DB2 for z/OS Version 8 Performance

Akira Shibamiya presented by Roger Miller

Tuesday March 7, 2006

Session 1328

- **Abstract:** **DB2 UDB for z/OS V8 Application and System Performance:**

- **Abstract: This session covers the application /system performance topics for DB2 UDB for z/OS V8 including:**
  - **Query performance enhancement such as materialized query table and non-index column distribution statistics**
  - **SQL performance enhancement such as more indexable predicates and multi-row Fetch, Update, Delete, Insert**
  - **Index enhancement such as variable length index keys**
  - **Other application performance enhancement such as trigger and lock avoidance**
- **Speaker: Akira Shibamiya is the primary source, presented by Roger Miller, IBM Silicon Valley Lab**
- **This presentation provides information on DB2 UDB for z/OS V8 performance. Please note that some product changes may result in changes.**

Acknowledgment and Disclaimer

- Measurement data included in this presentation are obtained by the members of the DB2 performance department at the IBM Silicon Valley Laboratory.
- The materials in this presentation are subject to
  - enhancements at some future date,
  - a new release of DB2, or
  - a Programming Temporary Fix
- The information contained in this presentation has not been submitted to any formal IBM review and is distributed on an "As Is" basis without any warranty either expressed or implied. The use of this information is a customer responsibility.

3

Performance Highlight

- 10 to 1000 times improvement possible from
  - Materialized Query Table
  - Stage 1 and indexable predicate for unlike data types
  - Distribution statistics on non-indexed columns
  - Other access path selection enhancements

Underlined features require rebind

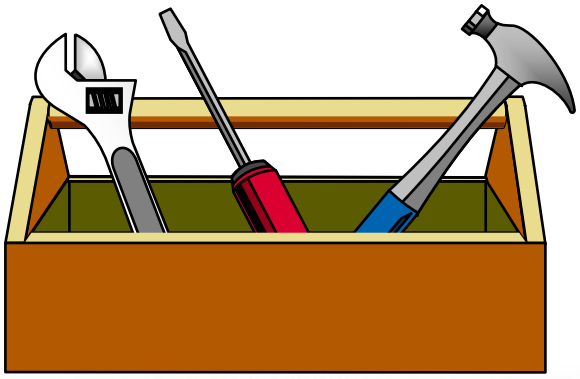
4

Performance Highlight - continued

- 2 to 5 times improvement possible from
  - Star Join with work file index and in-memory work file
  - Partition Load/Reorg/Rebuild with DPSI
  - DBM1 virtual storage constraint relief
- Up to 2 times (more in distributed environment) improvement possible from
  - Multi-row Fetch, cursor Update, cursor Delete, Insert

Underlined features require rebind

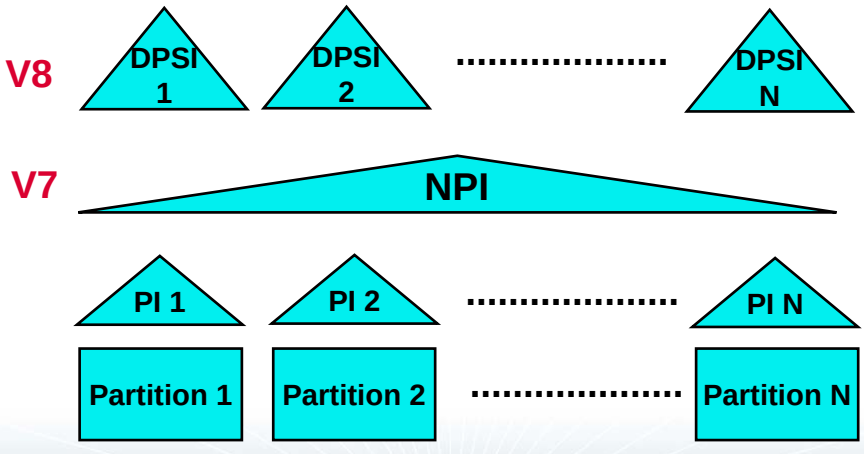
Utility Performance



Utility Performance Outline

- Parallel Partition Load/Reorg/Rebuild with Data Partitioned Secondary Indexes
- Others
  - SORTDATA/SORTKEYS default for Load, Reorg, Rebuild
  - V7 PQ56293 4/2002 Parallel Copy and Restore of objects on tape
  - V6/V7 PQ73605 8/2003 Long Load Replace with unique index when duplicate keys exist
  - V7 PQ74111 8/2003 Copy share change OPTIONS EVENT to avoid all objects being unavailable for the duration of the Copy job

Parallel Partition Load / Reorg / Rebuild with DPSI



DPSI ...

- PI = Partitioning Index
  - One data set per partition
  - Example: unique ACCOUNT#
- DPSI = Data Partitioned Secondary Index
  - One data set per partition
  - Example: non unique CUSTNAME
- NPI = Non Partitioning Index
  - One, or multiple if PIECESIZE, data set(s) per tablespace
  - Possible contentions by concurrent partition utilities
- A single table may have a mix of NPI and DPSI

Partition Load/Reorg/Rebuild - continued

- Much faster partition-level operation when multiple indexes present
  - Up to N times faster where N is the number of partitions
    - Avoids accessing entire index in single partition utility
    - Avoids contention and insert mode processing in parallel partition utilities such as Load

Partition Load/Reorg/Rebuild - continued

- Much faster partition-level operation when multiple indexes present - continued
  - Reduces data sharing overhead if partition affinity by member
  - Avoids Online Reorg Build2 phase (invoked when partition utility with NPI present)
    - Build2 typically results in the longest period of unavailability for selected partitions and logical partitions of NPIs during Online Reorg

DPSI Usage Considerations

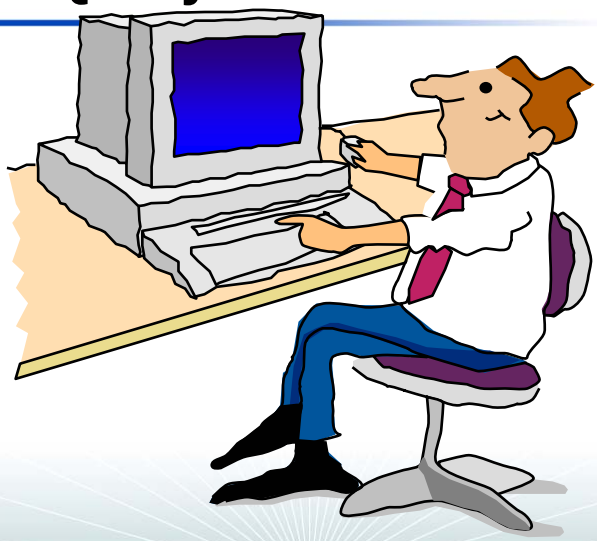
- Not for unique index
- Query performance impact

1. Depending on PI predicates available, some or all DPSI partitions may have to be scanned because the same DPSI key values may be in multiple partitions
   - More % overhead with fewer rows scanned and/or more partitions scanned
2. ORDER BY DPSI column may require extra processing
3. Trade-off between partition tablespace utility and some query performance

DPSI ...

- Insert of each row results in all DPSI partitions to be checked to make sure it is unique, if unique index is to be supported.
- Example for (1)
  - SELECT FROM table WHERE CUSTNAME=x, with
    - unique ACCOUNT# as PI key
    - non-unique CUSTNAME as DPSI key
- Example for (2)
  - SELECT FROM table WHERE CUSTNAME BETWEEN x AND y ORDER BY CUSTNAME

Query Performance



Query outline

- Materialized query table
- Distribution statistics on non-indexed columns
- Star join
- Others

Materialized Query Table

- Pre-selected and/or pre-computed results from large table(s) saved in much smaller MQT for fast subsequent access
  - Example: Avg Income, Height, NetAssetValue, ... of 300 million US residents grouped by 50 states
  - 10 to 1000 times faster possible for some queries
- Automatic query rewrite for dynamic SQL to take advantage of relevant MQT
  - Summary table can be used directly by both static and dynamic SQL

MQT ...

- MQT = Materialized Query Table, sometimes called Automatic Summary Table
- Existing tables can be registered as MQT via ALTER TABLE
- If MQT is used, Plan Table TABLE_TYPE='M'

Materialized Query Table - continued

- Performance considerations for maximum use
  - For large MQT,
    - Use segmented tablespace because of almost instantaneous mass delete in REFRESH TABLE
    - Runstats after REFRESH for good access path selection
      - ☞ especially useful in join involving MQT
  - Zparm SPRMMQT for threshold to prevent unnecessary additional bind overhead for short-running SQL

MQT ...

