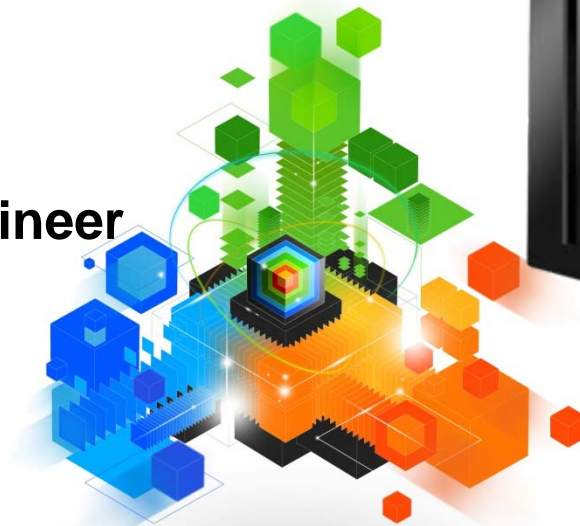


Gain insight into DB2 9 and DB2 10 for z/OS performance updates and save costs

John B. Tobler
DB2 for z/OS Senior Software Engineer
jtobler@us.ibm.com



Disclaimer:

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The Information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance Disclaimer:

This document contains performance information based on measurements done in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the numbers stated here.

DB2 10 Performance Preview

- **Abstract**

This session offers a look at performance impact of DB2 9 and DB2 10 for z/OS with particular emphasis on the DB2 10 improvements.

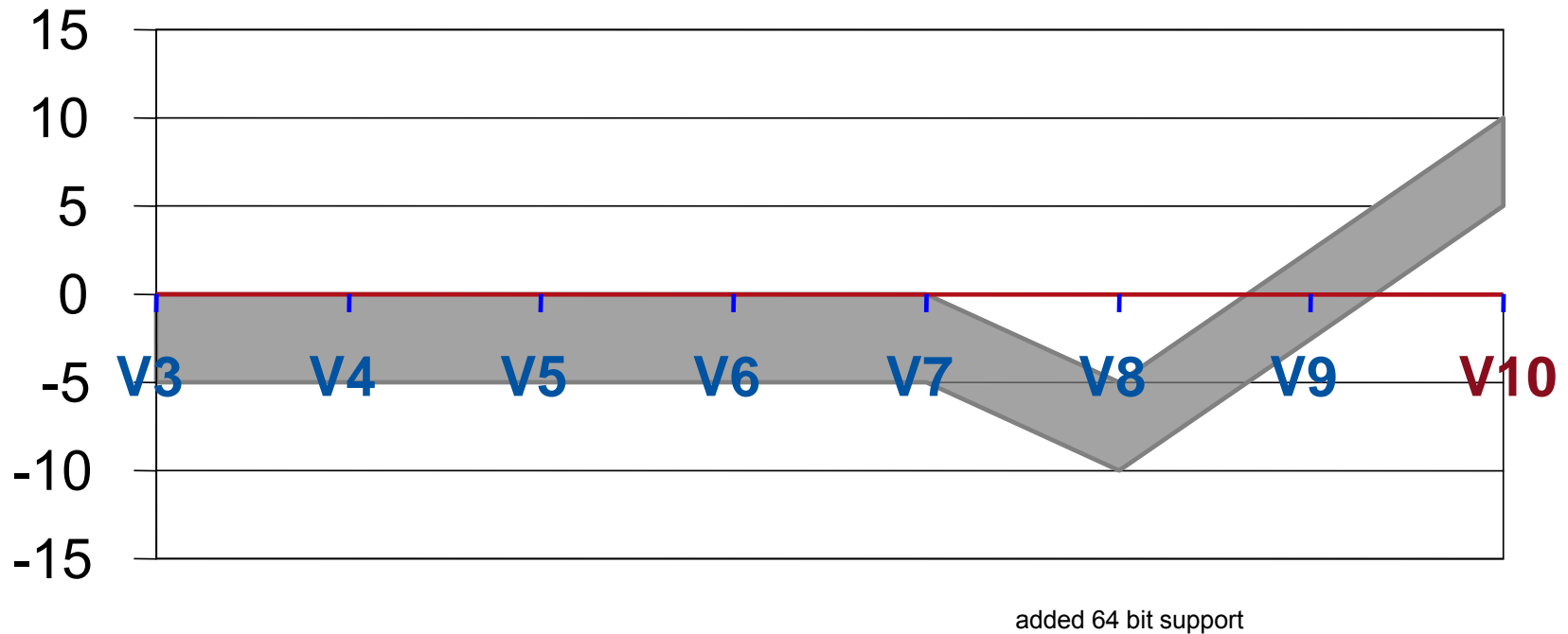
- **Agenda**

- DB2 10 for z/OS performance goals and expectations
- Scalability and buffer pool enhancements
- INSERT improvement
- FETCH/SELECT improvement
- LOB, XML, and SQL procedure performance
- JDBC and DDF performance

DB2 10 Performance Objective

Historical goal of <5% version-to-version performance regression
Goal of 5% -10% performance improvement for DB2 10

Average %CPU improvements
version to version



DB2 10 Performance Expectation

Most of workloads...

