Z29

Introduction to DB2 for z/OS and OS/390 Capacity Planning for Basic SQL

Akira Shibamiya

IBM Data Management Technical Conference

Anaheim, CA Sept 9 - 13, 2002

NOTES

- Abstract: A simple pencil-and-paper technique of estimating CPU, I/O, and elapsed time of various SQL calls for DB2 for z/OS and OS/390 V5, V6, and V7, based on Rule-of-Thumb numbers, is covered in this presentation.

- Speaker: Akira Shibamiya, IBM Silicon Valley Laboratory.

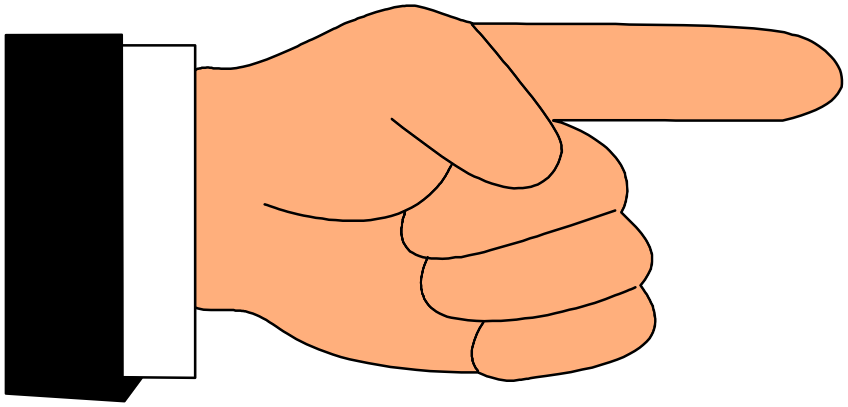
Acknowledgment and Disclaimer

- 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

- Capacity planning objectives and alternatives
- "Quick" estimation of CPU time, I/O time, and elapsed time of SQL calls using Rule-of-Thumb numbers
- What if analysis - table check/referential integrity constraint, trigger, distributed environment, stored procedure, UDF, many columns or host variables, trace options, LOB, DB2 data compression

Capacity Planning



NOTES

- Capacity planning here is defined as the process of estimating CPU and I/O time required for a given request
  - in a relatively well-tuned application, database, and hardware/software configuration environment
- CPU and I/O time estimations here are intended for V5, V6, and V7 of DB2 for OS/390 and z/OS.
  - Any version-dependent information will be identified wherever appropriate.

Capacity Planning Objectives

I don't have a DB2 yet but want to know a ballpark figure of how much CPU, I/O, and elapsed time are expected for a set of frequently executed queries or transactions, so that I can plan for a required hardware configuration.

We are thinking of adding referential integrity or table check constraints or triggers but what would be the performance impact?

Many additional "What If" questions

NOTES

- **Primarily, there are two types of capacity planning objectives:**

    1. **No DB2 subsystem is available yet, or no DB2 measurement data is available, but want to get a ballpark performance estimate for <u>hardware/software planning or application/database design</u>**

    2. **DB2 measurement data is available and want to know if an observed performance is reasonable, or if there is a potential for a significant performance improvement via tuning**

- **This page is for the first type of objective.**

Capacity Planning Objectives - continued

My SELECT SQL takes 10 minutes
of G6 CMOS CPU time. Is this reasonable?

We are only getting 10 transactions per second.
For our transactions with the number and
type of SQL calls given, is this expected?

☛ **Intelligent DB2 Performance Design and Tuning**

NOTES

- **This page is for the second type of capacity planning objective:**

  That is, DB2 measurement data is available and want to know if an observed performance is reasonable, or if there is a potential for a significant performance improvement via tuning