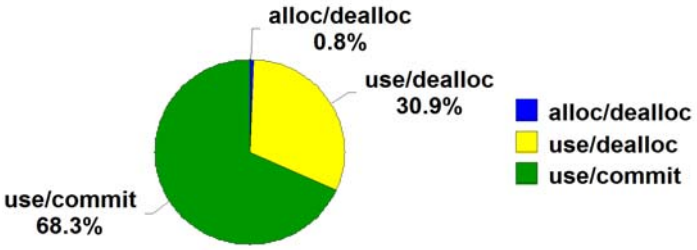
- REFRESH TABLE deletes current data and then inserts new data. MQT is locked out during this process.
- System-maintained MQT can be used just like any other table except for some restrictions
  - No Insert, Update, or Delete allowed
- User-maintained MQT supports both REFRESH TABLE and Insert, Update, Delete

Distribution stats on single and multiple columns

- Top N highest, and/or lowest, frequency of values and cardinality

Bind option

Acquire / release example



SELECT FROM A, SYSIBM.SYSPLAN B WHERE B.ACQUIRE='A' AND B.RELEASE='D' ...

Better join sequence from more precise filter factor estimation of combined predicates

Distribution statistics ...

- DSTATS (Distribution stats for DB2 for z/OS)
  - A down-loadable tool available prior to V8
  - http://www.ibm.com/support/docview.wss?uid=swg24001598
- Fixes the most typical access path selection problems encountered today
  - Optimizer unable to come up with the best access path because of a lack of distribution stats on non-indexed columns which are referenced in predicates
    - ➔Can cause performance degradation due to access path change in a new release or after access-path-related maintenance

Star Join Performance

- Use of sparse index on work file to reduce workfile scan
  - Sparse index up to 240 KB in memory
  - Binary search of index followed by sequential scan of workfile subset with nest loop rather than merge join
  - 2 to 5 times faster for some star join queries
  - Also in V7 PQ61458 6/2002
- Use of memory above 2GB rather than workfile when available

Notes

- Normal index: 1 index entry for each row
- Sparse index: 1 index entry for every N rows

- Sparse index first used in DB2 V4
  - 16 KB sparse index in memory for non-correlated IN subquery for up to 100 times performance improvement
- Accesstype='T' in Plan Table for sparse index access for workfile
- Star join in-memory work file can also prevent performance disruption on other threads using work files, such as sort, merge join, trigger, created temp table, non-correlated subquery, table UDF, outer join, materialization of nested table expression and/or view, ...

Others

- More parallel sort enablement
- Cost-based parallel sort
- Multi-column merge join parallelism
- Visual Explain rewrite
- Faster bind for many table query
- Numerous access path selection enhancements

Other performance ...

- Examples of IN-list performance enhancement
  - Dynamic instead of sequential prefetch in IN-list index access to data if not contiguous (V6 PQ71925 5/2003)
  - IN-list predicate pushdown into materialized view or table expression
  - Cross query block transitive closure for IN-list
  - Correlated subquery transformation with IN-list

SQL Performance

SQL Performance Outline

- More indexable predicates
- Multi-row Fetch
- Multi-row Insert, cursor Update, cursor Delete
- Multi-row Fetch, Insert, Update, Delete in distributed environment
- Automatic use of multi-row Fetch

More Indexable Predicates

- For column comp-op value with unlike type or length
  - 4 byte char column = 8 byte host variable
  - Integer column = decimal host variable
  - Stage 2 and non indexable in V7
  - Stage 1 and indexable in V8
    - So index on char or integer column here can be used in V8 but not in V7
  - Also useful where a programming language does not support SQL data types. For example,
    - No decimal type by C/C++, no fixed-length char by Java

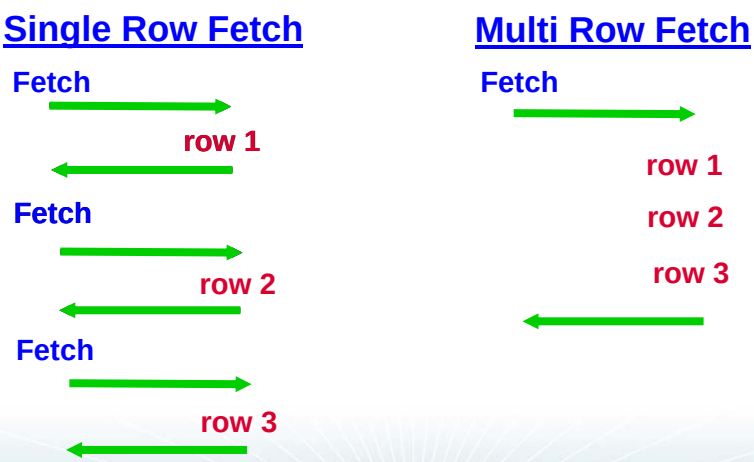
NOTES

- Stage 1 and indexable predicate in
  - V6: Column comp-op non column expression such as SELECT FROM A WHERE a1=x+y
    - also char/varchar of different size in equi-join such as SELECT FROM A,B WHERE 10byte char a1=20byte varchar b1
  - V7: Column comp-op column expression in join such as SELECT FROM A,B WHERE a1=b1+x, if table B joined to A
- But generally only if left side column has equal or bigger size and precision
- V8 removed this restriction for both local and join predicates

Indexable Predicates- continued

- SELECT FROM Unicode table U, EBCDIC table E WHERE u1=e1 ...
  - In V7, join of U and E tables not allowed.
  - In V8, multiple CCSID sets per SQL statement supported
    - Useful in joining with catalog table
  - If join of E to U, stage 1 and indexable. An index on u1 can be used.
  - If join of U to E (E is inner table), stage 1 but not indexable

Multi-row Fetch



Multi-row Fetch - continued

- FETCH NEXT ROWSET FROM cursor FOR N ROWS INTO hva1, hva2, hva3
- Up to 50% CPU time reduction by avoiding API (Application Programming Interface) overhead for each row fetch
  - % improvement lower if more columns and/or fewer rows fetched per call
    - Higher improvement if accounting class 2 on, CICS without OTE, many rows, few columns
  - See later foils for distributed

Multi-row Insert

- INSERT INTO TABLE FOR N ROWS VALUES(:hva1,:hva2,...)
- Up to 40% CPU time reduction by avoiding API overhead for each row insert
  - % improvement lower if more indexes, more columns, and/or fewer rows inserted per call
- Similar improvement for multi-row cursor Update and Delete

Multi-row operations

- hva = host variable array
- API = Application Program Interface overhead for each SQL call

- Atomic (default) specifies that if insert of any row fails, then all changes made are undone.
  - Atomic requires SAVEPOINT which takes about 15us on z900 typically, contributing less than 5% overhead with 2 row Insert and completely negligible for many row Insert.

- Typical best use is 100 to 1000 rows
- Up to 32767 rows can be processed in one call
- Support for C, C++, Cobol, PL/I, Assembler

Multi-row in distributed environment

- Fetch, insert, update & delete
- Dramatic reduction in network traffic and response time possible
  - by avoiding message send/receive for each row in
    - Fetch when not [read-only or (CURRENTDATA NO and ambiguous cursor)]
    - Update and/or Delete with cursor
    - Insert
  - Up to 8 times elapsed time reduction observed (up to 4 times CPU time reduction)

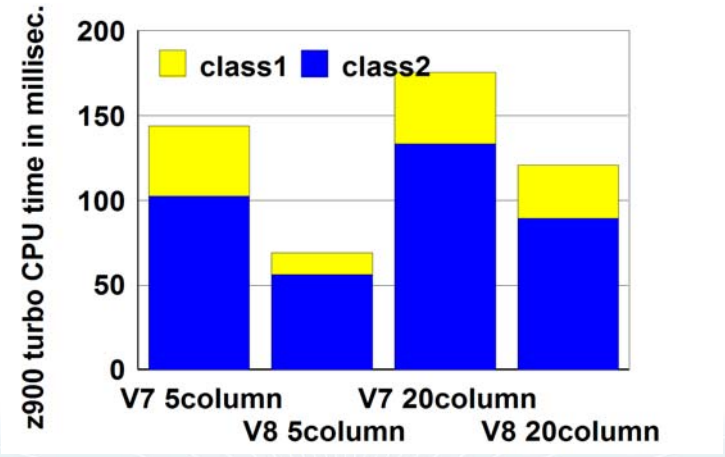
Distributed multi-row ...