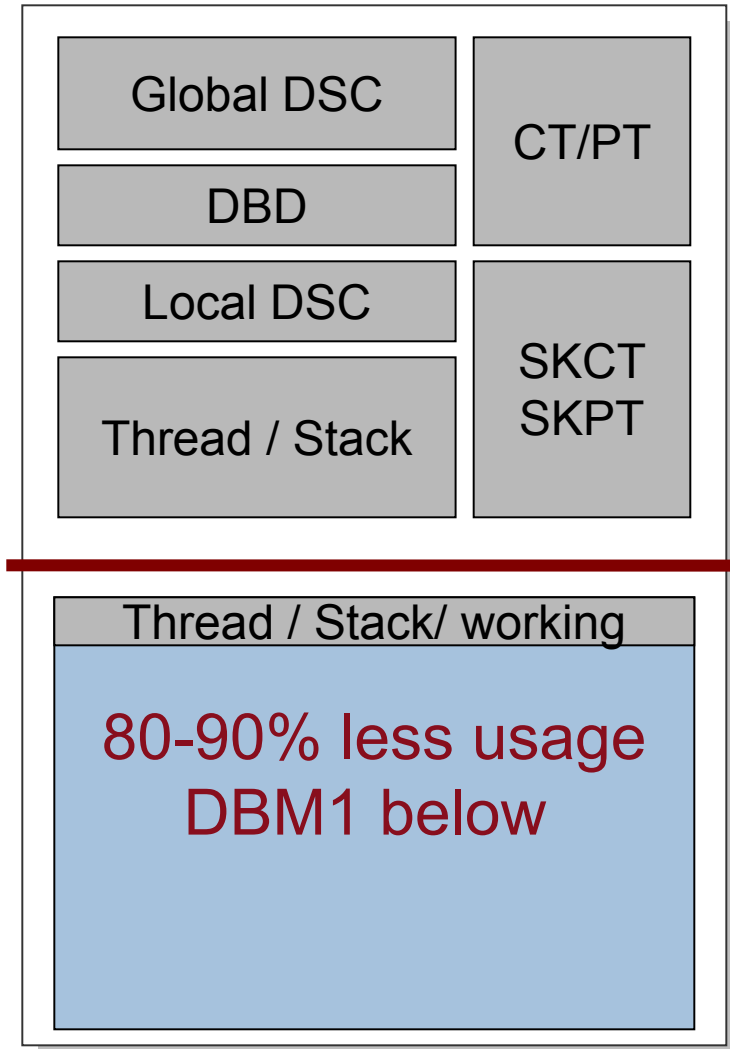
- Up to 10% CPU reduction after REBIND packages
- Higher improvement with workload with scalability issues in V8/V9 or accessed thru DRDA

Sweet Spots...

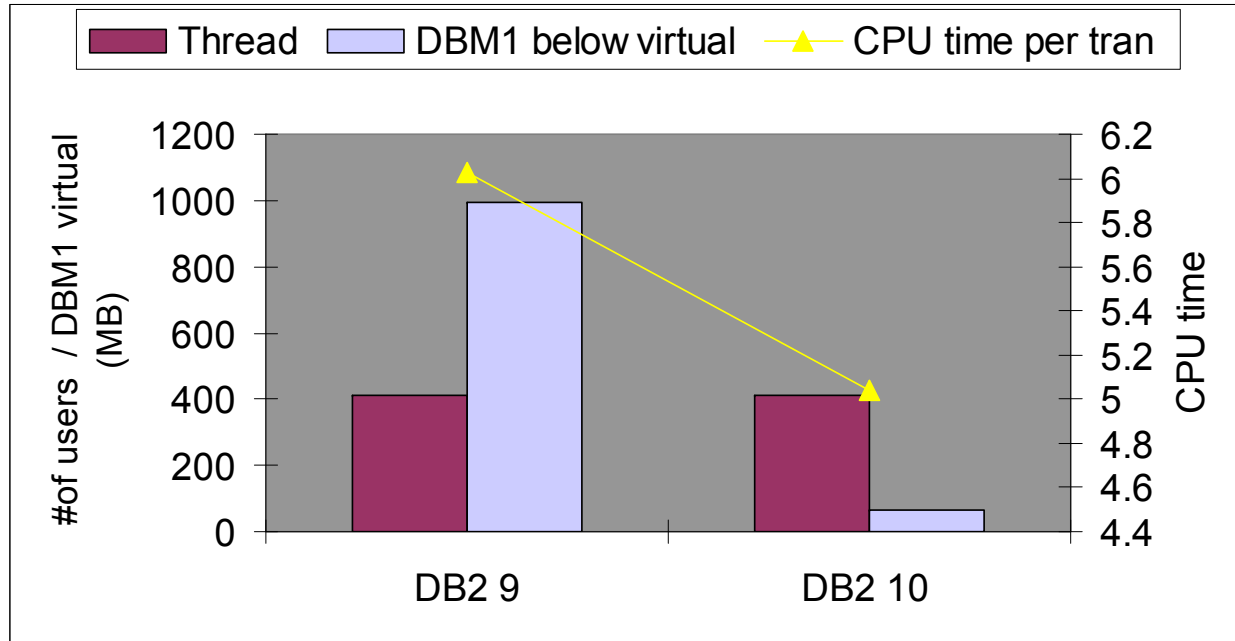
- Workload using native SQL procedures: up to 20% CPU reduction after DROP/CREATE or REGENERATE the procedures
- Query workload with positive access path changes
- Workload with frequent access on small LOB (NFM with Inline LOB)
- Workload with random, singleton select/update (NFM with Hash access)

