DB2 UDB for z/OS V8 Application and System Performance

Akira Shibamiya, presented by Roger Miller

Monday March 1, 2005 3:00 PM

Anaheim Session 1333

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 measurements are still ongoing and some product changes may result.**

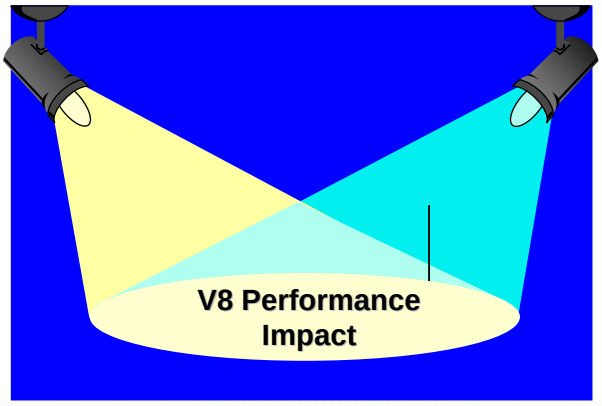
Acknowledgment and Disclaimer

- 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.

Outline

- Highlights of V8 Performance Impact
- Utility Performance
- Query Performance
- SQL Performance
- Miscellaneous Performance Considerations

Highlight of V8 Performance Impact

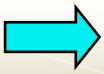


Prerequisites

- Prerequisites for V8
  - zSeries: z800, z890, z900 or z990
  - z/OS 1.3 which supports 64 bit, 1.4 or 1.5 for some function
  - IRLM 2.2 which supports 64 bit
  - Note
    - DBM1 and IRLM address spaces use 64 bit.
    - MSTR, DDF, Stored Procedure and application address spaces use 31 bit.
- Performance Highlight
- DBM1 Virtual Storage Constraint Relief
- Real Storage Usage

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

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

Performance Highlight - continued

For those applications not taking any advantage of V8 performance enhancements,

Some CPU time increase is unavoidable to support a dramatic improvement in user productivity, availability, scalability, portability, family consistency.

DBM1 virtual storage constraint relief with 64 bit

Long names, long index keys

Longer and more complex SQL statements

Unicode catalog

No change in I/O time

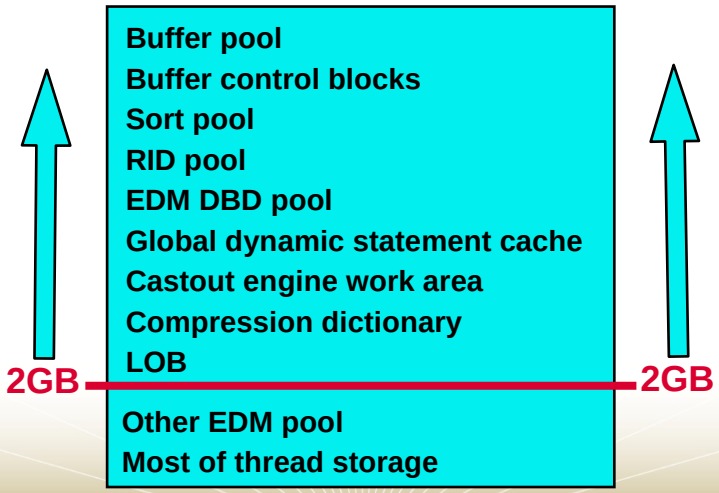
→Plan to take advantage of V8 performance

Page fix, rebind, multirow fetch & insert, ...

CPU Performance Regression: Subject to change

- Performance objective is less than 10% average regression
- Typical customer workload regression is expected to be 5 to 10% higher on average, differing by work load:
  - 0 to +15% online transaction
  - -5 to +10% transaction in data sharing
  - -5 to +20% batch
    - -5 to +5% insert
    - +5 to +20% fetch, select, update
  - -10 to +15% batch data sharing
  - -20 to +15% batch DRDA
  - -5 to +10% utility
  - -20 to +15% query
- Options with significant potential to offset an increase include multirow fetch, multirow insert, long term page fix and rebind

DBM1 Virtual Storage Constraint Relief



DBM1 Virtual Storage - continued

- Allows scalability of performance
  - As the processor power continues to improve, 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
  - More concurrent threads to increase throughput
  - Use of thread private pool rather than shared pool to reduce class 32 latch contention with many concurrent active threads (eg V7 PQ81904 2/2004)