Capacity Planning Alternatives

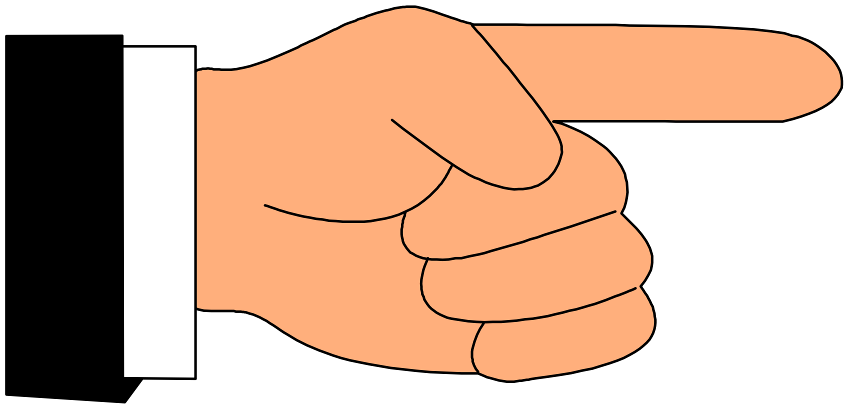
- The following four alternatives are listed in an increasing order of cost <u>and</u> accuracy:
  1. Use of measurement results reported in published articles or hearsay
  2. Quick Pencil-and-Paper Analysis
  3. Use of capacity planning models such as DB2 Estimator (www.ibm.com/db2) and SQL/PA (www.redbooks.ibm.com DB2 for z/OS and OS/390 Tools SG24-6139, SG24-6508)
  4. Workload-specific benchmark measurement

NOTES

- **The simplest and fastest technique in estimating performance is to rely on available measurement report for some other workload.**
  - **But an accuracy of such estimation can be in serious doubt as the performance critically depends on the specific workload used.**
- **Potentially most precise but most expensive is the benchmark measurement using a specific workload, carefully duplicating the real environment as much as possible.**
- **This presentation covers an intermediate approach of quick pencil-and-paper analysis.**

Pencil-and-Paper Capacity Planning of SQL Calls

- **Quick estimation of CPU time, I/O time, elapsed time via Rule-of-Thumb numbers**



NOTES

OUTLINE

- **Rough SQL CPU time for online transaction**
- **Rough SQL CPU time for query**
- **Rough I/O time estimation**
- **Rough elapsed time estimation**

Rough SQL CPU Time for Online Transaction

- Select, Insert, Update, Delete = 55 to 165us
  - + 40 to 80us for each index entry insert or delete

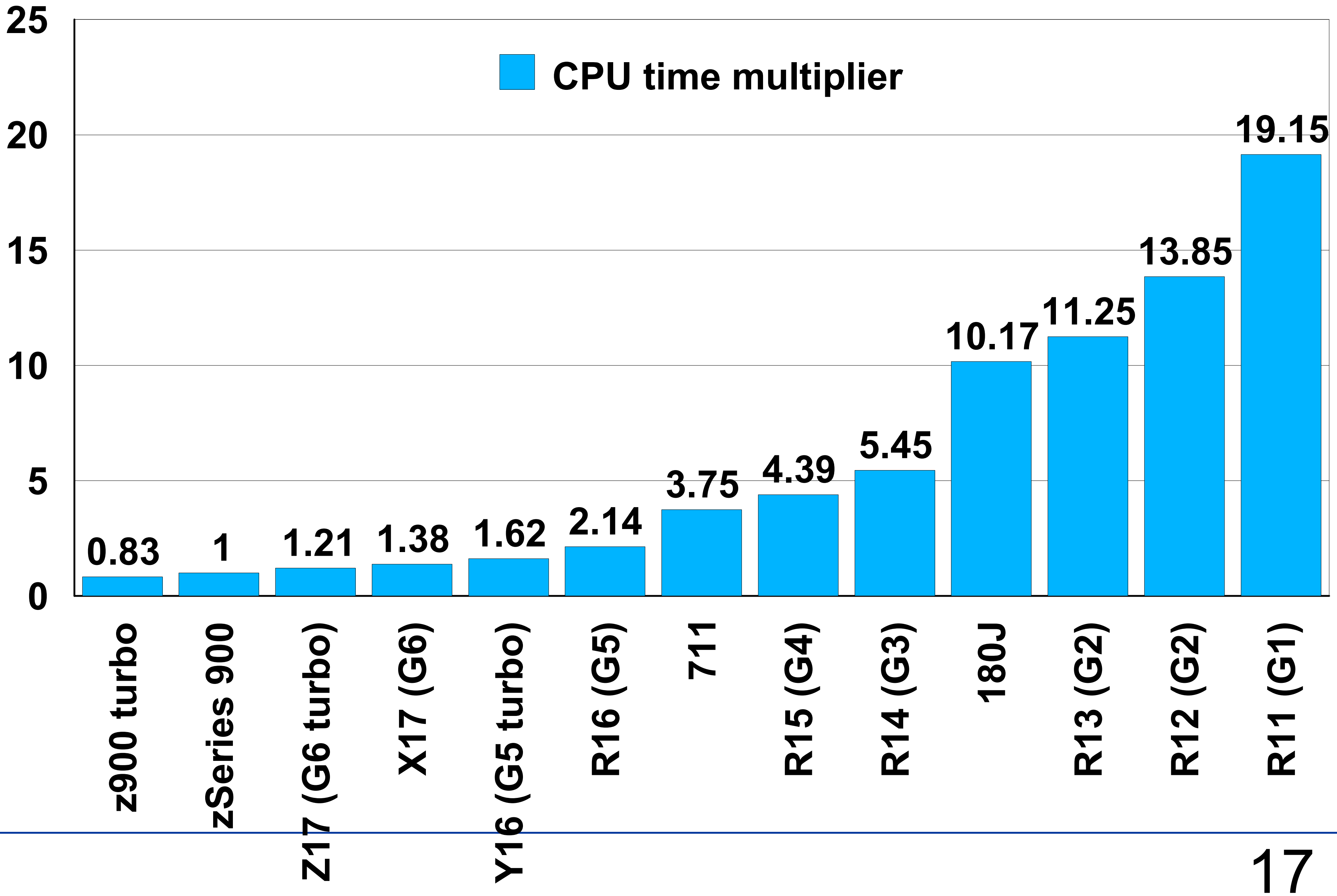
- Open/Fetch/Close = 80 to 200us
  + 22 to 44us for each additional Fetch
- Prepare
  - 40 to 80us if hit in global dynamic statement cache (CACHEDYNAMIC YES in Install panel)
  - more than 2ms if no hit in dynamic statement cache (can be much higher for complex SQL statement)