DBM1 Virtual Storage Constraint Relief V10

- **DBM1 below 2GB**
 - 80-90% less usage in V10 compared to V9
 - Some of working storage (stack, xproc storage) stays below 2GB
- **Larger number of threads**
 - Possible data sharing member consolidation
- **Improve CPU with storage**
 - More release deallocate
 - Larger MAXKEEPD values for KEEP DYNAMIC=YES



Virtual Storage Reduction from SAP Workload



- **412 concurrent threads**
- **Virtual storage below the bar**
 - 997 MB with DB2 9
 - 63 MB in DB2 10
- **No significant increase in real storage**

DBM1 VSCR Monitoring

- **More focus on**

- Real storage usage (PM24723)
- Common storage (ECSA and ESQA) usage

- **New statistics in IFCID 225 reports**

- DBM1 and DIST address space: virtual below and above, real, and aux
- Common and Shared storage usage (z/OS APAR OA33106 SRB ESQA reduction)

DBM1 AND MVS STORAGE BELOW 2 GB	QUANTITY
-----	-----
TOTAL NUMBER OF ACTIVE USER THREADS	2694.28
NUMBER OF ALLIED THREADS	386.00
NUMBER OF ACTIVE DBATS	2275.06
NUMBER OF POOLED DBATS	33.21
REAL AND AUXILIARY STORAGE FOR DBM1	QUANTITY
-----	-----
REAL STORAGE IN USE (MB)	5396.07
31 BIT IN USE (MB)	289.45
64 BIT IN USE (MB)	5106.62
HWM 64 BIT REAL STORAGE IN USE (MB)	5106.64

Performance Scalability - DB2 Latches (CM)

Most of DB2 latches from 64 cp scalability evaluation will have a relief

- LC12 : Global Transaction ID serialization
- LC14 : Buffer Manager serialization
- LC19 : Log write in both data sharing and non data sharing
- LC24 : EDM thread storage serialization (Latch 24)
- LC24 : Buffer Manager serialization (Latch 56)
- LC25 : EDM hash serialization
- LC27 : WLM serialization latch for stored proc/UDF
- LC32 : Storage Manager serialization
- IRLM : IRLM hash contention
- CML : z/OS Cross Memory Local suspend lock
- UTSERIAL : Utility serialization lock for SYSLGRNG (NFM)

Performance Scalability - H/W synergy

Exploitation of z10 features

- CPU improvement using z10 prefetch instructions
- Large fixed page frames for buffer pool
 - Buffer pools with PGFIX=YES
 - Define IEASYSxx LFAREA 1MB page frames
 - Reduction of hit miss in TLB (translation lookaside buffer)
- Observed 1-4% CPU reduction

In memory buffer pool with large real

- DB2 managed in memory buffer pool
 - PGSTEAL = NONE
 - Pre-load the data at the first open or at ALTER BPOOL
 - Avoid unnecessary prefetch request
 - Avoid LRU maintenance → no LRU latch (LC14)

INSERT Performance Improvement

DB2 9

- Large index pages
- Asymmetric index split
- Data sharing Log latch contention and LRSN spin loop reduction
- More index look aside
- Support APPEND option
- RTS LASTUSED support

DB2 10 CM

- Space search improvement
- Index I/O parallelism
- Log latch contention reduction and faster commit process
- Additional index look aside

DB2 10 NFM

- INCLUDE index
- Support Member Cluster in UTS
- Complete LRSN spin avoidance

Universal Table Space (UTS) – Member Cluster (NFM)