- If Fetch with read-only or [CURRENTDATA NO and ambiguous cursor], multi-row Fetch is automatically enabled, resulting in
  - CPU time saving of up to 50%
  - But no significant difference in message traffic compared to V7 with Block Fetch
    - Note that multi-row Fetch is unblocked; i.e. if 10 Fetch calls are issued for 10 rows each, 10 blocks are sent, compared to 1 block if multi-row Fetch is not explicitly used.
    - V7 PQ49458 8/2003
      - OPTIMIZE FOR for access path and network blocking
      - FETCH FIRST for access path but not network blocking when no OPTIMIZE FOR clause

Automatic use of multi-row Fetch

- DRDA as discussed previously
- DSNTEP4 = DSNTEP2 with automatic multi-row fetch
  - Up to 35% CPU reduction in fetching 10000 rows with 5 and 20 columns
- DSNTIAUL (sample Unload utility)
  - Up to 50% CPU reduction in fetching 10000 rows with 5 and 20 columns

DSNTIAUL fetching 10000 rows with 5 and 20 columns



Long-term page fix option by buffer pool

- Important for minimizing CPU time increase in V8
- LRU (Least Recently Used) buffer steal algorithm guarantees paging if insufficient real storage to back up the buffer pool in entirety

Therefore, it is always strongly recommended that there is sufficient real storage to back up the buffer pool 100%. 99.99% is not good enough.

- Given 100% real storage, might as well page fix all buffers just once to avoid the cost of page fix and free for each and every I/O

Long-term page fix - continued

- Via new option
  - ALTER BPOOL(name) PGFIX(YES)
- Page fix for each buffer in buffer pool once and keep it fixed
- 7% overall CPU time reduction for IRWW transaction observed
  - 0 to 8% saving typically expected
- Especially beneficial for I/O intensive application
  - Recommended for BPs with low hit ratio, i.e. lots of I/O's

Notes

- IRWW = IBM Relational Warehouse Workload
- ALTER effective at next BP allocation
  - For user data,
    - ALTER BPOOL(name) VPSIZE(0)
    - ALTER BPOOL(name) VPSIZE(...) PGFIX(YES)
  - For catalog/directory,
    - ALTER BPOOL(name) PGFIX(YES)
    - STOP DATABASE or DB2
    - START DATABASE or DB2

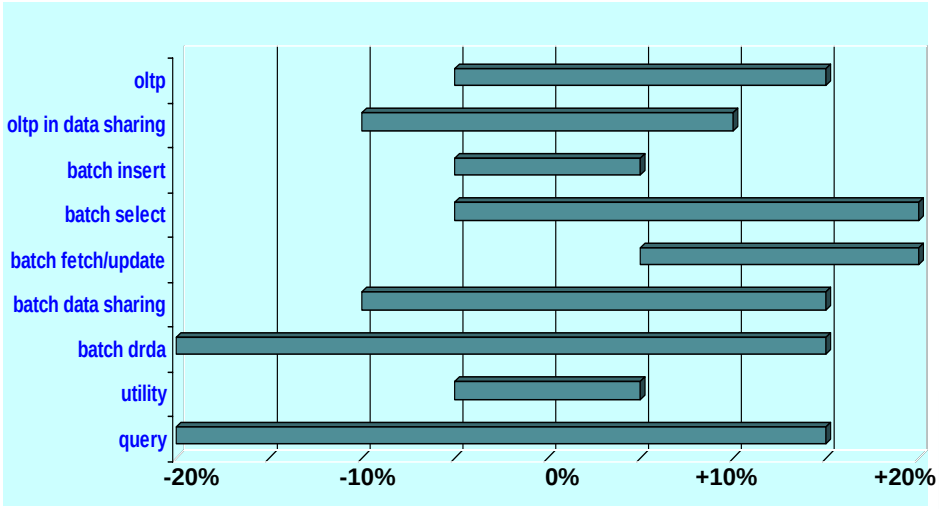
Data sharing performance enhancement

- Reduced global and false contention for pageset/partition locks
  - -6% overall CPU time for IRWW 2 way data sharing
  - Less need for Release Deallocate bind option
- Batching of multiple coupling facility (CF) write and castout read requests into one CF access with z/OS 1.4 and CF level 12
  - Bigger benefit for Insert/Update/Delete-intensive application
- CF level 13 useful for DB2 V7 (PQ86049) and V8

For applications not taking advantage of V8 performance features

- **Some CPU time increase is expected in order to support a dramatic improvement in user productivity, availability, scalability, portability, family consistency,..**
  - **DBM1 virtual storage constraint relief with 64bit instructions**
  - **Long names, long index keys**
  - **Longer and more complex SQL statements**
- **I/O time**
  - **No change for 4K page I/O**
  - **Significant sequential I/O time improvement possible for 8K, 16K, or 32K page because of bigger Vsam Control Interval size**
    - **Up to 70% I/O data rate (MB/sec) improvement**
    - **Also VSAM I/O striping now supported**
  - **V8 PQ99608 2/05 to fix excessive log write i/o's**

CPU change - continued



CPU change based on laboratory measurements with no application nor aggressive configuration/environment change

'+' means cpu increase, '-' means reduction, compared to V7 Minus is good

- **-5 to +15% online transaction**
- **-10 to +10% online transaction in data sharing (NFM)**
- **-5 to +20% batch**
  - **-5 to +5% insert**
  - **-5 to +20% select**
  - **+5 to 20% fetch, update**
- **-10 to +15% batch data sharing (NFM)**
- **-20 to +15% batch DRDA**
- **-5 to +5% utility**
- **-20 to +15% query**

➲ **Numbers subject to change as more data become available**

CPU change - continued

➲ **Typically, no difference between CM and NFM except in data sharing for workloads with**

- **No application change**
- **No aggressive configuration/environment change**

- **Examples of what CM supports**
  - **Most access path selection enhancements**
  - **DBM1 virtual storage constraint relief**
  - **Lock avoidance in Select Into with CurrentData Yes, overflow rows**
  - **180 CI limit removal in list prefetch and castout I/O**
  - **Long-term page fix option by buffer pool**
  - **SMF 89 performance enhancement (usage pricing)**
  - **Data sharing immediate write default change**
  - **Implicit multi-row operation in DRDA**

V8 vs V7 OLTP measurements

- **ITR=Internal Throughput Ratio, inversely proportional to CPU, plus is good**
- **CPU minus is good**
- **One ERP vendor transaction -8% CPU 6/04**
- **IRWW data sharing +7% ITR 11/04**
- **IRWW non data sharing 0% ITR 11/04**
- **5 SVL client/server transaction workloads 0 to -12% ITR 9/04**

➲ **-5 to +15%, or -10 to +10% data sharing, typical CPU observed**

V8 vs V7 Query measurements

- **Over 200 TPCD and other queries: 0% avg cpu, -20% total cpu 8/04**
  - **Some queries benefit from improved access path selection**
- **160 BW queries: -30 to -40% cpu and elapsed time because of star join enhancement 6/04 (NFM)**

➲ **-20 to +15% typical CPU observed**

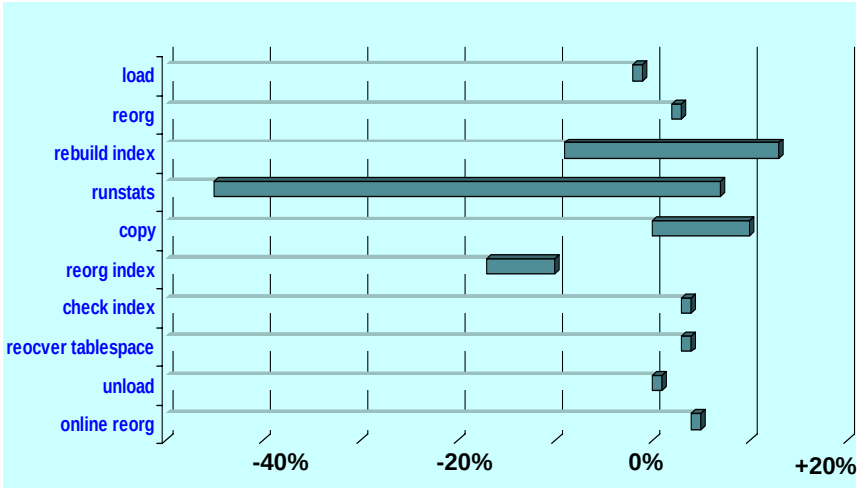
V8 vs V7 Batch CPU measurements

- **SVL batch 20 column rows 7/04**
  - **+12% Fetch/Update**
  - **+5% Fetch/Delete**
  - **-6% Insert**
- **IBM Japan batch at SVL 11/04, both CD YES and NO**
  - **+10 to 15% Fetch loop**
  - **-4% (if CD YES) to +15% (if CD NO) Select loop**
    - **V8 PQ89070 8/04 for lock avoidance in Select Into with Cursor Stability and Currentdata Yes**
  - **+5% Select Max**
  - **+20% Fetch/Update or Delete loop**
  - **+2% Insert**