Synergy & scaling new hardware

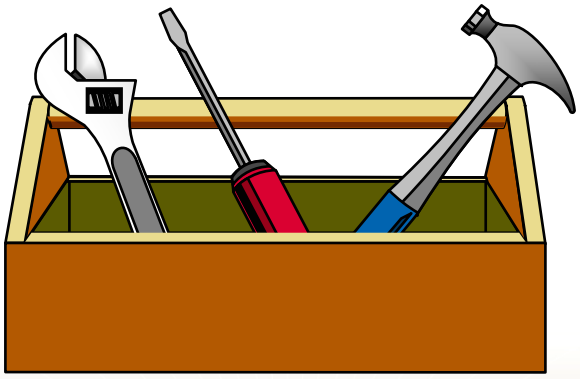
- Using bigger, faster hardware without premature constraint
  - zSeries z990 5/2003 GA
    - 1.4 to 1.6 times higher MIPS compared to z900 turbo (single engine)
    - 1.8 to 2 times higher MIPS compared to z900
    - 256 GB real storage up from 64 GB for z900
    - Without 64 bit support, it was difficult to use more than 20 GB of real storage effectively.
  - ESS 800 (2105-800) 8/2002 GA
    - 0.08 to 0.12 ms/4K page in sequential read/write compared to 0.13 to 0.2 ms/page for ESS F20 and FICON

Real Storage Usage

- From V1 R1 in 1985 to the present, real storage usage growing at about 20 to 30% per year to support performance scalability
  - More and bigger buffer pools, sort pools, threads, ...
- V8 continues a similar trend
  - By removing bottlenecks which would have prevented the use of bigger real storage
  - For same buffer pool size, etc, 1 to 10% increase in real storage typically expected for medium & large DB2 subsystems
    - 4 to 5% increase in one SVL measurement
    - For smaller system, more than 10% increase likely

- Example of more real storage usage
  - Higher default and maximum buffer pool size, RID pool size, sort pool size, EDM pool size, authorization cache
  - More user threads
  - Bigger modules, control blocks, internal work storage
  - More deferred write engines and castout engines (300->600 max)
  - More parallelism enabled, eg
    - Parallel sort for multiple tables in composite
    - Parallel multi-column merge join

Utility Performance



Utility Performance Outline

- Catalog migration
- 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

Catalog Migration

- V6 to V7
  - Expected to take 10 to 100 seconds, depending on the size of catalog/directory (medium to large) and disk / processor model
- V5 TO V7
  - Expected to take 2 to 20 minutes

Version Migration

- 3 possible migrations to V8
  - V5 to V7 to V8
    - V6 end of marketing June 2002
    - V6 end of service June 30, 2005
    - V5 end of service December 31, 2002
  - V6 to V7 to V8
  - V7 to V8

Catalog Migration

- V7 to V8 Compatibility Mode
  - Currently 0.2 to 10 minutes observed, depending on the size of catalog/directory (medium to large), in non data sharing
  - Compatibility mode
    - DB2 catalog in EBCDIC
    - Fallback to V7 allowed
    - This mode supported as long as V8 is
    - Check for type 1 index
      - ➔ If any found, catalog migration rolled back

Version Migration ...

- Time heavily depends on disk and channel model used also
- Catalog size similar between V7 and V8 Compatibility Mode

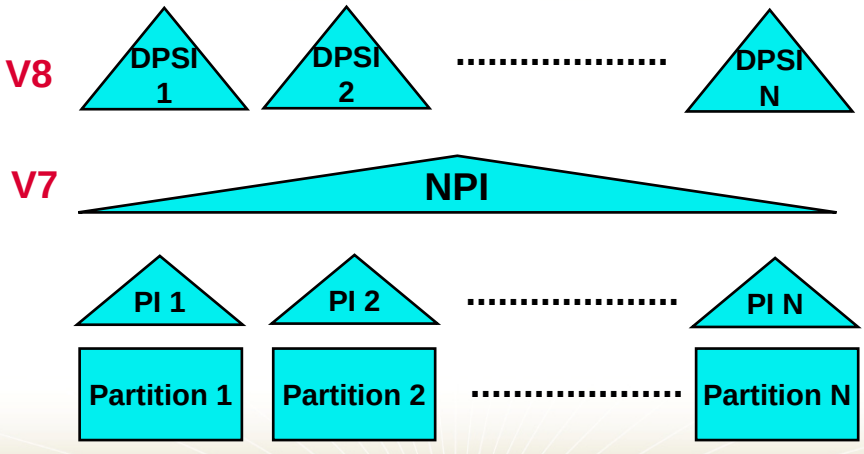
Catalog Migration

- V8 Compatibility to New Function Mode
  - Currently 0.1 to 2 hours observed, depending on the size of catalog/directory (medium to large), in non data sharing
  - New Function Mode
    - DB2 catalog in unicode
    - No fallback allowed
  - 1 to 10% increase in size of catalog observed for both data and index