- **Member Cluster option in create table space**

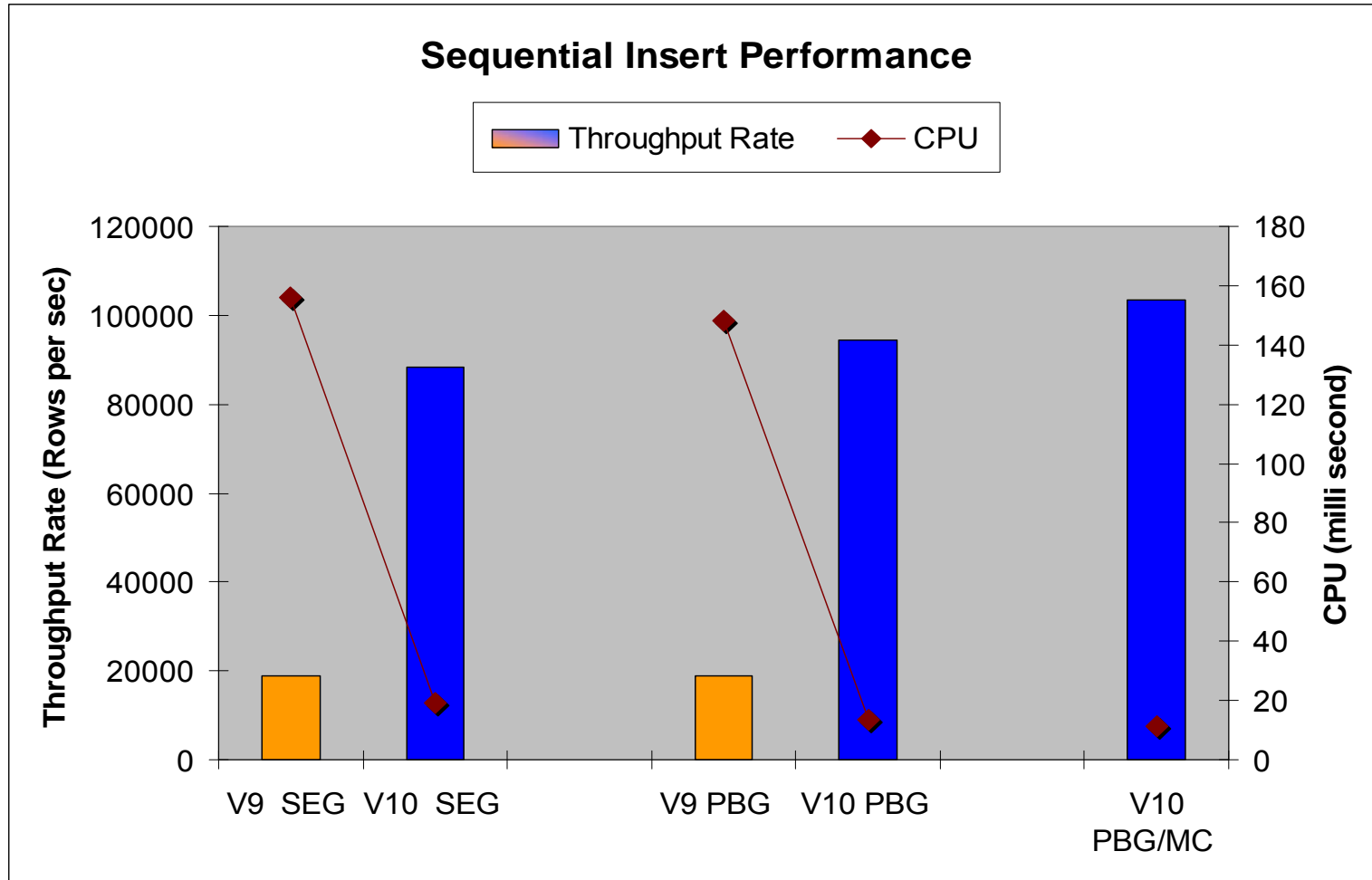
- Assigns a set of pages and associated space map page to each member
- Remove the “hot spots” in concurrent sequential insert in data sharing
- It does not maintain data cluster during the INSERT
- Data cluster needs to be restored via REORG
- Each space map contains 10 segments

- **Altering to MEMBER CLUSTER**

```
ALTER TABLESPACE MyTableSp  
MEMBER CLUSTER YES/NO;
```

- REORG to materialize the pending alter

INSERT Performance Improvement



Sequential key insert into 3 tables from JDBC 240 clients in two way data sharing members. Using Multi Row Insert (batch size 100). Each member resides on LPARs with z10 8CPs.

I/O Parallelism for Index Updates (CM)

V9 During insert, DB2 executes index updates sequentially.
Tables with many non-clustering indexes may suffer high synchronous read I/O wait

V10 I/O parallelism by prefetching index pages to overlap the I/Os against non-clustering indexes

- Still one processing task. No improvement if all indexes are in the buffer pools
- Effective to reduce I/O wait for large indexes which cannot fit in the buffer pools.
- New zparm `INDEX_IO_PARALLELISM` with default YES
- Classic Partitioned TS and UTS (both PBG/PBR) but not for segmented TS

Additional Non-key Columns in a Unique Index (NFM)

V9 Multiple indexes per table

An index is used to enforce uniqueness constraint. Additional indexes are necessary to achieve index only access on columns not part of the unique constraint during queries.

Higher Insert / Delete CPU time, increased storage requirements

V10 Additional Non-key Columns in an unique indexes

Reduce index maintenance cost during insert, DASD space savings

Additional Non-key Columns in a Unique Index

- **V9 definition**

```
CREATE UNIQUE INDEX i1 ON t1(c1,c2,c3)
```

```
CREATE INDEX i2 ON t1(c1,c2,c3,c4,c5)
```

- **Possible V10 definition**

```
CREATE UNIQUE INDEX i1 ON t1(c1,c2,c3) INCLUDE (c4,c5)
```

or

```
ALTER INDEX i1 ADD INCLUDE (c4,c5) and DROP INDEX i2
```

- **The following restrictions will apply:**

- INCLUDE columns are not allowed in non-unique indexes
- Indexes on Expression will not support INCLUDE columns
- Indexes with INCLUDEd columns can not have additional unique columns
ALTER ADDED to the index

SELECT/FETCH Performance Improvement

V9

Sort performance improvement, in memory workfile/Sparse index
-Index on Expression
-Many access path related improvements
Plan Stability for static SQL statements
Histogram stats, etc.

