DB2 for z/OS
Best Practices

Locks and Latches

John Campbell
Distinguished Engineer
DB2 for z/OS Development

CAMPBELJ@uk.ibm.com

Disclaimer/Trademarks

THE INFORMATION CONTAINED IN THIS DOCUMENT HAS NOT BEEN SUBMITTED TO ANY FORMAL IBM TEST AND IS DISTRIBUTED AS IS. THE USE OF THIS INFORMATION OR THE IMPLEMENTATION OF ANY OF THESE TECHNIQUES IS A CUSTOMER RESPONSIBILITY AND DEPENDS ON THE CUSTOMER'S ABILITY TO EVALUATE AND INTEGRATE THEM INTO THE CUSTOMER'S OPERATIONAL ENVIRONMENT. WHILE IBM MAY HAVE REVIEWED EACH ITEM FOR ACCURACY IN A SPECIFIC SITUATION, THERE IS NO GUARANTEE THAT THE SAME OR SIMILAR RESULTS WILL BE OBTAINED ELSEWHERE. ANYONE ATTEMPTING TO ADAPT THESE TECHNIQUES TO THEIR OWN ENVIRONMENTS DO SO AT THEIR OWN RISK.

ANY PERFORMANCE DATA CONTAINED IN THIS DOCUMENT WERE DETERMINED IN VARIOUS CONTROLLED LABORATORY ENVIRONMENTS AND ARE FOR REFERENCE PURPOSES ONLY. CUSTOMERS SHOULD NOT ADAPT THESE PERFORMANCE NUMBERS TO THEIR OWN ENVIRONMENTS AS SYSTEM PERFORMANCE STANDARDS. THE RESULTS THAT MAY BE OBTAINED IN OTHER OPERATING ENVIRONMENTS MAY VARY SIGNIFICANTLY. USERS OF THIS DOCUMENT SHOULD VERIFY THE APPLICABLE DATA FOR THEIR SPECIFIC ENVIRONMENT.

IRLM

- In V8, 64bit IRLM is supported
  - PC=YES is forced
  - Reduced requirement for ECSA
- Make sure of high IRLM dispatching priority
  - Use WLM service class SYSSTC
- IRLM trace can add up to 25%
  - Can also increase IRLM latch contention
- MODIFY irlmproc,SET,DEADLOK= or TIMEOUT= to dynamically change deadlock and timeout frequency

	TIMEOUT	***DEADLOK***
Range allowed	1 to 3600 sec	0.1 to 5 sec
Default	60 sec	1 sec
Recommendation	30 sec	0.5 sec

Lock Avoidance

- Combination of techniques to prevent retrieval of uncommitted data
  - (1) Page latching (and page p-lock in data sharing) controlled by DB2 to ensure physical consistency of the page
  - (2) Commit log sequence number (CLSN) – at the page level
    - DB2 tracks "time" of last update to page (on page) **(A)**
    - DB2 tracks "time" of oldest uncommitted activity on every pageset/partition **(B)**
      - Non Data Sharing
        - CLSN = lowest uncommitted RBA for all active transactions for a given pageset
      - Data Sharing
        - For non-GBP-dependent page sets, each member uses a local CLSN = lowest uncommitted LRSN for all active transactions for a given pageset
        - For GBP-dependent page sets, a Global CLSN value is maintained for the entire data sharing group = lowest CLSN value across all members across all page sets (GBP-dependent or not)
    - If **(A)** < **(B)** everything on the page is guaranteed to be committed
      - Else, check Possibly UNCommitted bits (PUNC bits)

Lock Avoidance …

- Combination of techniques to prevent retrieval of uncommitted data …
  - (3) Possibly UNCommitted bits (PUNC bits) – at the row level
    - On each row, a PUNC bit is set when the row is updated
      - PUNC bits are periodically reset
        - If successful CLSN check and more than 25% of the rows have the PUNC bit ON
        - RR scanner
        - REORG TABLESPACE
    - If the PUNC bit is not ON, the row/key is guaranteed to be committed

