

May 6-10, 2007

San Jose Convention Center

San Jose, California, USA

How to be a Hero by Taking Small Steps

IDUG® 2007

North America

Steen Rasmussen
CA

Platform: <DB2 for z/OS>



GoFurther



Abstract

This presentation will describe the performance impact experienced by making small changes to both the application as well as the physical design in a real life scenario. Issues like UR and lock avoidance will be covered and the considerations for choosing UR.

Referential Integrity is used in many applications, but is it really necessary and what is the performance impact.

Dynamic statements is becoming more and more "popular" - but should they be coded using constants and literals - or ? Another topic which can improve performance is compression - but what is the impact and what about the old "myths". Finally the necessity of reorganizations is being discussed and how to avoid some of these reorganizations.

Disclaimer

- The information provided in this presentation is based on experiences from a specific scenario, so the results may vary in any other environment or application.

Agenda

- Performance complaints – where to start
- Analysis of executed SQL
- Referential Integrity analysis
- Locking analysis
- Dynamic SQL analysis
- Compression analysis
- Reorg avoidance / considerations
- Wrap-up and Steen's philosophy

Scenario outline

- Someone complained about performance !
 - Where to start
 - The complaint was related to a huge DB2 CPU consumption – this helped a lot and eliminated a lot of investigation (network, server, DB2, application code, DRDA,.....)
 - Helps a lot if you know the application – I had no clue
 - Can you justify if the SQL statements are correct and appropriate ?
 - I quickly found out the developers knew the programming language – but not much about DB2 (we were a *great team* since I had no clue about the programming environment Java, JDBC etc.)
 - Start to ask questions
 - Collect and consolidate information
 - Create an identical environment to **test changes** in order not to impact the application unless needed to improve performance

Collect info about Objects and Attributes

- Only 8 tables involved in this application with a load of indexes
- One table had 17 indexes where a few were redundant

- IX1 : COL1
- IX2 : COL1 , COL3
- IX3 : COL1 , COL3 , COL4







Unless needed for Primary Key, these are **overhead** for Insert, Delete, Update and Utilities

- A lot of FK's found (Foreign Key) which could explain a lot of the indexes found
(this is where the “ask questions” task kicks in)
- Beside the many indexes – nothing “unusual”

Collect information about executed SQL

- Some challenges:
 - Static SQL can be located in the catalog and explained
 - This was ALL dynamic SQL, so I had to capture all the statements executed within a 24 hour period to make sure a full production day was covered.
- Important tuning issue/priority
 - Focus on a SQL statement which executes in **5 minutes** or on the statement which executes in **sub-seconds** ?
 - The answer is the “famous DB2 answer” – IT DEPENDS
 - We will need to look at execution frequency – where can we get “the biggest bang for the bucks”

Findings based on information collected

- The most frequently executed statement was one of the cheapest - an INSERT
 - Average Insert : 0.003924 CPU-sec.
 - It was executed THOUSANDS of times, so the time spent added up 
 - The table inserted into had a number of Foreign Keys 
- The Insert statement used LITERALS/CONSTANTS 
- GETP activity showed Catalog tables were hit a lot 
- The most expensive statement per execution was a SELECT statement – and a very ugly one (in my opinion)
- The above identified issues were sufficient to get started and kick off “OPERATION HERO”

Agenda

- Performance complaints – where to start
- **Analysis of executed SQL**
- Referential Integrity analysis
- Locking analysis
- Dynamic SQL analysis
- Compression analysis
- Reorg avoidance / considerations
- Wrap-up and Steen's philosophy

Investigate the “ugly statement”

- Most of the catalog GETP requests appeared to come from the following SELECT statement.
- It was time to chat with the Developers – remember this is dynamic SQL in a JDBC environment
 - Remember ALWAYS to talk to the developers and train them
 - Rapid development is a major requirement in order to support the business – so Dynamic SQL is about to become “the standard”
 - Not trivial to explain everything before the application is live – often we don’t even know something changed (no bind has to pass the Change Management process)
 - **A SQL statement might look all right, but we NEED to question the necessity of the statement really being executed (we might even need to know the business – scary!!) - How many statements do you have which actually don’t need to be executed ?**