- Add to each SQL call 22us if CICS attach (can be as low as 0 if DB2 V6 or V7 and CICS/TS 2.2), 4us if class2 acctg

NOTES

- 'Typical' CPU time expressed in us (microseconds) of zSeries 900 processor (z900), including CPU time for I/O unless otherwise specified
- Intended for simple reasonably-tuned online transaction workload
  - Matching index access to one or very small number of rows in one table (use query CPU time if more than one table)
  - <500byte row and <30 columns or host variables processed
  - No DB2 trace other than basic statistics and accounting
- Range of numbers
  - Lower number for fewer predicates, columns, host variables, concurrent activities, and contention. Also for repeated execution in a loop, smaller tables and indexes with fewer I/O's.
  - If not sure, an average number can be used

CPU Time multiplier for various processor models



NOTES

- zSeries 900 processor CPU time in microseconds is used throughout this presentation, unless otherwise specified.
  - For other processor models, adjust with desired ITR (Internal Throughput Rate) in www.s390.ibm.com/lspr, or DB2 for OS/390 Capacity Planning redbook SG24-2244, or use the CPU time multiplier shown here.
    - For example, 110us SELECT on zSeries 900 processor would take 110us*1.21=133us on Z17 (G6 turbo or G6T) processor.
    - Please note these multipliers represent AVERAGE as they depend on the type of workload.

Online Transaction CPU Time

- Read-only commit = 45 to 90us
- Update commit = 160 to 280us
- Create/Terminate Thread = 250 to 500us
  - or thread reuse and release deallocate = 80us signon
- Distributed Create/Terminate Thread = 2000 to 4000us
  - or V6 inactive thread = 300 to 600us

NOTES

- **Range of numbers depending on the number of distinct SQL statements executed, bind option resource release at commit or deallocate, complexity of SQL calls executed, etc.**

Online Transaction Example

IRWW transaction	z900 CPU time
Signon	80us
4.89 Select * 110	538us
2.85 Insert * [110 + 60 for 1 index]	485us
3.69 Update * 110	406us
0.21 Delete * [110 + 60 for 1 index]	36us
4.55 Open * 140	637us
9.25-4.55 Additional Fetch * 33	155us
0.54 read-only commit * 68	37us
DB2 accounting class 2 = 30 calls * 4	120us
Total accounting TCB time, estimated	2.5ms
" measured	2.7ms

NOTES

- Accounting TCB time of IRWW transaction measured on zSeries 900 processor with DB2 V7 and OS/390 2.10
  - IRWW (IBM Relational Warehouse Workload) described in ITSO Redbook "DB2 for MVS/ESA V4 Data Sharing Performance Topics" SG24-4611
  - "Average" CPU time used in estimate calculation, i.e. (minimum+maximum)/2
- For this average transaction, 54% are read-only and 46% are update.
  - Update_commit of 0.46*220us = 101us/transaction is charged to MSTR SRB time.