Lock Avoidance …

- Benefits of lock avoidance
  - Increase in concurrency
  - Decrease in lock and unlock activity requests, with an associated decrease in CPU resource consumption and data sharing overhead
- Plans and packages have a better chance for lock avoidance if they are bound with ISOLATION(CS) and CURRENTDATA(NO)
- V8 improvements
  - Lock avoidance for non-cursor 'singleton' SELECT
    - In V7, ISO(CS) CD(YES) acquires S page/row lock on the qualified row
    - In V8, DB2 will no longer acquire and hold S page/row lock on the qualified row for ISO(CS) CD(YES or NO)
  - Overflow lock avoidance when the update of a variable length row in a data page results in a new row that cannot fit in that page
    - In V7, no lock avoidance on both pointer and overflow
    - In V8, lock on pointer only

Lock Avoidance …

Field Name	Description
QTXALOCK	LOCK REQUESTS
QTXAUNLK	UNLOCK REQUESTS

```
LOCKING ACTIVITY          QUANTITY  /SECOND  /THREAD  /COMMIT
------------------------  --------  -------  -------  -------
...
LOCK REQUESTS               521.0M    24.2K  3134.34  1050.75
UNLOCK REQUESTS             478.1M    22.2K  2876.06   964.16
```

Lock avoidance may not be working effectively if Unlock requests/commit is high, e.g. >5

- BIND option ISOLATION(CS) with CURRENTDATA(NO) could reduce # Lock/Unlock requests dramatically
- High Unlock requests/commit could also be possible from
  - Large number of relocated rows after update of compressed or VL row
    - Lock/Unlock of pointer record (or page)
  - Large number of pseudo-deleted entries in unique indexes
    - Lock/Unlock of data (page or row) in insert to unique index when pseudo-deleted entries exist
  - Both can be eliminated by REORG

Lock Avoidance …

- Effective lock avoidance is very important in data sharing
  - Long-running URs can reduce the effectiveness of lock avoidance by stopping the Global CLSN value from moving forward
  - Recommendation: Aggressively monitor long-running URs
    - 'First cut' ROTs:
      - URs running for a long time without committing: zparm URCHKTH<=5
        - Message DSNR035I
      - URs performing massive update activity without committing: zparm URLGWTH=10(K)
        - Message DSNJ031I
    - Need Management ownership and process for getting rogue applications fixed up so that they commit frequently based on
      - Elapsed time and/or
      - CPU time (no. of SQL update statements)

Lock Tuning

- As a general rule, start with LOCKSIZE PAGE as design default
  - If high deadlock or timeout, consider LOCKSIZE ROW
  - Not much difference between one row lock and one page lock request
  - However, the number of IRLM requests issued can be quite different
    - No difference in a random access
    - In a sequential scan,
      - No difference if 1 row per page (MAXROWS=1) or ISOLATION(UR)
      - Negligible difference if ISO(CS) with CURRENTDATA(NO)
      - Bigger difference if ISO(RR|RS), or sequential Insert, Update, Delete
      - Biggest difference if ISO(CS) with CURRENTDATA(YES) and many rows per page
  - In data sharing, additional data page P-locks are acquired when LOCKSIZE ROW is used

Lock Tuning …

- MAX PG/ROW LOCKS HELD from Accounting trace is a useful indicator of commit frequency
  - Page or row locks only
  - AVERAGE is for average of MAX, TOTAL is for max of MAX (of Accounting records)
    - So if transaction A had max. locks of 10 and transaction B had 20, then
      - AVERAGE (avg. of max.) = 15
      - TOTAL (max. of max.) = 20
  - In general, try to issue Commit to keep max. locks held below 100