➲ **-5 to +20% typical CPU observed**

- **Could be significantly better if data sharing or DRDA**
- **V8 multi-row can have a significant improvement here**

Utility CPU - continued



V8 vs V7 Utility CPU measurements

- **SVL Utility 9/04**
  - **Load -2%**
  - **Reorg +2%**
  - **Rebuild Index -9 to +13%**
  - **Runstats -45% to +7%**
  - **Copy 0 to +10%**
  - **Reorg Index -10 to -17%**
  - **Check Index +3%**
  - **Recover Tablespace +3%**
  - **Unload 0%**
  - **Online Reorg +4%**

➲ **-5 to +5% typical CPU observed**

Tuning for CPU usage in V8

- **If DBM1 virtual storage constrained in V7, recheck various actions taken to reduce virtual storage usage at the cost of additional CPU time, such as**
  - **Reduced size of buffer pools and other pools**
  - **Bind option release commit and/or no thread reuse to reduce thread and EDM storage size**
  - **EDM best fit to reduce EDM pool size**
  - **MINSTOR and CONTSTOR to reduce thread storage size**
  - **Lower DSMAX to reduce storage for data set control blocks and compression dictionary**

Notes

- **Performance enhancements introduced to compensate for increased CPU time to support new V8 functions are described next.**

Long-term page fix option for buffer pool (BP) with frequent I/O's

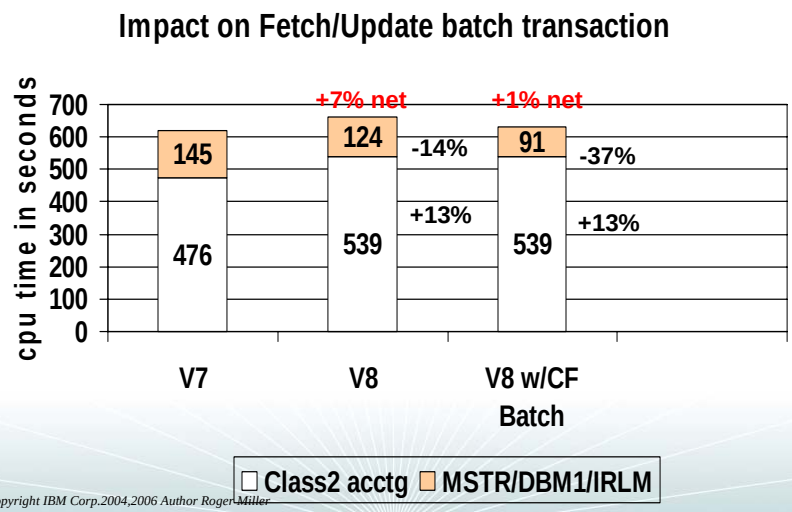
- **DB2 BPs have always been strongly recommended to be backed up 100% by real storage**
  - **To avoid paging which occurs even if only one buffer short of real storage because of Least-Recently-Used buffer steal algorithm**
  - **Given 100% real storage, might as well page fix all buffers just once to avoid the cost of page fix and free for each and every I/O**
- **Up to 8% reduction overall IRWW transaction cpu time**
- **New option: ALTER BPOOL(name) PGFIX(YES/NO)**
- **Recommended for BPs with high buffer I/O intensity = [pages read or written]/[number of buffers]**

One example with 100,000 buffers total (400MB)

	#buffers	Pages read or written	Buffer I/O intensity
BP0 catalog/directory	2000	10000	5 #2
BP1 workfile	10000	20000	2 #3
BP2 in-memory index or data	30000	100	0.003
BP3 other index	53000	100000	1.9 #4
BP4 other data	5000	100000	20 #1

BP4, 0, 1, 3 in that order. Don't bother with BP2.

Batch multiple CF write & castout read requests into one CF access: NFM & z/OS1.4 & CF level 12



More Tuning for CPU usage in V8

- Rebind plans/packages
  - **Better access path selection, especially beneficial for complex query**
  - **Enable SPROC (fast column processing) for 64 bit**
  - **Take advantage of some ALTERed objects; for example**
    - **Matching index access or index-only after Alter Index Add Column**
    - **Index-only after Alter padded to non padded index**

More CPU Tuning - continued

- **PQ86071/89919 6/04 to remove 180 VSAM Control Interval limit in list prefetch and castout I/O**
  - **2 to 3% overall CPU time reduction for IRWW**
- **Group-wide shutdown and restart to reduce global and false contentions for pageset/partition locks when release commit in data sharing (NFM)**
  - **-6% overall cpu for 2 way data sharing IRWW**

Notes

- **Less regression possible if**
  - **IRLM PC YES and LOCKPART YES in V7**
  - **Hiperpool/Dataspace buffer pool used in V7**
  - **SMF 89 (usage pricing) active**
    - **CPU captured in class 1, not class 2 accounting**
  - **DRDA, DSNTEP4, DSNTIAUL which can automatically exploit multi-row operation**
    - **Up to 75, 35, and 50% cpu reduction, respectively, compared to V7**
  - **Bind option release commit in data sharing**
  - **I/O-intensive workload**
    - **Long-term page fix and/or 180 CI limit removal help**

More CPU Tuning - continued

- **PQ89070 8/04 to avoid locks in Select Into in CS and Currentdata Yes**
- **PQ95867 12/04 for faster DB2 storage management**

➔ **Ultimately, aggressive exploitation of V8 performance features via application rewrite or configuration change, eg multi-row operation, can make V8 perform significantly better than V7 in affected areas.**
QMF multirow fetch PQ99482

Notes

- More regression possible if
  - Non padded index (default with V8 install) with small varchar columns
  - DPSI in some SQL calls
  - Many column processing
    - Consider Alter to less expensive column type
      - *V5 Alter Varchar length, but Varchar no longer necessary to alter length*

Caution on observed CPU time increase

In general, higher %cpu accounting increase

- But lower %cpu increase, or even reduction, in MSTR, DBM1, and IRLM address spaces possible
- For lower %cpu increase overall

Example: IRWW 11/04

	Non data sharing	Data sharing
Acctg	+4%	+14%
MSTR/ DBM1/ IRLM	-15%	-48%
Total	0%	-8%

Notes

- Less DBM1 SRB time from
  - Long term page fix option especially in prefetch and write i/o
  - 180 VSAM Control Interval limit removal in list prefetch and castout I/O PQ86071 / PQ89919 6/04
  - z/OS 1.4 Coupling Facility Level 12 batching of multiple CF write and castout read requests into one CF access
    - Bigger benefit for Insert/Update/Delete-intensive application
- Less MSTR SRB time from
  - Default immediate write change from NO(Phase2) to Phase1
    - MSTR SRB shifted to Acctg TCB time – no net change
- Less IRLM time from
  - Reduced global and false contention for pageset / partition locks when release commit in data sharing
    - Less need for release deallocate bind option

V8 Virtual Storage Usage in DBM1 Address Space
'typical' V7 below 2GB storage usage shown

2GB

Buffer pool 0 to 1GB
Dataspace lookaside buffer 0 to 100MB
Buffer control blocks 1MB to 300MB
RID pool 4 to 80MB
Castout engine work area 0 to 80MB
Compression dictionary 0 to 500MB
RDS OP pool 5 to 500MB
Dyn Stmt Cache control blocks 0 to 200MB
BufMgr/DataMgr Trace Table 10 to 100MB

Other EDM pool 20 to 300MB
Local Dynamic Statement Cache 0 to 300MB
Thread and stack storage 50 to 800MB

2GB

Notes

- **LOB, global dynamic statement cache, and dataspace buffer pool are not shown here as these can be in dataspace rather than below 2GB DBM1 address space in V7.**
  - **These are all in above 2GB DBM1 address space in V8.**