Online Transaction Example - continued

- **Internal Throughput Rate (ITR) estimation**
  - **If 20% in other DB2 time including IRLM, DBM1, and MSTR TCB and SRB time and 1ms in application CPU time and no other significant workload,**

    then the maximum possible transaction rate can be calculated as follows:
    - **average trans CPU time = 2.5ms*1.2 + 1ms = 4ms**
    - **ITR = 1sec/(4ms/trans) = 250trans/sec/processor**

NOTES

- ITR = Internal Throughput Rate = Calculated throughput at 100% CPU utilization
  - used for comparison purpose by converting ETR (External Throughput Rate) at the measured CPU utilization which can be different for each measurement

24

Rough SQL CPU Time for Query

- "Query" here is intended to cover retrieval SQL calls processing many rows in contrast to one or few rows in online transaction

- 0.3 to 1.7us for each row scan
  - Range of numbers depending on number of predicates evaluated, number and type of columns processed, row size, index scan or tablespace scan, etc.

- + 5us for each row sort

- + 13us for each page scan, including CPU time for prefetch I/O.
  - If no I/O, use 6us instead.
  - If random I/O, use 39us instead.

SQL CPU Time for Query - continued

- + 17 to 23us for each fetch
- + 15us for each table
- Add for each Fetch SQL call
  - 22us if CICS attach (can be as low as 0 if DB2 V6 or V7 and CICS/TS 2.2)
  - 4us if class 2 accounting

Ballpark I/O time per page

	Sequential Read or Write		Random Read	
	4K page	32K page	4K page	32K page
3390, Ramac1, Ramac2	1.6 to 2ms	14ms	20ms	30ms
Ramac3, RVA2	0.6 to 0.9ms	6ms	20ms	30ms
ESS E20/ESCON	0.3 to 0.4ms	3ms	10ms	15ms
ESS F20/ESCON	0.25 to 0.35ms	2ms	10ms	15ms
ESS F20/FICON	0.13 to 0.2ms	1.5ms	10ms	15ms

NOTES

- For 8K and 16K page, interpolate from 4K and 32K page numbers.
- For skip sequential read or write, i.e. reading or writing of pages which are not contiguous, the time would be somewhere between sequential and random and depends on the distance between pages read or written.
- I/O time for sequential read or write is prorated on a per page basis, since multiple pages can be read or written by one Start I/O.
- Read I/O time tends to be faster than write I/O; eg use 1.6to1.8ms seq read and 1.8to2ms seq write for 3390.
- Random read I/O time would go down to 0.6 to 2ms range if cache hit.

Rough Elapsed Time (ET) Estimation

- If synchronous I/O (read), ET = SUM(CPU, IO1, IO2, ...)
- If asynchronous I/O (write or prefetch read),
  ET = MAX(CPU, IO1, IO2, ...)

Example of SELECT call scanning 20M rows via 200K index pages and 1M data pages by sequential prefetch and returning one row, using zSeries 900 processor and ESS E20 DASD

- ►CPU time = 20M*1us + 1.2M*13us = 36sec
- ►IO1 time for index = 0.2M*0.3ms = 60sec
- ►IO2 time for data = 1M*0.3ms = 300sec
- ►ET = MAX(36, 60, 300) = 300sec

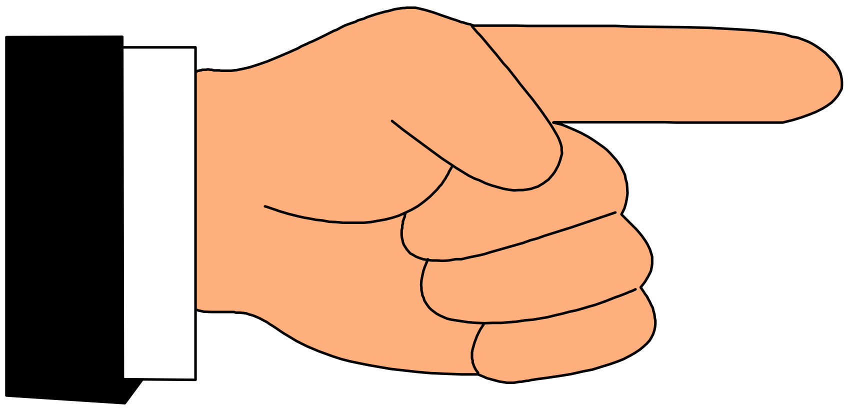
NOTES

- CPU time from "Rough SQL CPU time for Query" on page 25 = 0.3 to 1.7/row + 13/page

- If degree any, ET reduction of up to N times for N degrees possible for both CPU-bound and I/O-bound query if sufficient resource available

What If Analysis

- More estimation of CPU time and I/O time for different cases



NOTES

OUTLINE

- Check constraint
- Trigger
- Distributed environment
- Stored procedure
- UDF
- Many columns or host variables
- Trace options
- LOB
- DB2 data compression

What if Check Constraint?