V10

CPU reduction on index predicate evaluation
Better performance using a disorganized index
Row Level Sequential Detection
Group by using Hash, More in memory workfile usage
Dynamic statement cache support for literal constants
Many access path related enhancements
-Plan stability for both static and dynamic statements
-Parallelism improvement
-IN list access improvement
-Auto stats...and more

CPU reduction in Predicate Evaluation (CM)

- **Optimize in index predicate evaluation process**

- Applicable in any workload but query with many predicates shows higher improvement

- **Performance improvement**

- Average improvement shows average 20% CPU reduction from generic 150 queries.
- Individual queries show between 1 and 70% improvement

Improvement in using Disorganized Index (CM)

- **Index scan using disorganized index causes high sync I/O wait**
- **Disorganized index detection at execution**
- **Use List Prefetch on index leaf pages with range scan**
 - Reduce Synchronous I/O waits for queries accessing disorganized indexes.
 - Reduce the need of REORG Index
 - Throughput improvement in Reorg, Runstats, Check Index
 - Limited to forward index scan
- **Performance results**
 - Observed 2 to 6 times faster with simple SQL statements with small key size using list prefetch compared to Sync I/Os

Row Level Sequential Detection (CM)

- **Problem:**

Dynamic prefetch sequential works poorly when the number of rows per page is large.

- **Solution:**

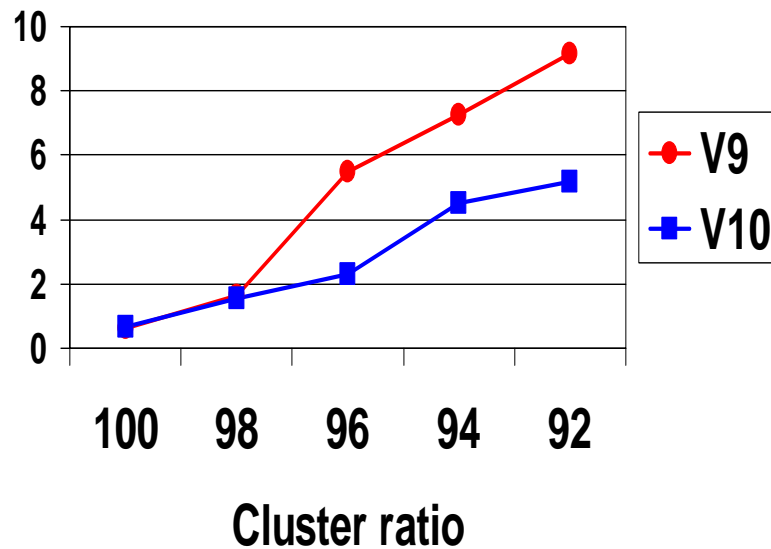
Row Level Sequential Detection (RLSD).

- Count rows, not pages to track the sequential detection.
- Since DB2 10 will trigger prefetch more quickly, it will use progressive prefetch quantity.
 - For example, with 4K pages the first prefetch I/O reads 8 pages, then 16 pages, then all subsequent I/Os will prefetch 32 pages (as today).
 - Also applies to indexes

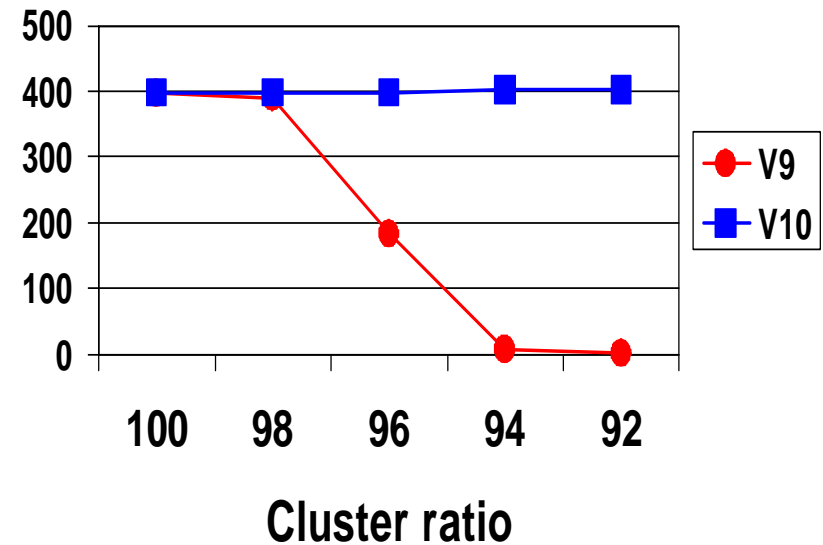
Index → Data Range Scan

Row size = 49 bytes, page size = 4K (81 rows per page)
Read 10% of the rows in key sequential order

Query Time (seconds)

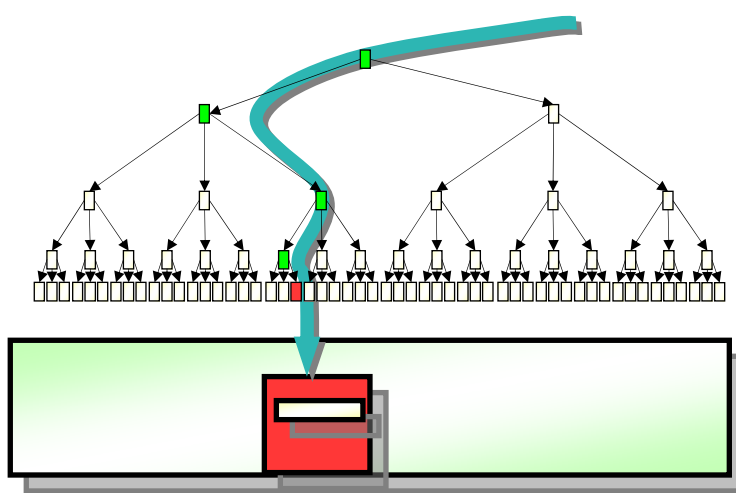


Dynamic Prefetch I/Os



- Row level sequential detection (RLSD) preserves good sequential performance for the clustered pages