Virtual storage - continued

- **DBM1 virtual storage constraint relief improves scalability of performance**
  - **As the processor power continues to grow, linear scalability, or ability to exploit increasing processor power without encountering a bottleneck which prevents the full CPU usage, becomes more important.**
  - **Bigger buffer pool and cache to reduce i/o bottleneck and CPU overhead**
  - **Without 64bit support, it was difficult to exploit more than 20GB of real storage**
    - **Up to 32GB on z800, 64GB on z900, 256GB on z990**
  - **CPU-Storage trade-off**
- **However, thread storage is still below 2GB in V8**
  - **Hence, maximum number of threads supported, such as CTHREAD (2000) and MAXDBAT (1999), is not increased.**

Notes

- **Bigger DBM1 virtual storage constraint relief if V7 with bigger**
  - **Buffer pool below 2GB**
  - **Buffer control blocks for virtual pool, hiperpool, dataspace buffer pool**
  - **Dataspace buffer pool lookaside buffer**
  - **Compression dictionary**
  - **Castout buffers**
  - **RID pool, sort pool (in RDS Op pool)**
  - **Data Manager/Buffer Manager trace table**
  - **Dynamic statement cache control block**
  - **IRLM PC=NO**
- **Less relief from more**
  - **User and system thread storage with associated stack**
  - **Local dynamic statement cache**

Estimation of V8 Below 2GB DBM1 Use, based on V7 Stats

- **The following <u>average</u> estimates are <u>very preliminary and</u> <u>subject to change</u> as more customer data become available**
- **Thread storage: 0 to +90% (0 for Image Copy utility, +40% for system, +40 to 90% for user thread)**
- **Stack storage: +100%**
- **Dynamic statement cache: +60%**
- **EDM pool: roughly the same**
- **RID pool: -90%**
- **Others: -100%**
- **V7 PQ99658 Storage stats in class 1**

Notes

- **For a fair comparison, use the same pool size, # of threads, etc. instead of the defaults which may have changed.**
- **Bigger thread/stack storage in V8 for**
  - **Long names, keys, statements, other new functions**
  - **A portion of RDS op pool for dynamic SQL**
  - **More system agents**

➲ **How much more room in DBM1 address space below 2GB depends on % of storage used for threads, stacks, and local DSC versus others**

Customer 1 (1/05)	V7 measured	V8 estimated
Virtual Pool	0 MB	0 MB
Buffer control block	78	0
Dataspace lookaside buffer	5	0
Castout buffer	38	0
EDM pool	118	118
Compression dictionary	340	0
1170 system agents	73	103
25 user threads	18	35
RDS OP pool	29	0
RID pool	92	10
DSC control block	53	0
Trace table	15	0
Stack storage	49	98
TOTAL DBM1 below 2GB	908MB	364MB

Notes

- **Negligible local dynamic statement cache**
- **No virtual buffer pool as dataspace buffer pool is used instead.**
- **Assumes**
  - **Same number of system agents in V8**
  - **90% increase in user thread storage**

➲ **Good DBM1 virtual storage constraint relief for this customer**

- ➲ **Even though dataspace buffer pool was used exclusively**
- ➲ **Because of large compression dictionary and small thread/stack storage**

Customer 2 (1/05)	V7 measured	V8 estimated
Virtual Pool	98 MB	0 MB
Buffer control block	13	0
Dataspace lookaside buffer	6	0
Castout buffer	17	0
EDM pool	71	71
Compression dictionary	51	0
837 system agents	64	90
493 user threads	291	553
RDS OP pool	420	0
Dynamic Statement Cache	41	66
DSC control block	42	0
Trace table	36	0
Stack storage	143	286
TOTAL DBM1 below 2GB	1293MB	1066MB

Notes

- **Negligible RID pool**
- **Both virtual buffer pool & data space buffer pool used here**
- **Large RDS OP pool due to lots of concurrent sort**
- **Assumes**
  - **Same number of system agents in V8**
  - **90% increase in user thread storage**

➲ **DBM1 virtual storage constraint relief not as good as the Customer 1 because of large thread/stack storage**

Virtual Storage-Related Tuning

- **Bind option release commit vs deallocate**
  - **Commit releases pageset/partition locks, RDS sections, and storages sooner, resulting in better concurrency and less DBM1 virtual storage usage**
    - **Recommended default**
  - **Deallocate holds on to these resources longer, resulting in possibly less CPU time (0 to 20%)**
    - **Recommended only for frequently-executed, high-volume, and performance-sensitive packages or plans**
    - **DB2PM/PE Acctg Report Short shows a list of pkgs/plans executed along with #occurrences and avg# SQL statements, elapsed time, and cpu time.**

Notes

- **If BP size increase is considered,**
  - **Make sure there is sufficient real storage to back it up 100%.**
    - **For BP with 100,000 buffers as an example, if 99,999 real storage frames are available, then every I/O could result in paging because of Least Recently Used buffer steal algorithm.**
  - **Deferred write threshold (VDWT) may need to be reduced in order to avoid "hiccup effect" at checkpoint.**
    - **Eg 5% VDWT of much bigger BP can have many more updated pages to be written out at checkpoint.**
- **V8 deferred write threshold default change**
  - **Buffer pool threshold (DWT): 50% to 30%**
  - **Dataset threshold (VDWT): 10% to 5%**
- **If migrating from VP+HP configuration with new BP size = VP+HP size, adjust BP thresholds, which are based on VP but not HP, appropriately.**

Virtual Storage-Related Tuning continued

- **If necessary, reduce MAXKEEPD to reduce local DSC**
  - **Rely more on global DSC which is above 2GB**
- **V8 PQ96772 2/05 to move dynamic statement cache control blocks above 2GB**
- **CONTSTOR and/or MINSTOR to reduce thread storage, especially for >1MB per thread storage**

Notes

- **Extended CSA size could be reduced in V8 if IRLM PC=NO was used in V7, enabling additional DBM1 virtual storage available below 2GB**
- **V7/V8 PQ98043 1/05 to reduce excessive stack storage for inactive or disconnected threads with dynamic SQL**
- **Stay current with service.**
- **Watch Info APAR II10817.**

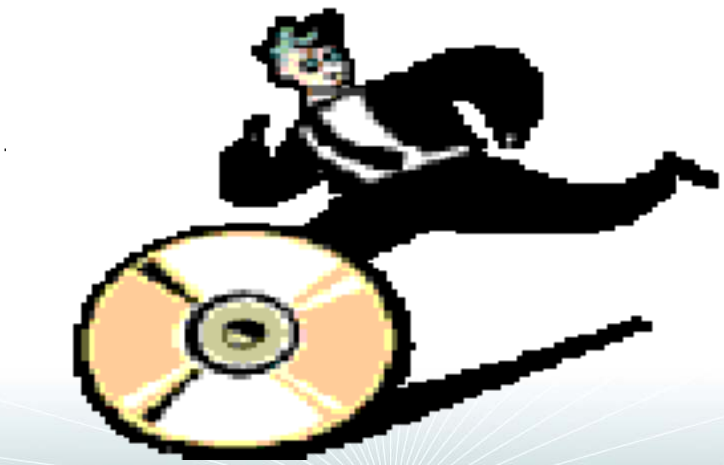
Real Storage (RS) Usage

- **From V1 R1 in 1985 to the present, real storage usage growing at about 20 to 30% yearly to support performance scalability**
  - **More and bigger buffer pools, other pools, threads, …**
- **V8 continues a similar trend**
  - **By removing bottlenecks which would have prevented the exploitation of bigger real storage**
  - **If everything under user control is kept constant, 1 to 20% increase in real storage typically observed**
    - **Less %increase for larger DB2 subsystem**
    - **Bigger %increase for smaller DB2 subsystem**

Notes

- **Without 64 bit support, difficult to exploit more than 20GB of real storage because of DBM1 virtual storage constraint**
  - **z800 up to 32GB, z900 64GB, z990 256GB, System z9 512GB**
- **Example of more real storage usage**
  - **Higher default and maximum buffer pool size, RID pool size, sort pool size, EDM pool size, …**
  - **Bigger and possibly more threads**
  - **Bigger modules, control blocks, internal working storage**
  - **More deferred write, castout, and GBP write engines (300->600 max each)**
  - **More parallelism enabled**
    - **Parallel sort for multiple tables in composite**
    - **Parallel multi-column merge join**

Elapsed Time Analysis



NOTES

- Accounting report (not trace) by connection type most useful for initial analysis
  - DB2PE ACCOUNTING REPORT LAYOUT(LONG) ORDER(CONNTYPE) EXCLUDE(PACKAGE(*)) to group by thread connection type such as TSO, CICS, DB2CALL, RRS, IMS, DRDA, etc. for the period of interest.
  - Also DB2PE STATISTICS REPORT LAYOUT(LONG) for the corresponding period extremely desirable

