261, 298
locking counters 100
LOCKMAX 244, 305
locks 8, 32, 89, 143, 230, 243–244
LOCKSIZE
 ANY 305
LOG 21, 58, 85, 97, 121, 219, 233, 250, 258–259, 266
log record 88, 107
LOGGED 147
logging 24, 55, 83, 147, 190, 221, 239
long running batch job 207
LOOK 86
LPAR 24, 30, 32–33, 54, 100, 113–114, 140, 180–181
LPARs 30, 32, 114, 180, 228, 294
LPL 81, 86, 143, 147
LRSN 8, 84, 107

M

M 92, 144
maintenance 3, 58, 92, 136, 186–187, 309
map page 103

- mass delete 107
- materialization 168
- MAX 136, 158, 171, 173–174, 219, 233, 246
- MAX_RIDBLOCKS 292
- MAXCONQN 155
- MAXCONQW 155
- MAXDBAT 155
- maximum number 70, 79, 91, 155, 197, 244
- MAXTEMPS 176
- MAXTEMPS_RID 178
- MEMBER CLUSTER 104, 249
- MERGE 173, 220
- MIN 136, 158
- MODIFY 20, 91, 157, 262
- MONITOR 4, 37, 194
- MSTR 32, 53, 55, 112, 148, 190
- MSTR address space 33, 55, 112, 121, 260

N

- NFM 169
- node 172
- NONE 107, 128, 135, 160, 162
- NPI 134, 137
- NULL 20, 286, 294
- NUMLKTS 93
- NUMLKUS 94

O

- OA31116 141
- Object 9–10, 111
- ODBC 71, 154
- Omegamon 298
- online schema change 156
- optimization 97
- OPTIONS 308–309
- options 18, 40, 47, 58, 67, 88, 90, 144, 146, 156, 185
- OR 81, 282
- ORDER 21, 218, 239
- ORDER BY 127, 164
- OUTPUT 85, 107, 307

P

- Package 299–300
- PADDED 106
- page set 78, 86, 94, 151, 262
- page size 140, 150, 170, 239, 249, 252
- Parallel tasks 111
- parallelism 87, 105, 114, 119, 136, 191, 202, 211, 250, 252, 262, 277, 292, 294
- parameter marker 72
- PART 82, 101, 249
- PARTITION 144, 249
- partitioned table space 103, 249
- partitioning 253
- partitions 27, 78, 86, 90, 244, 247
- PCIe 24
- PCIe Activity Report 25
- PDB 182–183, 297

- PERFORMANCE 4, 218, 220, 281
- Performance xxi–xxii, 3–6, 15, 18, 23, 30, 44, 47, 156–157, 174, 182, 228, 265, 285, 287, 299
- performance 3, 13–14, 29–30, 56, 58, 62, 77, 83, 91, 110, 118, 125–126, 154, 170, 180–181, 201–202, 219, 221, 232, 235, 245–246, 262–263, 274, 277, 279, 298
- Performance Database 306–307
- Performance database 8
- performance database 6, 49, 126, 182, 187, 297, 299
- performance improvement 71, 137, 189
- performance management 6, 49, 231
- Performance Reporter 4
- performance trace 15, 17, 82, 106, 284–285, 287
- Performance Warehouse 4, 299
- PGFIX 140
- ping 59
- PLAN 15, 36, 65–66, 96, 194, 281, 286–287, 305
- plan 309
- PLAN_TABLE 293
- PM26973 292
- PM27872 23
- PM62797 56
- PM88166 78
- POOLINAC 155
- POSITION 21
- predicate 72, 94, 254
- PREFETCH 65, 173, 175, 254
- prefetch quantity 175
- prefix 13, 17
- PROCESS 4, 144
- PROFILEID 293
- protocol 301
- PT 64, 108, 188
- PTF 142

Q

- QMF 3, 11, 22, 31
- Queries 3
- QUERY 3, 81, 96, 246
- query 11, 49, 82, 95, 110, 126, 164, 202, 235, 252, 259, 262, 277, 282, 294
- query performance 9, 12
- QUERYNO 293