Investigate the “ugly statement”

- This one turned out to be “an idea they were playing with” but the “idea” was never implemented.
- This statement was executed on top of the hour 6 times.....
- This is a “simple” JDBC statement generated by z/OS DB2’s implementation of the JDBC getColumnns method – looks VERY harmless ?

```
dbProdName = dbmd.getDatabaseProductName();  
dbProdVer = dbmd.getDatabaseProductVersion();  
ResultSet rs = dbmd.getColumnns(null,null,"NUMERICFACT","NFMONTH");
```

[IDUG NA2007 alt.doc](#)

(Hyperlink to the SQL statement generated – statement is not in handout due to the size)

Agenda

- Performance complaints – where to start
- Analysis of executed SQL
- **Referential Integrity analysis**
- Locking analysis
- Dynamic SQL analysis
- Compression analysis
- Reorg avoidance / considerations
- Wrap-up and Steen's philosophy

Investigate Referential Integrity

- What was the reason behind the decision to look into the Referential Integrity in place ?
 - The application was doing mass inserts
 - Most of the inserts were insertions into child tables
 - Some of the biggest CPU consuming statements were these Inserts
 - Remember - DB2 has to check the parent for the child table insertion every time in order not to end up with orphans
- So prior to going crazy, let's look at RI

Investigate Referential Integrity

- Why use system enforced RI instead of Application defined RI ?
 - Good method to document relations
 - Guarantees integrity
 - No need to code programs and periodically execute these to check integrity (let DB2 do the hard work)
 - Clean-up / Delete processing greatly simplified
- Some challenges
 - RI complicates the “DBA life”
 - Utilities can be a pain (Load, Check, Recover)
- Don't forget – in general it is cheaper to have DB2 guarantee the integrity than having the application doing it !!

Investigate Referential Integrity

- Time to ask questions again
 - Is the application written to always make sure the “parent” also gets inserted ?
 - Does the application do delete processing which is depending on the cascade effect ?
 - Any “outside” applications manipulating data ?
 - Any API’s available which makes sure integrity isn’t violated
- All questions answered with NO
 - Of course there is no guarantee someone with the appropriate DML authorizations can “make a mess” if the system enforced RI is NOT in place

Investigate Referential Integrity

- This application behaved like an “Honor Student”, so..
 - I suggested the application group to DROP all the FOREIGN KEYS - but keep the Primary keys
 - 💣💣💣 (this scared the development manager to death)
 - The impact was NOT insignificant (average cost per Insert)

	CPU-SEC	ELAPSED-SEC
WITH FK	0.003924	0.035983
WITHOUT FK	0.002329	0.019938
SAVINGS	40.7%	44.6%

- No Buffer Pool changes implemented
- A nice reduction in GETP requests too
- Don't go home and drop all your FK's – define a test environment and verify your results with your workload where it is “safe” to remove the foreign keys

Status of the Analysis

- Where are we now in the process
 - Foreign keys dropped
 - A couple of indexes dropped since they were “duplicate”
 - One statement / JDBC function removed from the application.
- Issues still to be investigated
 - Millions of locks observed
 - What can be done (if anything) to the numerous heavy select statements
 - What about compression – and what is the impact
 - Lots of catalog GETP requests

Agenda

- Performance complaints – where to start
- Analysis of executed SQL
- Referential Integrity analysis
- **Locking analysis**
- Dynamic SQL analysis
- Compression analysis
- Reorg avoidance / considerations
- Wrap-up and Steen's philosophy