LOCKING	AVERAGE	TOTAL
TIMEOUTS	0.00	1
DEADLOCKS	0.00	0
ESCAL.(SHARED)	0.00	0
ESCAL.(EXCLUS)	0.00	0
MAX PG/ROW LOCKS HELD	2.43	350
LOCK REQUEST	78.61	24637270
UNLOCK REQUEST	26.91	8433176
QUERY REQUEST	0.00	0
CHANGE REQUEST	1.72	539607
OTHER REQUEST	0.00	0
TOTAL SUSPENSIONS	0.20	63617
LOCK SUSPENSIONS	0.00	1518
IRLM LATCH SUSPENS.	0.20	62091
OTHER SUSPENS.	0.00	8

Lock Tuning …

- ZPARMS
  - RRULOCK – U-lock on SELECT FOR UPDATE for ISOLATION(RR|RS)
    - NO – Acquires S-locks on FETCH (default)
    - YES – Acquires U-locks on FETCH
      - If no update, U is demoted to S on next fetch
      - If update, U is promoted to X in COMMIT duration
  - XLKUPDLT – X-lock for searched UPDATE/DELETE
    - Take X-lock immediately on qualifying rows/pages instead of S|U-lock
    - Good if most accessed objects are changed
    - V8 PQ98172 introduces a new option: TARGET
      - X-lock only for the specific table that is targeted by UPDATE or DELETE
      - S- or U-lock when scanning for rows/pages of other tables referenced by the query
  - SKIPUNCI – Skip uncommitted inserts for ISOLATION(CS|RS)
    - New parameter in V8 and restricted to row level locking only

DDL and DML Concurrency

- DBD is locked
  - X by SQL DDL like CREATE/DROP/ALTER (TS, TBL, IX) and some utilities
    - Prevents parallel SQL DDLs against the same database
  - S by BIND, DB2 utility, dynamic SQL without caching
- How to improve DDL concurrency?
  - Reduce the number of objects within a database
    - Smaller DBDs also helps to reduce the DBD Pool Size and logging volume
      - Example: Create Index with 1000 page DBD results in 2000 pages logged
    - Group all SQL DDLs within the same database in the same commit scope
      - Only one delete/insert of DBD rather than one per SQL DDL
      - Avoid mixing DDLs and DMLs within the same COMMIT
      - Don't delay commit after DDLs

DDL and DML Concurrency …

- How to improve DDL and DML concurrency?
  - Dynamic SQL will not acquire S DBD locks if ZPARM CACHEDYN = YES
    - Unless dynamic statement caching use has been prohibited for another reason e.g. REOPT(VARS), or DDL was performed in the same unit of work
- To minimise lock contention on DB2 Catalog/Directory pages and DBD
  - Assign unique authid and a private database to each user
    - Each Catalog page contains only one value for dbid or authid
  - Be careful about user query on Catalog tables
    - e.g. SQL LOCK TABLE or ISOLATION(RR)

IRLM Latch Contention

Field Name	*Description*
QTXASLAT	SUSPENSIONS (IRLM LATCH)
QTXALOCK	LOCK REQUESTS
QTXAUNLK	UNLOCK REQUESTS
QTXAQRY	QUERY REQUESTS
QTXACHG	CHANGE REQUESTS

```
LOCKING ACTIVITY            QUANTITY  /SECOND  /THREAD  /COMMIT
-------------------------   --------  -------  -------  -------
SUSPENSIONS (ALL)            1498.6K    69.57     9.02     3.02
SUSPENSIONS (LOCK ONLY)     33120.00     1.54     0.20     0.07
SUSPENSIONS (IRLM LATCH)     1405.3K    65.24     8.45     2.83
SUSPENSIONS (OTHER)         60185.00     2.79     0.36     0.12
...
LOCK REQUESTS                521.0M    24.2K  3134.34  1050.75
UNLOCK REQUESTS              478.1M    22.2K  2876.06   964.16
QUERY REQUESTS             50741.00     2.36     0.31     0.10
CHANGE REQUESTS              4488.4K   208.38    27.00     9.05
OTHER REQUESTS                  0.00     0.00     0.00     0.00
```