Accounting Class 1 and 2

AVERAGE	CLASS1	CLASS2
ELAPSED TIME	233ms	19ms
CPU TIME	2.95ms	2.71ms
WAIT TIME		14.76ms
NOT ACCOUNTED TIME		1.31ms

- For most cases
  - Class 1 for application + DB2 time
  - Class 2 for DB2 time only
- CICS without TS 2.2 or later threadsafe option
  - Class 1 CPU for task switch + DB2 time
  - Class 2 for DB2 time only

NOTES

- **Accounting class 1 and 2 for elapsed time and CPU time**
- **Accounting class 3 for elapsed time breakdown such as**
  - **I/O wait time**
  - **Lock/latch wait time**
  - **Commit time**
- **For thread reuse, class 1 can be much higher than class 2 time because the time from commit to the first SQL call of next transaction to reuse the thread is also included.**
- **NOT ACCOUNTED TIME = Class 2 elapsed time - class 2 cpu time - class 3 wait time**

High NOT ACCOUNTED time – 2 most likely causes

- CPU wait under high cpu utilization, especially with lower dispatching priority
  - **E.g. goal mode with low priority for DB2 address space compared to DDF enclave, CICS, WebSphere address space, or DDF enclave with SYSOTHER (discretionary)**
- Excessive detailed online tracing with vendor tools

NOTES

- Other causes are much less frequent and widely varied
  - **Some events not being captured by DB2, but more events are being captured in newer versions**
  - **Online support document: http://www.ibm.com/support/docview.wss?rs=64&context=SSEPEK&uid=swg21045823**

Accounting Class 3

SUSPENSIONS	TOTAL TIME	#EVENTS
LOCK/LATCH	0.11ms	0.3
SYNC DATABASE I/O	8.73ms	8.86
SYNC LOG WRITE I/O	1.64ms	0.49
OTHER READ I/O	2.64ms	0.76
OTHER WRITE I/O	0.004ms	0.00
SERVICE TASK	1.60ms	0.47
.....		
TOTAL CLASS 3 WAIT	14.76ms	10.88

- **Class 3 acctg strongly recommended: Negligible overhead except when high internal DB2 latch contention, eg over 10000/sec**

NOTES

- Lock/Latch wait = Lock wait + IRLM latch wait + internal DB2 latch wait
  - In the rare case of over 10000 per second, disabling class 3 may significantly bring down class 1 and 2 cpu time.
- Sync I/O wait = wait for read or write i/o by this application agent
  - Avg time = 8.73ms/8.86 = 0.985ms
- Other read I/O wait = wait for read i/o by another application agent or prefetch engine
- Other write I/O wait = wait for write i/o by another application agent or write engine, may include some time waiting for log write-ahead

I/O wait time tuning

- Buffer pool tuning - discussed in Buffer Pool section
- I/O configuration tuning
  - Make sure of sufficient I/O resources
  - Faster device, such as ESS 800 or DS8000 as needed
  - ESS PAV (Parallel Access Volume) beneficial if I/O contention with high IOSQ time in RMF
  - I/O striping

NOTES

- I/O striping
  - Appendix B of DB2 for z/OS and OS/390 V7 Performance Topics redbook SG24-6129
  - Clearly beneficial for active log
    - 2.5 times throughput improvement with 4 stripes
  - Potentially beneficial for segmented tablespace or other non partitioned tablespace, non partitioning index, LOB, work file, ...
    - Striping for 4K page only in V7
    - Striping for 4K, 8K, 16K, and 32K page in V8
  - Partitioning if possible can have better performance
    - Maximum of 16 stripes versus 254 in V7 or 4096 in V8 partitions
    - Optimizer understands partitions but not stripes
    - But a combination of partitions and stripes could result in better performance if partition activities are heavily skewed.

Service Task Wait

- Dataset Open / Close
  - Up to 20 tasks for parallel Open
  - V8 with z/OS 1.6 increases the maximum number of open datasets DSMAX to 65000 from 32767 PQ96189
    - 80000 with z/OS 1.7, 100000 with z/OS 1.7 GRS SPE and V8 PK13287
    - When #open datasets reaches 99% of DSMAX, MIN(3%DSMAX, 300) will be physically closed
  - DFSMS 1.5 ECS recommended
    - ECS = Enhanced Catalog Sharing to move VSAM Volume Record for both user and master catalogs to CF from DASD
      - Eliminates Reserve/Release and Vsam Volume Data Set I/O for much faster dataset Open
  - Avoid GRS processing if no data sharing and no CF
    - GRS Star can be up to 3 times faster than GRS RING

Service Task Wait - continued

- Update commit (commit after any Insert, Update, or Delete)
- SYSLGRNX Update
- Dataset Extend/Define/Delete
  - Extend for preformatting, takes 0.05 to 1 sec each time depending on device type and alloc unit/size
  - x'09' lock wait in async preformat starting in V7
  - Dataset Delete/Define when Stogroup used
  - Ifcid 258 contains dbid/obid for Dataset Extend
  - Ifcid 92 contains Define Cluster statement for Define/Delete
- Other
  - Time waiting for server in requestor accounting
  - VSAM catalog update
  - Parallel query
  - And many others which are typically infrequent

How to find which service task invoked

- Performance trace of IFCID 46 thru 50
  - IFCID 46 for service task switch
    - 47 and 48 for Begin and End of SRB service task
    - 49 and 50 for Begin and End of TCB service task
  - Shows all nested service tasks
- Or IFCID 170 and 171 for less output
  - Shows one (outermost) service task for each event in class 3 accounting

NOTES

- Correlation name of system agent (SYSOPR planname) in performance trace:

  e.g. 021.OPNL for SYSLGRNX update in pageset open

- Chapter 9 of DB2 Diagnosis Guide and Reference contains a list of system agent correlation identifiers as well as a list of Resource Manager id's displayed in IFCID 46, 47, 49, and 170.

Commonly observed service tasks

- Commonly observed service tasks and RMID/FC identifiers as shown in IFCID 46, 47, 49, and 170
  - Update Commit x'03'/x'49'
  - Dataset Open x'0A'/x'59'
  - Dataset Close x'0A'/x'4F'
  - Syslgrnx Update x'15'/x'44'
  - Dataset Extend x'12'/x'65'
  - Dataset Delete x'12'/x'6A'
  - Dataset Define x'12'/x'68'
  - Dataset Reset x'12'/x'6C'

NOTES

- RMID = Resource Manager ID
- FC = Function Code
- Commonly observed OTHER SERVICE TASKs
  a. Parallel query cleanup x'14'/x'77' or x'84'
     - Shows up in repeated execution of short-running queries in parallelism.
     - Use higher SPRMPTH zparm value to reduce this (default=120(ms))
     - Of course, for a long-running query, parallel query with degree ANY can result in many times reduction in elapsed time
  b. VSAM catalog update x'0A'/x'94'
  c. Requestor waiting for server response x'1B'/x'8F'

Elapsed Time in Data Sharing

- Possibly higher sync log write I/O waits from index split or massive delete, as updated index pages are written to GBP together with log force writes
- Global Contention Wait
  - Sum of IRLM, XES, and False contentions in Data Sharing Locking section
- Update commit service task wait includes GBP write and P-lock unlock for each updated page
- Lock/Latch wait can increase because of possibly higher IRLM latch contention
- Asynchronous CF request wait

NOTES

- To minimize P-lock contention/negotiation
  - First find out from GBP statistics what is contributing to P-lock contention
  - If spacemap page P-lock and/or data page P-lock, consider using Member Cluster tablespace
  - If index page P-lock, consider setting VDWQT=0 to free P-locks as soon as possible
  - If index tree P-lock, consider
    - %freespace tuning
    - 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

CPU Time Tuning

NOTES: agenda

- Minimizing #SQL calls, columns, host variables, predicates evaluated, SQL statements, rows searched
- OPT for N ROWS
- Existence check
- Dynamic SQL, JDBC/SQLJ
- Bind option acquire and release
- Thread reuse
- DB2 trace
- Distributed / stored procedure
- Catalog statistics check
- Compression, Encryption, Row-level Security

Minimize SQL Calls to Reduce API Overhead

- Filter out unnecessary rows by adding predicates rather than by application program checking
- Use of DB2 column functions rather than application program code
- Example: find how many employees make more than $10,000/month
  1. Select, fetching all 100000 employee rows
  2. Select Where Salary>10000, fetching 1000 rows
  3. Select Count Where ..., fetching 1 row
  - ➢ 100 times CPU time reduction possible from API elimination