Index to Data Access Path vs. Hash Access



- Index->Data access
 - Traverse down Index Tree
 - For a 5 Level Index
 - 6 GETP
 - 2 I/O's
 - 5 index page searches

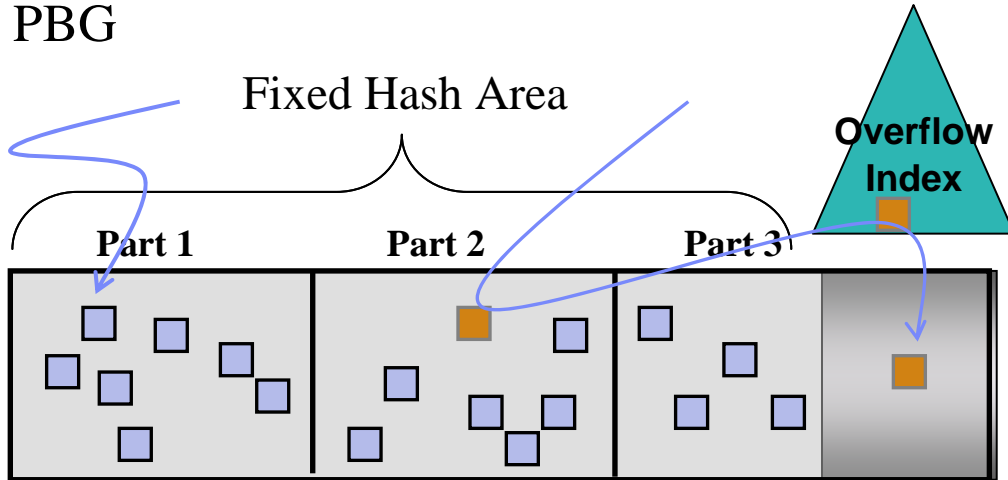
■ = Page in Bufferpool
■ = Page Read from Disk



- Hash Access
 - Locate a row without having to use an index
 - Single GETP in most cases
 - 1 Synch I/O in common case
 - Greatly reduced Search CPU expense

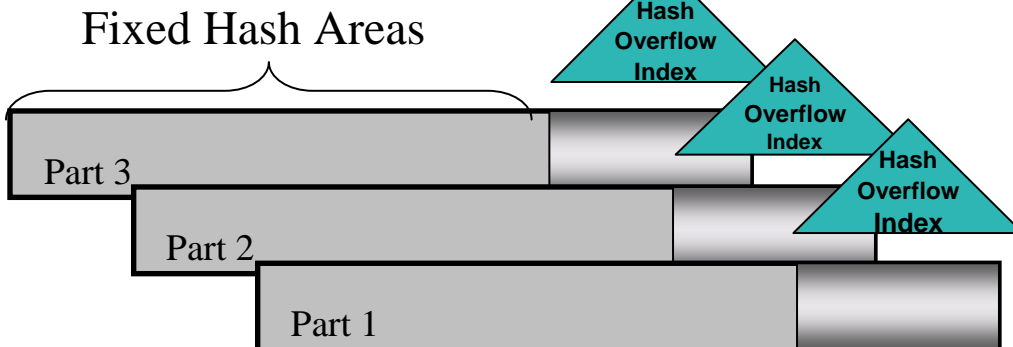
Hash Access and Hash Space

PBG



- Optimal to get from fixed area
 - 1 getpage, 1 I/O
- Overflow
 - 3 getpages, 2-3 I/Os
- Use REORG with **AUTOESTSPACE YES** unless you know better
- Real Time Statistics (RTS)

PBR



- # of overflow
TOTALENTRIES
- **TOTALENTRIES / TOTALROWS < 10%**
- FREEPAGE is not valid for HASH space but PCTFREE is honored

Hash Access Summary

▪ Performance benefit :

- Up to 30% DB2 CPU reduction with random access
 - Higher improvement with large table with small rows
 - Savings in index maintenance once you remove the clustering index
- Possible reduction in Hotspots
 - Rows are randomly distributed

▪ Performance concern :

- Not for sequential fetch nor insert
 - Significant Sync I/O increase if accessed in clustering order
 - No Member Cluster support
 - Careful research is necessary on picking the candidate
 - Statement level of monitoring for GetPage and I/Os
- Significant impact on LOAD utility using input data with clustering order
 - Relief is coming soon
- Possible INCREASE in I/O or BP space in some cases
 - In case of small 'active' working set
 - In case of many "row not found"