IRLM latch contention should be less than 1-5% of Total IRLM Requests

IRLM Latch Contention = SUSPENSIONS (IRLM LATCH) (A)
Total IRLM Requests = LOCK + UNLOCK + QUERY + CHANGE REQUESTS (B)
IRLM Latch Contention Rate (%) = (A)/(B)*100

- High number of IRLM latch suspensions could be caused by
  - IRLM Trace on
  - Low IRLM dispatching priority
  - Frequent IRLM Query requests (e.g. DISPLAY DATABASE LOCKS, or MODIFY irlmproc, STATUS)
  - Very low deadlock detection cycle time and very high locking rates

Data Sharing Lock Tuning

Field Name	*Description*
QTGSLSLM	SYNCH.XES - LOCK REQUESTS
QTGSCSLM	SYNCH.XES - CHANGE REQUESTS
QTGSUSLM	SYNCH.XES - UNLOCK REQUESTS
QTGSIGLO	SUSPENDS - IRLM GLOBAL CONT
QTGSSGLO	SUSPENDS - XES GLOBAL CONT
QTGSFLMG	SUSPENDS - FALSE CONTENTION

DATA SHARING LOCKING	QUANTITY	/SECOND	/THREAD	/COMMIT
GLOBAL CONTENTION RATE (%)	0.01			
...				
SYNCH.XES - LOCK REQUESTS	227.5M	10.6K	1368.75	458.86
SYNCH.XES - CHANGE REQUESTS	1340.7K	62.24	8.07	2.70
SYNCH.XES - UNLOCK REQUESTS	225.8M	10.5K	1358.14	455.30
ASYNCH.XES - RESOURCES	1315.00	0.06	0.01	0.00
...				
SUSPENDS - IRLM GLOBAL CONT	34192.00	1.59	0.21	0.07
SUSPENDS - XES GLOBAL CONT.	6076.00	0.28	0.04	0.01
SUSPENDS - FALSE CONTENTION	21575.00	1.00	0.13	0.04

Global Contention should be less than 3-5% of XES IRLM Requests

Global Contention = SUSPENDS – IRLM + XES + FALSE (A)
XES IRLM Requests = (SYNCH. XES – LOCK + CHANGE + UNLOCK) + (SUSPENDS – IRLM + XES + FALSE) (B)
Global Contention Rate (%) = (A)/(B)*100

False Contention should be less than 1-3% of XES IRLM Requests

False Contention = SUSPENDS – FALSE (C)
XES IRLM Requests = (SYNCH. XES – LOCK + CHANGE + UNLOCK) + (SUSPENDS – IRLM + XES + FALSE) (B)
False Contention Rate (%) = (C)/(B)*100

Data Sharing Lock Tuning …

- Notes
  - IRLM Cont. = IRLM resource contention
  - XES Cont. = XES-level resource cont. as XES only understands S|X mode
    - e.g. member 1 asking for IX and member 2 for IS
    - Big relief with Locking Protocol 2 when enabled after entry to V8 (NFM) but increased overhead for pageset/partitions opened for RO on certain members, e.g.
      - -START DATABASE(....) SPACENAM(...) ACCESS(RO)
      - SQL LOCK TABLE statement
      - LOCKSIZE TABLESPACE or LOCKSIZE TABLE for segmented tablespace
      - Table scan with Repeatable Read isolation level
      - Lock escalation occurs
  - False Cont. = false contention on lock table hash anchor point
    - Could be minimised by increasing the number of Lock entries in the CF Lock Table
    - Note: the counter is maintained on a per-LPAR basis
      - Will over-report false contentions in cases where multiple members from the same data sharing group run on the same z/OS image

Data Sharing Lock Tuning …

Field Name	*Description*
QTGSPPPE	PSET/PART P-LOCK NEGOTIATION
QTGSPGPE	PAGE P-LOCK NEGOTIATION
QTGSOTPE	OTHER P-LOCK NEGOTIATION