NOTES

- API (Application Program Interface) overhead = a round-trip cost between application program address space and DBM1 address space

  Also between
  - DDF address space and DBM1 address space (reduced in V9)
  - Stored Procedure address space and DBM1 address space

Minimize #SQL Calls - continued

- Singleton SELECT is more efficient than OPEN, FETCH, CLOSE
- Fetch First N Rows Only in V7
  - Limits the number of rows fetched to avoid fetching unwanted rows
  - Singleton Select (or SELECT INTO) can be used with Fetch First 1 Row even if multiple rows qualify
    - Avoids -811 SQLCODE
    - V8 supports ORDER BY for more meaningful query
  - Bigger improvement possible for CICS attach
- UPDATE without cursor is more efficient than OPEN, FETCH, cursor UPDATE, CLOSE
  - Up to 30% (possibly more if CICS) CPU time saving possible from singleton Select or Update compared to cursor operation

NOTES

- CICS TS 2.2 with Threadsafe option can avoid task switch for each SQL call
  - **See Redbook DB2 for z/OS and OS/390 V7 Selected Performance Topics SG24-6894**
  - **Up to 50% CPU reduction for simple (short-running) Fetch SQL calls**
  - **10 to 20% CPU reduction for other simple (short-running) SQL calls**
  - **Negligible impact for complex (long-running) SQL calls**

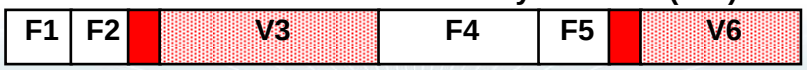
Minimize #SQL Calls - continued

- Reducing #SQL calls improves
  - **API pathlength**
  - **Processor MIPS for row processing**
    - **Up to 2 to 3 times processor MIPS improvement possible from high-speed processor cache hit by repeated execution of a small set of modules/instructions and reduction in data moves**
- V8 multi-row operation can significantly reduce the number of SQL calls issued
  - **Up to 50% cpu reduction for simple (short-running) local Fetches, more for distributed**

Minimize #Columns and Host Variables Referenced in SQL Calls

- Increasing order of cost
  - Local EBCDIC least -> ASCII or UNICODE or DRDA -> Single byte conversion -> Double byte conversion
  - Integer/char least and date/time/timestamp most expensive
- Try to avoid unnecessary columns
  - Doubled CPU time possible with 100 additional columns/host variables

- Put Varchar to end of row when many columns (>20)



Minimize #Predicates Evaluated

- Place most filtering predicates first in AND. (for predicates of the same type)

WHERE HOME_STATE='MONTANA'	FF= 1%
AND HAIR='BROWN'	FF=10%
AND SEX='MALE'	FF=50%

  - Weighted average of 1.01 predicates evaluated
  - If sequence of predicates is reversed, then the weighted average is 1.55, or 50% more predicate evaluation, which can lead to up to 20% cpu increase.
- Conversely, place most filtering predicates last in OR and IN-list without ACCESSTYPE=N.

  eg STATE IN ('NEW YORK','FLORIDA','MONTANA')

NOTES

- FF = Filter Factor, or % of rows qualifying after predicate evaluation
- Probabilities of 1, 2, or 3 predicates evaluated = 0.99 for 1 predicate,
  0.009 (=1%*90%) for 2 predicates, 0.001 (=1%*10%) for 3 predicates
  for a weighted average of 1.01 predicates.
- Sequence of predicate evaluation: sargable before non sargable, = before range, range before subquery, single value non correlated subquery before other non correlated or correlated subquery
  - ➡ Predicates of the same type are executed in the order specified
- In AND, predicate evaluation is terminated as soon as False condition is reached. In OR (including IN-list), predicate evaluation is terminated as soon as True condition is reached.
- For matching index IN-list, values in IN-list are sorted prior to execution. Thus, any user specification is overridden and has no effect.

Minimize #SQL Statements in a Program Where Possible

```
DO ....
   SELECT or INSERT or DELETE or UPDATE
END
                instead of
SELECT, INSERT, DELETE, or UPDATE
SELECT, INSERT, DELETE, or UPDATE
SELECT, INSERT, DELETE, or UPDATE
```

- Reduces EDM pool and thread storage
- Reduces allocate/deallocate cost at SQL execution and commit or deallocation
- Better exploitation of sequential detection and index lookaside
  - Potentially fewer Getpages, Lock requests, and faster I/O

NOTES

- RDS section and storages may be acquired for each SQL statement at
  - Create Thread (bind option acquire at allocate) or SQL execution time (acquire at use)

and released at

  - Terminate Thread (release at deallocate) or Commit time (release at commit)

Minimize # rows searched

- Try to get the maximum matching index columns for the best index filtering
- Insure predicate comparison for the same data type and length
  - Example: "where indexed-column=host-variable“
  - Especially prior to V8
    - V8 made most typical unlike data type comparisons stage 1 or sargable and indexable

Dynamic SQL

- Reduce dynamic bind frequency via
  - Dynamic statement caching with CACHEDYNAMIC YES
- Incremental bind in accounting
  - Static plan/package with VALIDATE(RUN) and bind time failure
  - Static SQL with REOPT(VARS), or referencing Declared Temp Table, or private protocol in requestor

JDBC/SQLJ

- Use CACHEDYN YES for JDBC, or better yet use SQLJ
- Select/Update/Insert required columns only
  - More important in JDBC/SQLJ environment
- Store numeric as smallint or int to minimize conversion and column processing cost
  - Relative cost: Integer (lowest) -> Float -> Char -> Decimal -> Date/Time -> Timestamp (highest)
- Match Java and DB2 data type
  - V8 enhancement for non-matching data type

NOTES

- 5 to 25% throughput improvement observed for SQLJ compared to JDBC in one measurement experiment.
- Comprehensive set of performance tuning recommendations for JDBC/SQLJ in DB2 for z/OS and OS/390 V7 Selective Performance Topics redbook SG24-6894

OPTIMIZE FOR N ROWS

- OPTIMIZE FOR N ROWS for better access path selection
  - Optimizer informed that partial results are to be returned
    - Tends to encourage non clustered index scan and nested loop join
    - Tends to discourage merge join, sequential prefetch, RID processing, hybrid join
- Use of index on sort key column(s)
  - Example: getting top 10 taxpayers among 300 million Americans

Example

- To get the top ten tax payers among 300 million Americans, 300 million rows must be accessed and sorted if there is no index on tax_paid column.
  - With a non clustering index but no OPTIMIZE clause, Optimizer may decide tablespace scan followed by sort based on cost estimate or insufficient RID pool available
  - With a non clustering index AND OPTIMIZE FOR 10 ROWS, only 10 rows need to be accessed via index with no sort.
- OPTIMIZE FOR for access path selection as well as network blocking
  - When OPTIMIZE FOR is missing, V7 FETCH FIRST can influence access path selection but not network blocking
  - See Distributed section coming later on network blocking

Existence Check

- SELECT FROM table WHERE EXISTS (SELECT FROM SYSIBM.SYSTABLES WHERE TYPE='A') …..
  - In V7, all qualifying rows in this EXISTS subquery are retrieved and stored in a work file.
    - Select from SYSIBM.SYSTABLES where Type='A' Fetch First 1 Row Only followed by Select from outer table can be much faster.
  - In V8, this subquery execution is terminated as soon as a first qualifying row is found.

NOTES

- A typical medium to large V7 DB2 system may contain an average of 25,000 rows in SYSTABLES and 18%, or 4500 rows, represent Type Alias
  - **Thus, 4500 rows are retrieved and stored in a work file prior to V8 in this example.**

Bind Option on Resource Acquire/Release

- **General recommendation**

Acquire USE/Release COMMIT for all except

Acquire USE/Release DEALLOCATE

- **For frequently-executed, high-volume, and performance-sensitive packages or plans**
  - **Can save 0 to 20% cpu at a possible cost in concurrency and DBM1 virtual storage usage**
- **Accounting Report Short shows a list of packages/plans executed along with #occurrences and average # SQL statements, elapsed time, and CPU time.**

NOTES

- Advantages of each

  Acquire at first use (default)
  - Resource allocated only when needed
  - Less EDM pool storage

  Acquire at allocate
  - Cost per resource allocated is cheaper

  Release at commit (default)
  - Pageset/partition locks, RDS sections, and storages freed sooner

  Release at deallocate
  - Efficient freeing of storages
  - Less pageset/partition locks

Thread Reuse

- Thread reuse for 5 to 20% cpu time reduction for light transactions

NORMAL TERMINATION	AVERAGE	TOTAL
NEW USER	1.00	174752
DEALLOCATION	0	0
RESIGNON	0	0
INACTIVE	0	0

- All except DEALLOCATION indicate successful thread reuse.