Catalog Migration

- Online Reorg Sharelevel Reference of SPT01 and 17 catalog tables the most time-consuming component
- Very Rough Rule-of-Thumb on estimating the time for medium to large catalog/directory in non data sharing = 6 min + 3 to 7 min/GB of SPT01, SYSPACKAGE, SYSDBASE, etc.
  - Example: If 10 GB SPT01, SYSPACKAGE, SYSDBASE, then (6 + 5x10) = roughly 1 hour
  - Most catalogs are smaller than 10GB and thus faster migration possible
- Time heavily dependent on disk and channel model used also

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 ...

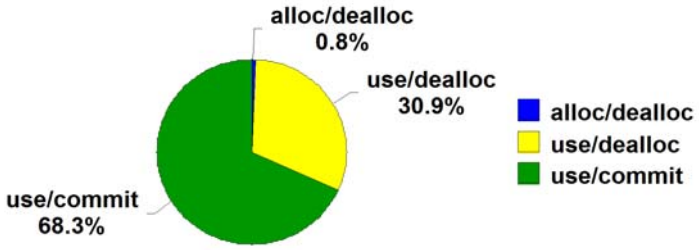
- REFRESH TABLE deletes current data and then inserts new data. MQT is locked out during this process.
- System-maintained MQT can be used just like any other table except for some restrictions
  - No Insert, Update, or Delete allowed
- User-maintained MQT supports both REFRESH TABLE and Insert, Update,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

- 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

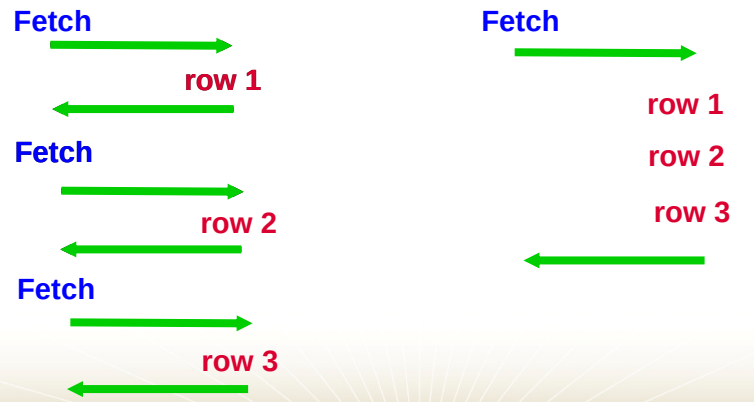
- Stage 1 and indexable predicate in
  - V6: Column comp-op non column expression such as SELECT FROM A WHERE a1=x+y
    - also char/varchar of different size in equi-join such as SELECT FROM A,B WHERE 10byte char a1=20byte varchar b1
  - V7: Column comp-op column expression in join such as SELECT FROM 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

- V7 PQ58420 3/2002 IS NOT NULL predicate made indexable
  - by searching for index entries < null
  - efficient when most entries are null

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)

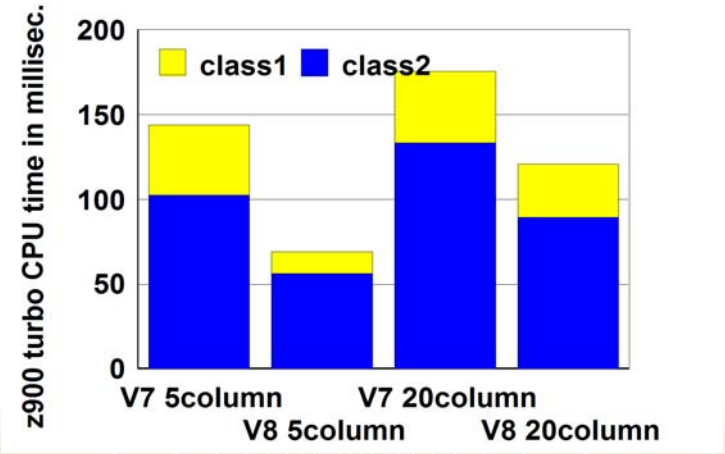
Distributed multi-row ...