DATA SHARING LOCKING	QUANTITY	/SECOND	/THREAD	/COMMIT
...				
SYNCH.XES - LOCK REQUESTS	227.5M	10.6K	1368.75	458.86
SYNCH.XES - CHANGE REQUESTS	1340.7K	62.24	8.07	2.70
SYNCH.XES - UNLOCK REQUESTS	225.8M	10.5K	1358.14	455.30
ASYNCH.XES - RESOURCES	1315.00	0.06	0.01	0.00
...				
PSET/PART P-LCK NEGOTIATION	18037.00	0.84	0.11	0.04
PAGE P-LOCK NEGOTIATION	2863.00	0.13	0.02	0.01
OTHER P-LOCK NEGOTIATION	9067.00	0.42	0.05	0.02
P-LOCK CHANGE DURING NEG.	15991.00	0.74	0.10	0.03

- P-lock contention and negotiation can cause IRLM latch contention, page latch contention, asynchronous GBP write, active log write, GBP read

P-lock Negotiation should be less than 3-5% of XES IRLM requests

P-lock Negotiation = P-LOCK NEGOTIATION – PSET/PART + PAGE + OTHER (A)
XES IRLM Requests = (SYNCH. XES – LOCK + CHANGE + UNLOCK) + (SUSPENDS – IRLM + XES + FALSE) (B)
P-lock Negotiation Rate (%) = (A)/(B)*100

Data Sharing Lock Tuning …

- Notes
  - Other P-lock negotiation for
    - Index tree P-lock (See LC06 contention in Latch section)
    - Castout P-lock
    - SKCT/SKPT P-lock
  - Breakdown by page P-lock type in GBP statistics

GROUP TOTAL CONTINUED	QUANTITY	/SECOND	/THREAD	/COMMIT
...				
PAGE P-LOCK LOCK REQ	877.4K	122.88	14.91	3.64
SPACE MAP PAGES	83552.00	11.70	1.42	0.35
DATA PAGES	10663.00	1.49	0.18	0.04
INDEX LEAF PAGES	783.2K	109.69	13.31	3.25
PAGE P-LOCK UNLOCK REQ	926.8K	129.80	15.75	3.84
PAGE P-LOCK LOCK SUSP	8967.00	1.26	0.15	0.04
SPACE MAP PAGES	593.00	0.08	0.01	0.00
DATA PAGES	143.00	0.02	0.00	0.00
INDEX LEAF PAGES	8231.00	1.15	0.14	0.03
PAGE P-LOCK LOCK NEG	10285.00	1.44	0.17	0.04
SPACE MAP PAGES	8.00	0.00	0.00	0.00
DATA PAGES	10.00	0.00	0.00	0.00
INDEX LEAF PAGES	10267.00	1.44	0.17	0.04

Data Sharing Lock Tuning …

- To reduce page p-lock and page latch contention on space map pages during heavy inserts into GBP-dependent tablespace
  - TRACKMOD NO
  - MEMBER CLUSTER
    - Option only available for partitioned tablespace
      - Switching to partitioned will likely result in extra getpages for true varying length rows and fixed length compressed
    - Increases the number of space map pages
      - 199 data pages per space map instead of 10K per space map
  - When MEMBER CLUSTER is used, LOCKSIZE ROW has the potential to provide additional relief for insert intensive workloads
    - Reduced page P-lock and page latch contention on data pages
    - Better space use
    - Reduced working set of pages in buffer pool

Data Sharing Lock Tuning …

- BUT…
  - Do not use LOCKSIZE ROW without MEMBER CLUSTER for an INSERT-at-the-end intensive workload
    - May result in excessive page p-lock contention on data pages and space map pages, in addition to the extra locking protocol that comes with taking page p-lock
  - Do not use LOCKSIZE ROW with or without MEMBER CLUSTER for a mixed INSERT/UPDATE/DELETE workload
    - Consider using LOCKSIZE PAGE MAXROWS 1 to reduce page P-lock and page latch contention on data pages