- Referential integrity check

Additional CPU time
= 30us for each row updated/deleted/inserted
for each index to be checked
+ CPU time for index I/O if any

- Table check constraint

Additional CPU time
= 4us for each row updated/deleted/inserted
for each constraint
+ 4us for each insert/update/delete SQL
statement with constraint

NOTES

- **Referential integrity check and Table check constraints can be specified via CREATE or ALTER TABLE.**
  - **Please see SQL Reference Manual for details.**

- **CPU time for I/O in Appendix**

- **Additional CPU time for Insert, Update, or Delete is shown**
  - **Assume referential integrity check via index-only access**
  - **Cascade Delete is not included**
  - <u>**Note no SQL Application Program Interface overhead here**</u>

What if Trigger? (V6)

- Trigger by Insert, Update, or Delete

Trigger invocation CPU time = 45us base

+ 40us for each package allocation (once per trigger executed in transaction)

+ 14us for each trigger invocation

+ __us for trigger itself

NOTES

- Trigger defined by CREATE TRIGGER.
  - Please see SQL Reference Manual for details.

- Trigger by V7 Online Load Resume and Referential Integrity check also

- Trigger invocation
  - once per statement execution if statement trigger
  - once per affected row if row trigger

- No transition variable nor transition table assumed
  - Higher CPU time and possibly work file I/O time if REFERENCING clause specified.

What if Trigger - continued

- Example of many Insert SQL calls in a loop with 2 indexes on a table
  - ►Insert overhead = 55us + NIX*60us = 175us
    - Minimum % trigger overhead = 14/175 = 8%
  - ►Insert SQL triggering Update
    - CPU time = 175 + 14 + 55 - 20 = 224us
    - If Insert SQL followed by Update SQL, then CPU time = 175+55 = 230us
    - Thus, % trigger overhead here is -3% compared to equivalent function without using trigger

NOTES

- Insert and Update Rule-of-Thumb from page 15
  = 55to165us + 40to80us for each index updated

- The cost of Insert/Update/Delete SQL executed in a trigger can be estimated by

  - estimated SQL cost - 20us for avoiding Application Program Interface

What if Distributed Environment?

- DRDA CPU time overhead

  - ► Non block fetch = 210us for each SQL call
  - ► Block Fetch = 5to10us for each Fetch SQL call
    + 80us for each message

+ 300 to 600us for V6 inactive thread scheduling per transaction (2000 to 4000us if Create/Terminate Thread)

NOTES

- For each SQL call in DRDA non block fetch, add
  - 28 to 56us + 170us message send/receive = 210us
- Block Fetch enabled if
  - Read-only query
  - or Current Data NO and ambiguous cursor (i.e. dynamic SQL present)

What if Stored Procedure?

Stored procedure invocation CPU time
= 220 to 560us
+ 170us for each message send/receive

- ► **0 to 2 message Send/Receive in stored procedure**
  - **0 if local**
  - **1 if COMMIT ON RETURN with WLM-managed stored procedure (default = No commit on return)**
  - **2 else**

NOTES

- **Assume STAY RESIDENT YES to avoid stored procedure reloading**
  - **default=NO**

- **Range of numbers depending on**
  - **Number and size of input and output parameters**
  - **Language used**
  - **PROGRAM TYPE of SUB instead of MAIN to reduce stored procedure invocation overhead**

- **Replace SQL statements in some SET assignment to C code in SQL/PSM procedure V6/V7 PQ55247 and 55446 3/02**

Stored Procedure Example

- Stored Procedure can reduce CPU usage and response time in a distributed environment
  - ► Example of 10 Select, Insert, Update, and/or Delete SQL calls in stored procedure
    - • Additional CPU time without stored procedure
      = 10calls*210us
      = 2100us
    - • Additional CPU time with stored procedure
      = about 600us
    - • Also faster response time because of as low as 1 rather than 10 message send/receive

NOTES

What if UDF? (V6)

Sourced UDF CPU time based on built-in function

- ➡ 3.5us each if V5
- ➡ 3us each if V6 or V7

► **If SELECT YEAR(   ), ...**

- **+3/110 for simple SELECT = +3%**

NOTES

- UDF (User Defined Function) created by CREATE FUNCTION
  - Sourced
  - External (Scalar or Table)
  - V7 SQL
- Sourced UDF based on internal DB2 built-in function has equivalent performance to built-in function
  - CHAR, DATE, DAY, DECIMAL, DIGITS, FLOAT, HEX, HOUR, INTEGER, MONTH, STRIP, SUBSTR, YEAR, ...
- Typically 1 to 10% more CPU time for V7 SQL Scalar UDF over built-in function