SQL Procedure Performance (CM)

v9

Introduced native SQL Procedure

Improvement by executing procedures in DBM1 instead of WLM address space

V10

Native SQL Procedures

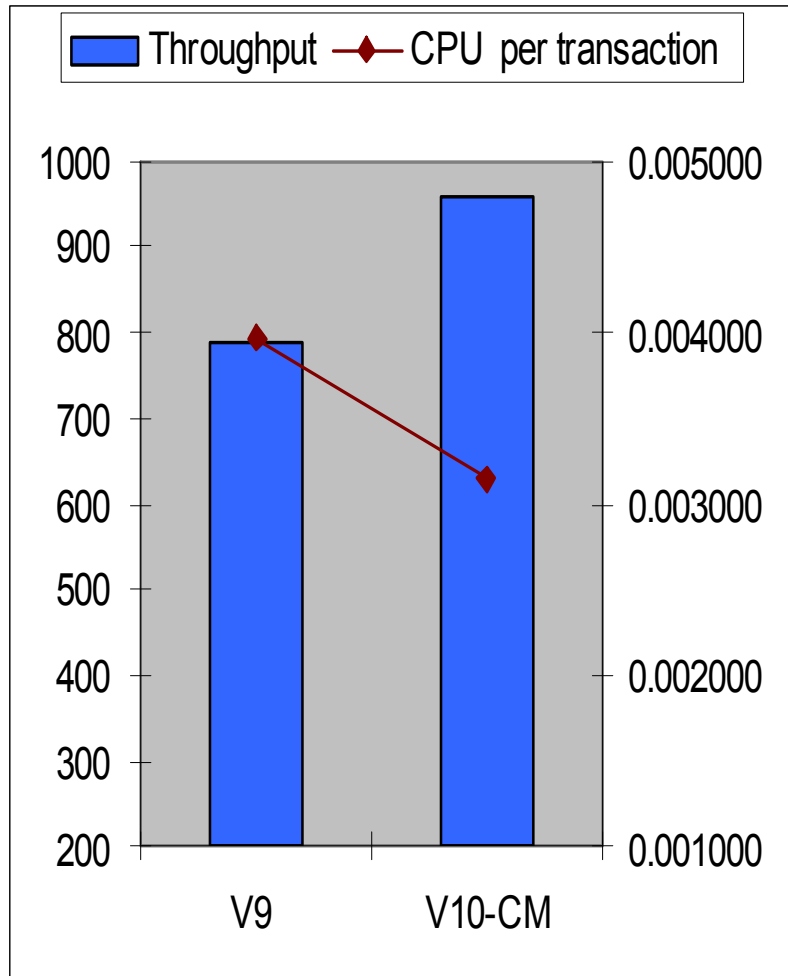
Further performance optimization

Specific CPU reduction in commonly used areas

-Pathlength reduction in IF statement

-Optimization in SELECT x from SYSDUMMY1

Measurements – SQLPL (CM)



- **OLTP using SQLPL**
 - 20% CPU reduction with V10 CM
 - 89% DBM1 Below the Bar usage reduction
 - 5% resp time improvement due to latch contention relief

Local JDBC and ODBC Application Performance

- **Local Java and ODBC applications did not always perform faster compared to the same application called remotely**
 - DDF optimized processing with DBM1 that was not available to local ODBC and JDBC application.
 - zIIP offload significantly reduced chargeable CP consumption
- **Open support of DDF optimization in DBM1 to local JCC type 2 and ODBC z/OS driver**
 - Limited block fetch
 - LOB progressive streaming
 - Implicit CLOSE
- **Expect significant performance improvement for applications with**
 - Queries that return more than 1 row
 - Queries that return LOBs

High Performance DBATs

- **Re-introducing RELEASE(DEALLOCATE) in distributed packages**
 - Could not break in to do DDL, BIND
 - V6 PQ63185 to disable RELEASE(DEALLOACTE) on DRDA DBATs
- **High Performance DBATs reduce CPU consumption by**
 - RELEASE(DEALLOCATE) to avoid repeated package allocation/deallocation
 - Avoids processing to go inactive and then back to active
 - Bigger CPU reduction for short transactions
- **Using High Performance DBATs**
 - Stay active if there is at least one RELEASE(DEALLOCATE) package exists
 - Connections will turn inactive after 200 times (not changeable) to free up DBAT
 - Normal idle thread time-out detection will be applied to these DBATs
 - Good match with JCC packages
 - Not for KEEP DYNAMIC YES users

High Performance DBAT...