R

- RACF 55
- RANDOM 136, 175
- RBA 8, 58, 84
- RDS 65, 108, 171, 178, 254
- READS 86, 136, 175
- Real 59, 99, 180, 182, 235, 252
- Real storage 4, 180, 187
- real storage 107, 137, 161, 179, 228
- reason code 91, 159, 176
- REBIND PACKAGE 156
- RECORD 49, 82, 174
- RECOVER 86
- RECTRACE 82, 282
- recycle 295

- Redbooks website 319
 - Contact us xxiii
- REDUCE 307
- referential integrity 95
- REGION 37, 193–194, 306–307
- register
 - CURRENT EXPLAIN MODE 73
- remote location 301
- REORG 86, 95, 156, 235, 310
- REORG utility 106
- REPORT 4, 36, 157, 162, 218, 220
- Report 4, 24, 34, 44, 181, 190, 218, 274, 283–284
- Reporter 4, 39–40
- requirements 3–4, 8, 23, 49, 292
- resource unavailable 67
- response time 3–4, 7, 30, 66–67, 110, 118, 125, 137, 171, 203, 206, 249
- RESTART 86
- RETURN 311
- return 9, 126, 156, 246, 270
- RID 8, 14, 65, 167, 177, 292, 294, 318
- RIDs 177
- RMF 24–25, 29, 34, 115, 118, 145, 159, 181, 235, 262
- ROLLBACK 156, 219, 233
- row format 88
- row level sequential detection 59, 135, 140
- RRS 310
- RRSAF 57, 218
- RRULOCK 94
- RUNSTATS 177–178, 291

S

- same page 99, 129, 235, 254
- same way 49, 159
- SAVE 4, 298
- Save-File utility 298
- SCA 143
- scalar functions 274
- SCHEMA 286, 303–304
- schema 8, 156, 275
- segmented table space 170, 244, 262
- SEGSIZE 171, 305
- SELECT 311
- SEQ 20–21, 175, 254, 281
- Server 8, 71, 122, 159
- SET 59, 73, 82, 91, 157, 220, 293
- SET CURRENT SCHEMA 304
- side 37, 46, 56, 58, 103, 117, 187, 259, 294
- SIMULATED_CPU_COUNT 292
- SIMULATED_CPU_SPEED 292
- SKIPUNCI 94
- SMF 17, 19, 44, 46, 105, 196–197, 204, 226, 298, 306
- SNA 122, 259
- Sort 111, 167–168, 172
- sort key 173
- SORT_POOL_SIZE 292
- space map 33, 103, 253
 - page 103
- SPT01 66, 128
- SPUFI 3, 22, 73

- SQL 5, 14, 20, 31, 33, 55, 58, 61, 90, 113, 154, 170, 190, 203, 219, 235, 244, 254, 266, 279, 284, 292, 298–299, 302
- SQL procedures 8, 266
- SQL queries 9, 11
- SQL statement 5–6, 9, 67, 203, 219, 273, 284, 293
- SQL stored procedure 113
- SQLCODE 244
- SQLJ 280
- SQLSTATE 68
- SRB time 55, 112, 138–139
- SSID 306
- STAT 16, 19, 306
- statement 5, 9, 63, 105, 128, 144, 188, 203, 219, 235, 246, 251, 262, 270, 277, 293
- statement identifier 74
- static SQL 164, 190, 219
- STATISTICS 21, 147, 149, 220, 306, 310
- Statistics 66, 87, 162, 164, 220, 252, 299, 301
- statistics 8, 13–14, 53–54, 56, 64, 85, 93, 111, 125, 130, 154, 168, 187, 190, 211, 219, 224, 232, 244, 249, 262, 284, 298
- statistics trace 301, 306
- STATUS 91, 144, 160
- STEPLIB 305–306
- STMT 64
- STMTID 74
- SUBSTR 72, 127, 286, 293
- SUM 126, 311
- suspensions 89, 201, 211, 221–222, 231–232, 243, 245, 258, 266
- synchronous I/O 130, 233, 235
- SYSADM 285
- SYSIBM.DSN_PROFILE_ATTRIBUTES 294
- SYSIBM.DSN_PROFILE_TABLE 293
- SYSIBM.SYSDUMMY1 293
- SYSIBM.SYSTABLESPACE 127
- SYSIN 21, 218, 220, 286, 304–305
- SYSLGRIX 78, 259, 262
- SYSOPR 82, 194
- SYSPRINT 285, 304, 307
- SYSPRINT DD SYSOUT 305, 308
- System Management Facility 17
- System Management Facility (SMF) data 23
 - set 23
- system parameter 84, 163, 211
- System parameters 299
- System z 24, 49, 98, 109–110