Data Sharing Lock Tuning …

- 50% of CLAIM_REQ can be used as a very rough estimate of # tablespace/partition locks acquired when effective thread reuse with RELEASE(COMMIT), or no thread reuse
  - CLAIM_REQ assumed once for index and once for tablespace/ partition for a very rough estimation
  - These tablespace/partition locks can be eliminated with effective thread reuse with use of RELEASE(DEALLOCATE)

Internal DB2 Latch Contention

Field Name	*Description*
QVLSLC01 to QVLSLC32	INTERNAL LATCH CONTENTION BY CLASS 1-32

LATCH CNT	/SECOND	/SECOND	/SECOND	/SECOND
LC01-LC04	0.00	0.00	0.00	0.00
LC05-LC08	0.00	75.62	0.00	0.01
LC09-LC12	0.00	0.79	0.00	1.25
LC13-LC16	0.01	676.17	0.00	0.00
LC17-LC20	0.00	0.00	105.58	0.00
LC21-LC24	0.08	0.00	6.01	4327.87
LC25-LC28	4.18	0.00	0.02	0.00
LC29-LC32	0.00	0.20	0.57	25.46

Try to keep latch contention rate < 1K-10K per second

- Disabling Acct Class 3 trace can help reduce CPU time when high latch contention
- Typical high latch contention classes highlighted
  - LC06 = Index split latch
  - LC14 = Buffer pool LRU and hash chain latch
  - LC19 = Log latch
  - LC24 = Prefetch latch or EDM LRU chain latch

Internal DB2 Latch Contention …

- LC06 for index tree P-lock by index split
  - Index split is particularly painful in data sharing
    - Results in two forced physical log writes
      - Index split time can be significantly reduced by using faster active log device
  - Options to reduce index split
    - Distributed freespace tuning
    - Minimum index key size especially if unique index
    - NOT PADDED index for large varchar columns (V8)
    - Large index page size (V9)
    - Asymmetric leaf-page split (V9)
  - A number of index splits in LEAFNEAR/FAR in SYSINDEXPART and RTS REORGLEAFNEAR/FAR
  - X'46' in IFCID 57 performance trace
  - X'FE' index tree latch in non data sharing not in Class 6

Internal DB2 Latch Contention …

- LC14 Buffer Pool latch
  - If many tablespaces and indexes, assign to separate buffer pools with an even Getpage frequency
  - If objects bigger than buffer pool, try enlarging buffer pool if possible
  - If high LC14 contention, use buffer pool with at least 3000 buffers
  - Use FIFO rather than LRU buffer steal algorithm if there is no read I/O, i.e. object(s) entirely in buffer pool
    - LRU = Least Recently Used buffer steal algorithm (default)
    - FIFO = First In First Out buffer steal algorithm
      - Eliminates a need to maintain LRU chain which in turn
        - Reduces CPU time for LRU chain maintenance
        - Reduces CPU time for LC14 contention processing

Internal DB2 Latch Contention …

- LC19 Log latch
  - Minimise #log records created via
    - LOAD RESUME/REPLACE with LOG NO instead of massive INSERT/UPDATE/DELETE
    - Segmented or UTS tablespace if mass delete occurs
  - Increase size of output log buffer if non-zero unavailable count
    - When unavailable, first agent waits for log write
    - All subsequent agents wait for LC19
  - Reduce size of output log buffer if non-zero output log buffer paging
    - See Log Statistics section
  - Potential for reduced LC19 Log latch contention in DB2 9 (CM, NFM)

Internal DB2 Latch Contention …

- LC24 latch
  - Sum of two types of latch contention
  - (1) EDM LRU latch – X'18' in IFCID 57 performance trace
    - Use EDMBFIT zparm of NO
    - Thread reuse with RELEASE DEALLOCATE instead of RELEASE COMMIT for frequently executed packages
  - (2) Prefetch scheduling – X'38' in IFCID 57 performance trace
    - Higher contention possible with many concurrent prefetches
    - Disable dynamic prefetch if unnecessary by setting VPSEQT=0
    - Use more partitions (one x'38' latch per data set)