- **New -MODIFY DDF PKGREL command**

- Options

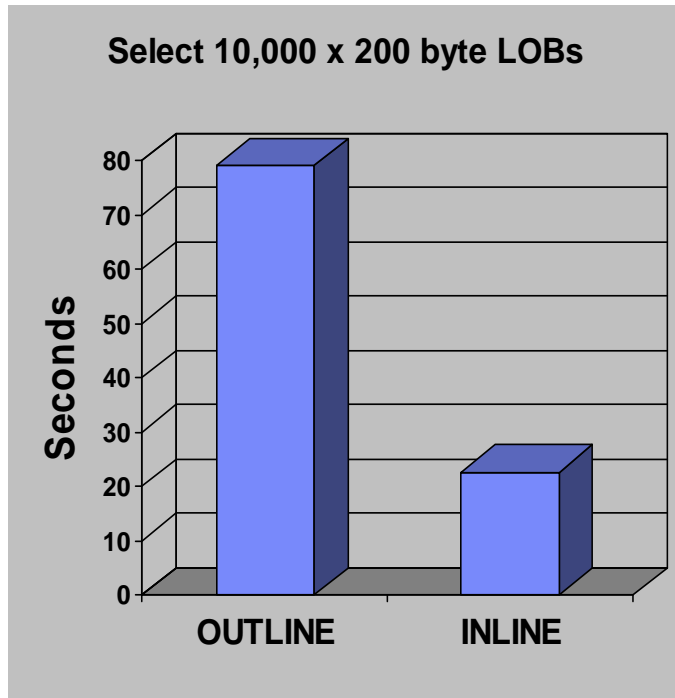
- PKGREL(BNDOPT) honors package bind option
- PKGREL(COMMIT) forces package bind option
RELEASE(COMMIT)
 - Same as V9 inactive connection behavior
 - Will allow BIND and DDL to run concurrently with distributed work
- PKGREL(DEALLOC) forces package bind option
RELEASE(DEALLOCATE)
 - Provides better performance behavior
 - BIND and DDL can not break in when concurrent distributed work runs

Inline LOBs (NFM)

- **CREATE or ALTER TABLE INLINE LENGTH on UTS**
 - INLINE to base table up to 32K bytes
- **Completely Inline LOBs**
 - Reduce DASD space
 - No more one LOB per page, Compression
 - CPU and I/O saving
 - Avoid LOB aux indexes overhead
- **Split LOBs**
 - A part of LOB resides in base and other part in LOB TS
 - Incur the cost of both inline and out of line
 - Index on expression can be used for INLINE portion

Inline LOBs

Elapsed time in random select



Very small LOBs select, insert shows
Up to 70% elapsed time reduction
with INLINE LOBs

- Inline is good, if
 - Most of LOBs are small and only a few large ones
 - Compress well
- Inline is not good, if
 - Most of LOBs become “split LOB” unless indexing is important for inlined portion
 - Majority of SQLs do not touch the LOB columns
- Base table becomes larger with Inline
 - Buffer hit ratio for base table may decrease
 - Image copy of base table becomes larger

XML Performance Improvement

- Significant Performance improvement in V9 service stream
- DB2 10 performance improvement
 - Binary XML support
 - Avoid the cost of XML parsing during insert
 - Reduce the XML size
 - Measured 10-30% CPU and elapsed time improvement
 - Schema Validation in engine
 - No more UDF call for validation
 - Utilize XML System Service Parser
 - 100% zIIP / zAAP eligible for validation parser cost
 - XML Update
 - No more full document replace

DB2 10 Monitoring Enhancements and Changes

- 1. New Monitor class 29 for statement detail level monitoring**
IFCID 316/318 for dynamic, 400/401 for static
- 2. Record index split with new IFCID 359**
- 3. Separate accounting to identify DB2 latch and transaction lock in class3**
- 4. Package LASTUSED**
- 5. Storage statistics(IFCID225) for DIST address space, shared, and common storage**
- 6. Specialty Engines**
Possible redirection value (zIIP SECP) is no longer supported, always zero
SE CPU (actual offloaded CPU time) continues to be available

Portion of RUNSTATS utility (redirect rate depends on RUNSTATS parms)

Prefetch and Deferred Write Engines redirected 100%

DB2 10 Monitoring Enhancements and Changes

7. **Package accounting information with rollup**
8. **Statistics trace interval**
Always 1 minute interval in V10 no matter what you use in STATIME for critical statistics records
9. **Compression for DB2 trace data in SMF with a new zparm (SMFCOMP)**
Overhead is minimum (up to 1% measured)

Up to 90% SMF data set saving from measurements Trace formatter needs to be modified to call z/OS services to decompress the data

Beta Customers' Feedback – Workload level

Workload	Results
CICS online transactions	Approx. 7% CPU reduction in DB2 10 CM after REBIND, 4% additional reduction when 50MB of 1MB page frames are used for selective buffer pools
CICS online transactions	Approx 12% CPU reduction
CICS online transactions	Approx 5% CPU reduction from DB2 8
CICS online transactions	No CPU reduction - Candidate of release deallocate usage
Distributed Concurrent Insert	50% DB2 elapsed time reduction, 15% chargeable CPU reduction after enabling high perf DBAT
Data sharing heavy concurrent insert	38% CPU reduction
Queries	Average CPU reduction 28% from V8 to DB2 10 NFM
Batch	Overall 20-25% CPU reduction after rebind packages

Beta Customers' Feedback – Line Item Focused

Workload	Results
Multi row insert	33% CPU reduction from V9, 4x improvement from V8 due to LRSN spin reduction
Query with 10 stage 1 predicates	5 index matching, 1 index screening, range and IN predicates 60% CPU reduction with same access path
Parallel Index Update	30-40% Elapsed time improvement with class 2 CPU time reduction
Inline LOB	SELECT LOB shows 80% CPU reduction
Include Index	17% CPU reduction in insert after using INCLUDE INDEX
Hash Access	20-30% CPU reduction in random access No improvement or some degradation in CICS workload 16% CPU reduction comparing Hash Access and Index-data access. 5% CPU reduction comparing Hash against Index only access 20x elapsed time increase in sequential access

Thank you !

John Tobler (jtobler@us.ibm.com)