- 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



Miscellaneous Performance Considerations

Miscellaneous Performance

- Index performance
  - Variable length index keys
  - Reading index backward
  - Others
- Online schema evolution
- Greater than 4K CI
- Long-term page fix option by buffer pool
- Data sharing performance enhancement

Variable length index keys

- VARCHAR index key no longer needs to be padded to maximum
  - V7: Always padded to maximum length
  - V8: Option of either padded or not
  - Especially useful for a large VARCHAR, e.g. DB2 catalog with 128 byte VARCHARs
  - In such a case, more index entries per index page, fewer index pages and index levels, less DASD space and buffer pool needed
- Further enablement of index-only access
  - SELECT varchar1 FROM table WHERE char1=x
  - with index on char1.varchar1

Misc. performance ...

- DEFIXPD zparm with default of
  - PADDED in migration to V8
  - NOT PADDED in new V8 install
- Maximum key length increased to 2000 from 255
  - Partition key is limited to 255
- CHAR(8) or VARCHAR(18) columns in catalog changed to VARCHAR(128)
- Preliminary Rule-of-Thumb: If less than 18 byte varchar columns, use padded key, because of
  - Extra CPU time for non-padded key processing
  - 2 extra bytes per varchar column in each non-padded key

Reading Index Backward

- Read multiple rows via index backwards to avoid sort
  - SELECT FROM TABLE ... ORDER BY c1 DESCENDING
    - with an ascending index on c1
    - Dynamic prefetch of index to make backward scan as efficient as forward scan
  - Supported with or without scrollable cursor
  - Accesstype = IR

Misc. performance ...

- Other index-related enhancements
  - Partitioned tablespace without index
    - ☞ Useful when PI created just for partitioning purpose and not for predicates
    - ☞ CPU and I/O reduction in Insert, Delete, Update
  - When no index is defined as clustering, the first created index is made clustering, making queries which reference this index potentially more efficient.
    - ☞ Compatible with Insert behavior
  - Clustering index separate from partitioning index

Online Schema Evolution (Alter)

- Highly available ALTER
  - Instead of Drop / Create of Table, Table space, and Index
- When some ALTER completes,
  - No existing data converted to new version format
  - Object placed in Advisory Reorg Pending (AREO) state
    - With some performance degradation
    - Shown in Display Database

Online Schema Evolution - continued

- ALTER TABLE ALTER COLUMN char to varchar, integer to decimal, char(8) to char(10)
  - 10 to 30% CPU time increase, depending on the number of columns processed, in Fetch because the fast column processing is not enabled prior to Reorg and possible conversion in each Fetch
  - CPU time increase brought down to +0 to 5% after Reorg (Alter varchar to char can result in -5% CPU)
  - Change allowed for longer length, precision, scale

Online Schema Evolution - continued

- ALTER INDEX
  1. CREATE INDEX PADDED, REORG, RUNSTATS
  2. SELECT using padded index <base case>
  3. ALTER INDEX NOT PADDED, REBUILD INDEX, RUNSTATS
  4. SELECT using not padded index
     - Can be faster or slower than base case
  5. ALTER PADDED, REBUILD INDEX, SELECT
     - No difference from the base

☞ No cumulative performance overhead

Index performance ...

- NOT PADDED index performance heavily dependent on the number and size of varchar columns in index key
  - As the difference between maximum and average varchar gets bigger, NOT PADDED index becomes better
    - for example, varchar(128) with an average of 8 bytes
- Alter Index between PADDED and NOT PADDED places index in RBDP (Rebuild Pending) state if at least one varchar column in index key
  - Index can not be accessed until it is rebuilt from the table

Greater than 4K CI Support

- New CI size equals page size by default
  - e.g. 16K CI for 16K page
- Enables VSAM I/O striping for 8K, 16K, 32K page
- Higher data rate for 8K, 16K, and 32K page
  - 16K page measurement with 16K instead of 4K CI
    - +36% for non EF (Extended Format) datasets
    - +70% for EF datasets
    - EF getting nearly equivalent to non EF in data rate performance

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
- 8% overall CPU time reduction for IRWW transaction observed
  - 0 to 10% 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

References

- V8 manuals
  - Especially Performance Monitoring and Tuning section of Administration Guide
- Redbooks at www.redbooks.ibm.com
  - DB2 UDB for z/OS V8 Performance SG24-6465, soon
  - DB2 UDB for z/OS V8 Everything you ever wanted to know ... SG24-6079
  - DB2 UDB for z/OS V8 Technical Review SG24-6871
- More DB2 UDB for z/OS information www.ibm.com/software/db2zos