T

- table space
 - buffer pool 254
 - data 20, 83, 103, 245, 284
 - data set 247
 - definition 284
 - level 103, 245
 - lock 99, 244
 - option 104
 - page 93, 245–246
 - partition 103, 247, 254

- scan 99, 112, 132, 254
- structure 247
- table space scans 224
- tables 10, 20, 66, 88, 90, 126, 156, 168, 244, 285, 293
- TCB time 55, 195
- TCP/IP 82, 122–123, 159, 163, 188, 219, 233, 259, 263–264, 267
- terminal I/O 4
- thread 298–299
- threshold 71, 80, 85, 130, 173–175, 234
- TIME 21, 55, 65, 82, 112, 144, 158, 211, 219, 233, 244, 249, 258, 262, 266
- TIMEOUT 91
- times DB2 105
- TOP 218
- TRACE 4, 14, 16, 82, 91, 211, 281, 284, 287
- trace record 13, 244
- trace types 14, 92
- traces 4, 13, 58, 134, 154, 201, 203, 224, 263, 265, 277
- transform 297
- triggers 134, 202, 238–239, 265, 279
- TS 21, 127, 171, 249
- TSO 82, 96, 218
- TYPE 19, 21, 37, 92, 126, 144, 158, 174, 194, 220, 233
- Type 2 122
- Type 4 122

U

- UDF 211, 219, 233, 266, 278
- UDFs 265, 279
- unique index 106
- universal table space 107
- UNLOAD 285
- UPDATE 58, 94, 219, 233, 258–259, 267, 311–312
- URLGWTH 97
- USAGE 144
- user-defined function 274
- USING 305
- UTILITY 57
- Utility 126, 145, 218
- UTS 170, 249

V

- VALUE 58, 92, 147, 149, 282
- VALUES 293
- VARCHAR 106, 249, 287
- VARIABLE 65, 196, 220
- variable 88, 95, 224
- VDWQT 129
- VERSION 144, 218, 220, 281
- Version 14, 67, 80
- versions 140, 181
- virtual storage
 - consumption 154
 - use 71, 192
- VPSIZE 132–133, 190, 234
- VSAM 298

W

- WebSphere 8, 32, 159, 181
- WITH 21, 71, 127, 144, 193, 246, 270
- WLM 29–30, 55, 97, 110, 112–114, 142, 154, 159, 181, 185, 218, 227–228, 266–267, 281
- WLM environment 227, 267
- work file 168
- workfile 128, 167–168
- workfile database 168
- workfile record 170
- workfiles 128, 170
- WORKLOAD 3–4, 34
- Workload Manager 30

X

- XLKUPDLT 94
- XML 24, 126, 219, 233, 259, 267

Z

- z/OS xxi, 3–4, 14, 17, 29–30, 54, 57, 71, 79, 90, 110, 126, 153, 176, 180, 221, 252, 262–263, 285, 299, 302
- z/OS enhancement 33
- zEDC 24–25
- zEnterprise Data Compression 25
- zIIP 9, 44, 53–54, 109, 140, 154, 225, 283, 292, 312
- zIIP processor 312
- zSeries 44, 47



Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS

(0.5" spine)
0.475" <-> 0.873"
250 <-> 459 pages



Redbooks®

Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS

Understand the DB2 traces

Monitor and tune the DB2 subsystem

Monitor and tune the DB2 transactions

This IBM Redbooks publication discusses in detail the facilities of DB2 for z/OS, which allow complete monitoring of a DB2 environment. It focuses on the use of the DB2 instrumentation facility component (IFC) to provide monitoring of DB2 data and events and includes suggestions for related tuning.

We discuss the collection of statistics for the verification of performance of the various components of the DB2 system and accounting for tracking the behavior of the applications.

We have intentionally omitted considerations for query optimization; they are worth a separate document.

Use this book to activate the right traces to help you monitor the performance of your DB2 system and to tune the various aspects of subsystem and application performance.

**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

SG24-8182-00

ISBN 0738439126