NOTES

- DEALLOCATION = Normal thread termination without thread reuse
- NEW USER = CICS with authid change, IMS
- RESIGNON = CICS without authid change (new user broadcast without authorization check when token=YES on type=ENTRY statement in RCT), RRS
- INACTIVE = DDF

'Typical' CPU Overhead of DB2 Trace

DB2 accounting

- Class 1: less than 5% cpu overhead
- Class 2: 1 to 10% cpu overhead
  - Class 2 acctg and class 3 performance trace overhead can be higher for Fetch-intensive applications, up to 20% and 100% respectively.
  - V8 multi-row Fetch can make this overhead negligible
- Class 3: less than 1% cpu overhead
  - Much higher than 1% cpu overhead for class 3 acctg has been observed in a rare situation of very high internal DB2 latch contention rate, eg over 10000/sec.
- Class 7 and 8: less than 1% cpu overhead

NOTES

- Monitor trace: similar to accounting
- DB2 performance trace with default classes (1-3): 5 to 100% cpu overhead
- DB2 global trace: 10 to 150% cpu overhead
  - Global trace is described in DB2 Diagnosis Guide and Reference.

'Typical' CPU Overhead of DB2 Trace - continued

DB2 statistics: negligible

- DB2 audit trace with all classes: less than 5% cpu overhead
- Do DISPLAY TRACE to make sure of no unnecessary or unintentional trace

<u>Check other traces also</u>

- CICS monitoring facility, internal trace, auxiliary trace, ...
- CAF DSNTRACE DD statement including DD DUMMY
- IMS monitoring facility, ...
- z/OS trace
  - D TRACE,COMP=ALL
  - TRACE CT,OFF,COMP=xxxx to turn off CTRACE
  - Capture only SMF data needed

NOTES

Usage pricing (SMF 89) – default is on

- SMFPRMxx member of PARMLIB SYS(TYPE(0:88,90:255) ...) to disable
- Bigger impact with more concurrent active threads, 0 to 15% overhead reported
  - ⇨ Less than 1% overhead in V8 with SMF89=NO (no detail, still get data)

Distributed/Stored Procedure

- Stored procedure to avoid DRDA overhead for each SQL call
  - Example: 10 Select, Insert, Update, and/or Delete calls in stored procedure
    - Results in 600us instead of 2100us overhead (10 SQL calls * 210us per SQL call) on z900 (2064-1) processor
    - Also faster response time because of as low as 1 rather than 10 message send/receive

NOTES

- To minimize stored procedure overhead,
  - STAY RESIDENT YES to avoid reloading (default=NO)
  - COMMIT-ON-RETURN with Work Load Manager-managed stored procedure (default=NO)
  - PROGRAM TYPE of SUB instead of MAIN

Distributed/Stored Procedure - continued

- Block fetch to reduce the number of messages
  - Automatic exploitation of V8 multi-row Fetch by DRDA further reduces cpu time
  - V8 FOR READ ONLY KEEP UPDATE LOCKS, instead of FOR UPDATE, enables block fetch
- Inactive thread to reduce both CPU time and storage usage in general
  - Zparm CMTSTAT INACTIVE instead of default ACTIVE

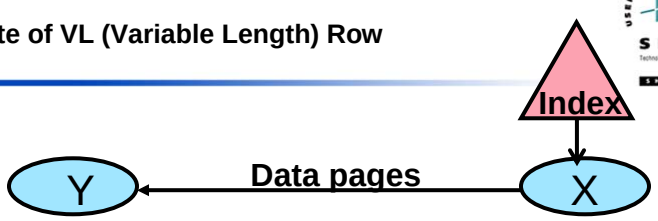
NOTES

- **Other ways to reduce #msgs across network**
  - **OPTIMIZE FOR N ROWS**
    - **If N=1, 2, or 3, 16 row block. If N>3, N row block**
    - **So OPT FOR 1 ROW takes 4 line turnarounds to fetch 50 rows**
  - **FETCH FIRST N ROWS ONLY for fast implicit close**
    - **Network blocking not affected**
    - **V8 PK11355 9/05 Restore block fetch when Fetch First 1 Row Only followed by Fetch Only**

Catalog stats check

- Near/Farindref in Systablepart catalog contains the number of rows stored on an overflow page as a result of variable-length or compressed row update
  - **Consider Reorg if >10% (>5% in data sharing) to reduce the number of I/O's, Getpages, and lock/unlock requests**
  - **Even with isolation CS and CurrentData No, both pointer and overflow records or pages are locked in V7**
    - **V8 supports lock avoidance for overflow record or page, resulting in up to 50% reduction in Lock/Unlock requests**

Notes: Update of VL (Variable Length) Row



- When an updated VL, or compressed, row can not fit in page X,
  - **New row stored in a different page Y**
  - **Its pointer stored in page X to avoid index update**
  - **If updating later with small row, it can be put back on home page X, again without index update (overflow row is deleted)**
- Problems
  - **Potential doubling of data I/O, Getpage, and locking**
  - **No lock avoidance for pointer/overflow record or page in query**
- Prevalent problem today as majority of large data are compressed

Catalog stats check - continued

- Pseudo_del_entries in Sysindexpart catalog contains the number of pseudo-deleted entries
  - **Consider Reorg Index if >10% (>5% in data sharing) to reduce index scan cpu time and lock/unlock requests**
  - **Additional lock/unlock requests possible in Insert, Update**
    - **Eg S-lock/unlock in Insert into unique index if a pseudo-deleted rid already exists for the key being inserted to make sure a rid is committed before replacing with a new rid.**

Compression

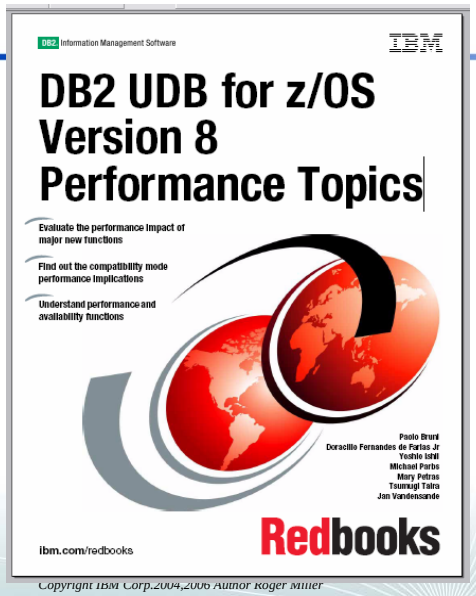
- z900(2064-1) up to 5 times faster than G6 turbo(9672), instead of normal 1.15 to 1.3 times, in compression and decompression
- z990(2084) 1.4 times additional speed up compared to z900 turbo in decompression
  - **z990 1.5 times faster than z900 turbo on average**
  - **But decompression is 1.5x1.4=2.1x faster**
- Decompression cost a function of compression ratio
  - **Use compression only if compression ratio >20%**
- Reduced disk, buffer pool, log, i/o, but potential increase in cpu
  - **Less than 5% increase typical for online transaction, higher for sequential data scan**

Encryption and Row-level Security

- V8 Column-level Encryption more expensive than Row-level Encryption from Data Management Tools
- Overhead of Row-level Encryption
  - **Similar but somewhat more expensive than DB2 data compression**

➲For encryption performance, z9 or z990 (2084) rather than z900(2064-1) processor is strongly recommended

- V8 Row-level security overhead in the same ballpark as DB2 data compression



Performance Details

Measurements

Service

462 pages

Reference

- **V8 manuals, especially Performance Monitoring and Tuning section of Administration Guide**
- **Redbooks at www.redbooks.ibm.com**
  - **DB2 UDB for z/OS V8 Technical Review SG24-6871**
  - **DB2 UDB for z/OS V8 Everything you ever wanted to know… SG24-6079**
  - **DB2 UDB for z/OS V8 Performance Topics SG24-6465**
- **More DB2 for z/OS information at www.ibm.com/software/db2zos**
  - **E-support (presentations and papers) at www.ibm.com/software/db2zos/support.html**