What if UDF - continued

External UDF CPU time

= 220 to 560us stored proc invocation
+ 85us for each UDF invocation
+ __us for UDF itself

- Example of many SELECT UDF1,... loop in a local environment
  - UDF invocation cpu time = 85/110 simple SELECT or 77% but can be much less for more complex SELECT SQL

NOTES

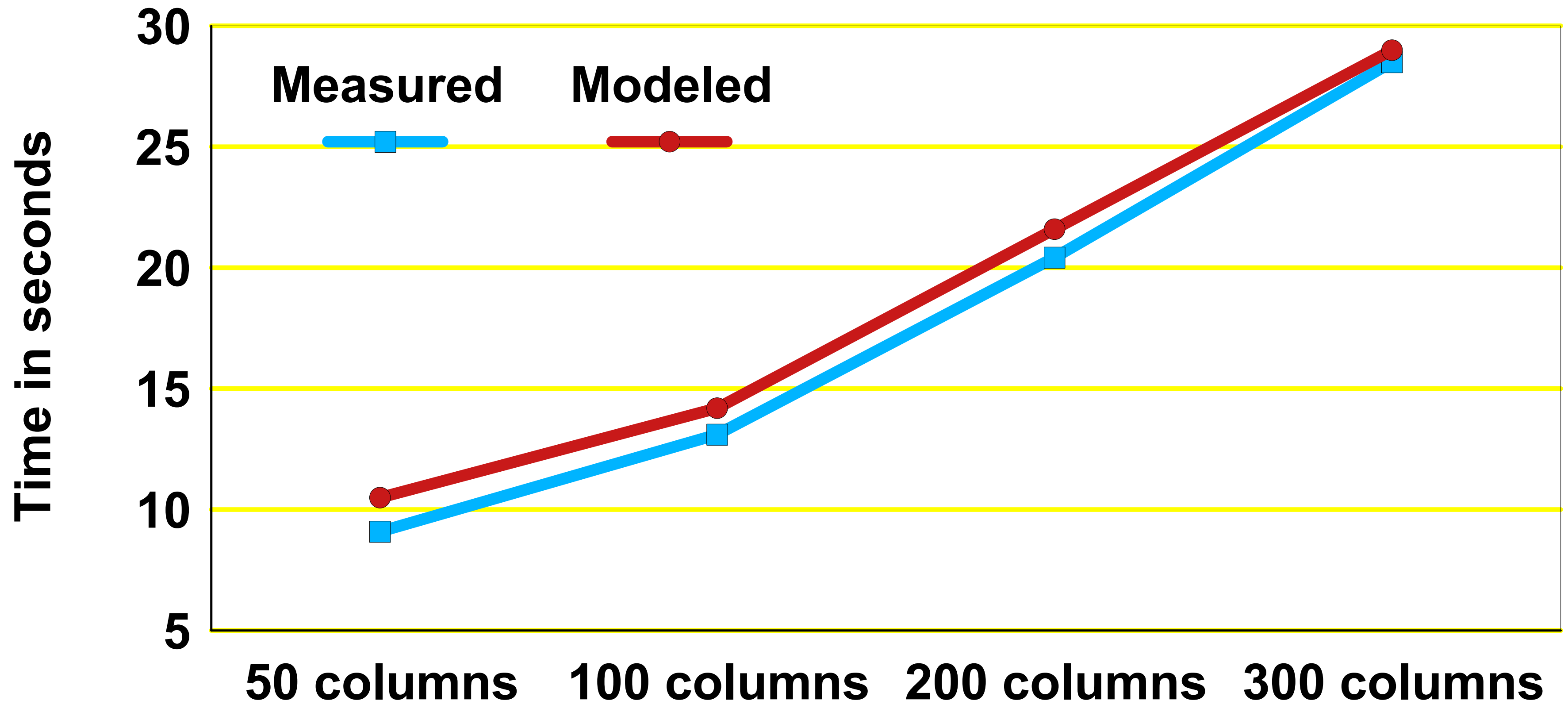
- **Performance tuning recommendations made for stored procedure would apply here**

- **External scalar, not table, function modeled here**
  - **Join predicate between UDF and base table made indexable if table UDF is accessed first (V7 PQ54042 12/01)**

- **The true cost of UDF must be evaluated by comparing with what it would take to perform the same function without UDF.**

What if Many Columns/Host Variables?

- If many column retrieval (e.g. >30),
  - 0.15 to 0.25us for each column fetched
  - Example of 340K row fetch G6T CPU time = 1.21*0.34M*[22 + 0.18*(#columns-30)]



NOTES

- Fetch CPU time Rule-of-Thumb = 22 to 44us on page 15
- Comparison between the measurement and the model

	Measured	Modeled
50 column Fetch	9.1sec	10.5sec
100 column Fetch	13.1	14.2
200 column Fetch	20.4	21.6
300 column Fetch	28.5	29.0

Many Columns/Host Variables - continued

- If many host variables (e.g. >30),
  - ►0.2 to 0.3us each
- If DRDA,
  - ►2 * (0.15 to 0.25us) for each column fetched
- If ASCII or Unicode instead of EBCDIC but without conversion,
  - ►1.5 to 2 * (0.15 to 0.25us) for each column fetched
  - ►1.5 to 2 * (0.2 to 0.3us) for each host variable
- If ASCII-EBCDIC or other single byte CCSID conversion,
  - ►1.5 to 3us each

NOTES

- Try to minimize number of columns and host variables processed to reduce CPU usage

- For numeric data, more efficient to use numeric column type

  - Less space required
  - More precise filter factor estimation in range predicate
  - Minimize conversion overhead

52

What if Trace Options?

- Statistics and Accounting trace
  - <5% overhead typical, except for Fetch-intensive application
    - accounting class 2 overhead of 4us per SQL call can add 11 to 18% for 22 to 44us simple Fetch ROT (Rule-of-Thumb)
- Audit trace
  - 7us for each audited table access or update first time in a transaction
    - <5% tran overhead typical with all audit classes on
- Performance trace
  - 6us OPn, 11us GTF, 25us SMF destination for each trace record

NOTES

- Watch out for performance trace with many trace records, e.g. SQL trace (performance trace class(3)) of Fetch-intensive application
  - 'In DB2' CPU time can double in this case as 2 trace records are produced for each SQL call.
- Accounting class 3 overhead is typically negligible
  - Exception: when more than 1000 DB2 internal latch contentions per second, especially class 19 log latch

What if LOB? (V6)

- Additional CPU time for each LOB with LOG NO

	CPU time in microseconds
Select into host variable	70 + 2*[KB of LOB]
Select into locator	80
Insert or Update	95 + 2*[KB of LOB]
Delete	95

  - Add CPU time for I/O as needed as in Appendix
  - For I/O time, refer to page 27 "Ballpark I/O Time Per Page"

NOTES

- LOB (Large OBject) for up to 2GB maximum column
- Select into locator is independent of LOB size. So is Delete because of pseudo-delete.
- Update of LOB results in delete and insert.
- Force write of updated pages at commit if LOG NO
  - Recommend DWT=0 for LOBs to promote continuous deferred writes
- Assume LOB size > 32KB as recommended
  - One exception to the recommendation: when LOB column is rarely referenced, a performance gain achievable for smaller LOBs by storing LOB column in LOB tablespace instead of all together in base table

Example of rarely referenced LOB

- Select 10 LOBs with average size of 30000 bytes via tablespace scan of base table containing 1M rows with 1000 byte each

Base table with varchar

1M rows, each on 32K page = 32GB

Base table with LOB tablespace

Base table of 1M rows on 250,000 4K page = 1GB

$\downarrow$

LOB tablespace of 1M rows, each on 32K page = 32GB

NOTES

- With varchar
  - Base table read = 0.13ms*8 * 1M 32K pages = 1040sec
- With LOB
  - Base table read = 0.13ms * 250000 4K page = 33sec
  - LOB tablespace read = (10ms * 10 4K index leaf page) + (15ms * 10 32K LOB data page) = 250ms
- So 30 times less I/O time with LOB in this example.

LOB - continued

- ROT (Rule-of-Thumb) on LOB compared to non LOB (varchar)
  - Good when >32KB average
    - Application program has to support >32KB logical record if LOB is not used
  - OK when >2KB average, especially if LOB is rarely referenced
- If a mix of small and large LOBs with an average of 4KB or less, use 4K page rather than 8K, 16K, or 32K page, as only 1 LOB can be stored per page
  - 4K page is also good for I/O striping

NOTES

- If a larger LOB, bigger page size can reduce CPU time for page processing
  - -25% CPU time in 30K LOB Insert on 32K page compared to 4K page
- LOB Load performance improvement via V7 PQ59820 5/02 to reduce LOB data write I/O time
  - -43% elapsed time in one measurement
- Use higher PQTY/SQTY as needed to avoid frequent dataset extends and reduce elapsed time

What if DB2 Data Compression?

- zSeries 900 CPU time in microseconds

  = 0.6 + rowsize*0.01 for each row compressed
  = 0.45 + rowsize*0.006/(1+2*CR) for each row decompressed

- G6 turbo CPU time in microseconds

  = 0.6 + rowsize*0.055 for each row compressed
  = 0.45 + rowsize*0.036/(1+2*CR) for each row decompressed

- CR = Compression Ratio

NOTES

- Generally, one processor MIPS ratio is used for all functions.
  - However, because there is a much larger difference in MIPS ratio for compression/decompression between zSeries 900 and G6 or older processors, the compression/decompression CPU time is shown separately. (up to 5 times difference instead of 1.15 to 1.3 times average)
- Rule-of-Thumb numbers intended for an average case in terms of rowsize, CR, and data content

3 Major Observations

1. z900 can be up to 5 times faster than G6T in both compression and decompression

2. Compression is significantly more expensive than decompression

3. Decompression cost is a function of CR (Compression Ratio)

   ► Faster decompression with higher CR

NOTES

- Rule-of-Thumb on when to avoid DB2 data compression
  - If CR < 10 to 20%
- DSN1COMP utility can be used to estimate CR of DB2 full image copy data sets, VSAM data sets, or DSN1COPY output containing DB2 tablespace

Compression Example

- Example of Insert subselect of 4M 142byte rows from uncompressed 142K page source to compressed 65K page target, no index access, no I/O
  - 142K page without and 65K page with compression, for a compression ratio of 54%
  - Estimated compression overhead on zSeries 900
    = 4M*(0.6+142*0.01) = 8sec (8sec measured)
  - Estimated compression overhead on G6 turbo
    = 4M*(0.6+142*0.055) = 34sec (33sec measured)

NOTES

CPU Time	G6 turbo	zSeries 900
With compression	115.42sec	69.90sec
Without compression	82.65sec	61.74sec
Difference	33sec	8sec

- Compared to G6 turbo, z900 is about 25% faster without compression but compression cost is about 75% less

Decompression Example

- Example of tablespace scan of 152M 154byte rows, no I/O (near-worst case in terms of %decompression overhead)
  - 5.8M page without and 4.6M page with compression, for a compression ratio CR of 21%
  - Estimated decompression overhead on zSeries 900
    = 152M*(0.45 + 154*0.006/(1+2*CR))
    = 167sec (155sec measured)
  - Estimated decompression overhead on G6 turbo
    = 152M*(0.45 + 154*0.036/(1+2*CR))
    = 662sec (634sec measured)

NOTES

CPU Time	G6 turbo	zSeries 900
With compression	941.76sec	385.38sec
Without compression	307.38sec	230.04sec
Difference	634sec	155sec

- Note less decompression cost with higher CR
- Also note estimates tend to be higher than the measurement because the page processing overhead for fewer number of pages due to compression is not accounted here for the sake of simplicity.

Reference

- Redbooks at www.redbooks.ibm.com
  - DB2 for z/OS Squeezing the Most Out of Dynamic SQL SG24-6418
  - DB2 for z/OS and OS/390 V7 Selected Performance Topics REDP0162 (Redpaper)
  - DB2 for z/OS and OS/390 Tools SG24-6139, SG24-6508
  - DB2 for z/OS and OS/390 V7 Performance Topics SG24-6129
  - DB2 UDB Server for OS/390 V6 Technical Update SG24-6108
  - DB2 UDB for OS/390 V6 Performance Topics SG24-5351
  - DB2 for OS/390 Capacity Planning SG24-2244

Reference - continued

- DB2 UDB for OS/390 Administration Guide, Performance Monitoring and Tuning Section, SC26-9003 for V6, SC26-9931 for V7
- More information on DB2 UDB for OS/390, including DB2 Estimator, at www.ibm.com/db2
- More on Insert capacity planning in Insert Performance Considerations in DB2 for OS/390 presentation, 2000 to 2001

Appendix - CPU time for I/O

- **CPU time for synchronous read I/O**

  = 33us if 4KB page, 56us if 32KB page

- **CPU time for asynchronous prefetch read or deferred write I/O**

  = 6.7us/page if 32 4KB page I/O