Locking Analysis

- Two possible methods to decrease locking
 - 1) Lock Avoidance possible when CURRENTDATA(NO) used as Bind option
 - 2) WITH UR - Uncommitted Read in select statements
 - FALSE
 - No PAGE locks or LATCH taken
 - Commit is not necessary
 - Rows can be “incomplete”
 - TRUE
 - UR gives you the smallest possible number of locks
 - DB2 will still issue locks for DBD, Package, Plan etc. in order to ensure structure is stable
 - SQL statements still get a CLAIM to prohibit e.g. Utilities co-existence (need to drain readers incl. OLR in final switch phase)
 - Each page will have a latch to ensure the page is “stable” – meaning not half the row inserted/updated or row about to be moved to another page

Locking Analysis

- Two of the heavy select/cursor statements measured to see the impact when using “lock reducing” methods
 - **OPEN CURSOR with ORDER BY**

	CPU-sec	Locks	Claims	
CURRENTDATA(YES)	38,828	141,867	1	
CURRENTDATA(NO)	35,925	136	1	

Remember every LOCK taken also needs to be released !!!

- **SELECT STATEMENT WITH UR**

	CPU-sec	Locks	Claims	
CURRENTDATA(YES)	82,152	66	31	
CURRENTDATA(NO)	77,858	9	3	

Locking Analysis

- The Packages were rebound with CURRENTDATA(NO)
- Select statements were verified for UR implementation
 - Don't use WITH UR without looking at the impact
 - What is the impact if a row being read by the statement using WITH UR - is deleted or updated and then rolled back – does it impact what this specific select statement would do ?

Agenda

- Performance complaints – where to start
- Analysis of executed SQL
- Referential Integrity analysis
- Locking analysis
- **Dynamic SQL analysis**
- Compression analysis
- Reorg avoidance / considerations
- Wrap-up and Steen's philosophy

Dynamic SQL Analysis

- Is a dynamic statement a dynamic statement – or can it be a “fake” static statement ?
- The answer is the TWO FAMOUS WORDS

IT DEPENDS !!!

Dynamic SQL Analysis

- A static statement is ready to execute (in general) because
 - The package was bound using VALIDATE(BIND)
 - The package is **valid and operative** – this can be impacted by several actions
 - A dependent object has been dropped and re-created without a subsequent REBIND
 - A dependent object has been altered (e.g. using OSE) without a subsequent REBIND
 - Rebind can be done manually – or DB2 can do it automatically first the package is referenced (unless the corresponding ZPARM parameter has been disabled)
 - The statement was validated by DB2 and the Optimizer knows which path to use

Dynamic SQL Analysis

- Normally - a dynamic statement needs to go through several steps prior to execution.
 - User authorization checking
 - Object validation
 - Statement validation
 - Access Path selection – especially AP selection by the Optimizer can be a very time consuming process
(I've seen cases where the Prepare took 80% of the execution CPU time)
 -
- A huge portion of the prepare can potentially be eliminated for dynamic statements
- A couple of requirements need to be in place
(we will get to these in a couple of slides)

Dynamic SQL Analysis

- Consider these two scenarios of dynamic statements and the DB2 dynamic statement cache is enabled

SCENARIO 1:

```
SELECT COL3 from TB1 where COL1 = 'ORA' and COL2 > 1000;  
SELECT COL3 from TB1 where COL1 = 'DB2' and COL2 > 50000;  
SELECT COL3 from TB1 where COL1 = 'IMS' and COL2 > 7000;
```

SCENARIO 2:

```
SELECT COL3 from TB1 where COL1 = ? and COL2 > ?;  
SELECT COL3 from TB1 where COL1 = ? and COL2 > ?;  
SELECT COL3 from TB1 where COL1 = ? and COL2 > ?;
```

(parameter markers or host variables instead of constants/literals)

- The two statements look very identical but the behavior between the two scenarios can be completely different seen with the performance glasses

Dynamic SQL Analysis

- One tuning parameter for dynamic statements is utilization of the Dynamic Statement Cache
 - The dynamic statement is placed in the cache first time it's being executed
 - The access path is chosen by the Optimizer
 - The next execution of the same statement will reuse the cached prepared statement IF

The statement is 100% identical

- Using literals / constants instead of parameter markers or host variables make the reuse almost impossible
- As mentioned earlier – the prepare can potentially be more expensive than the execution of the statement

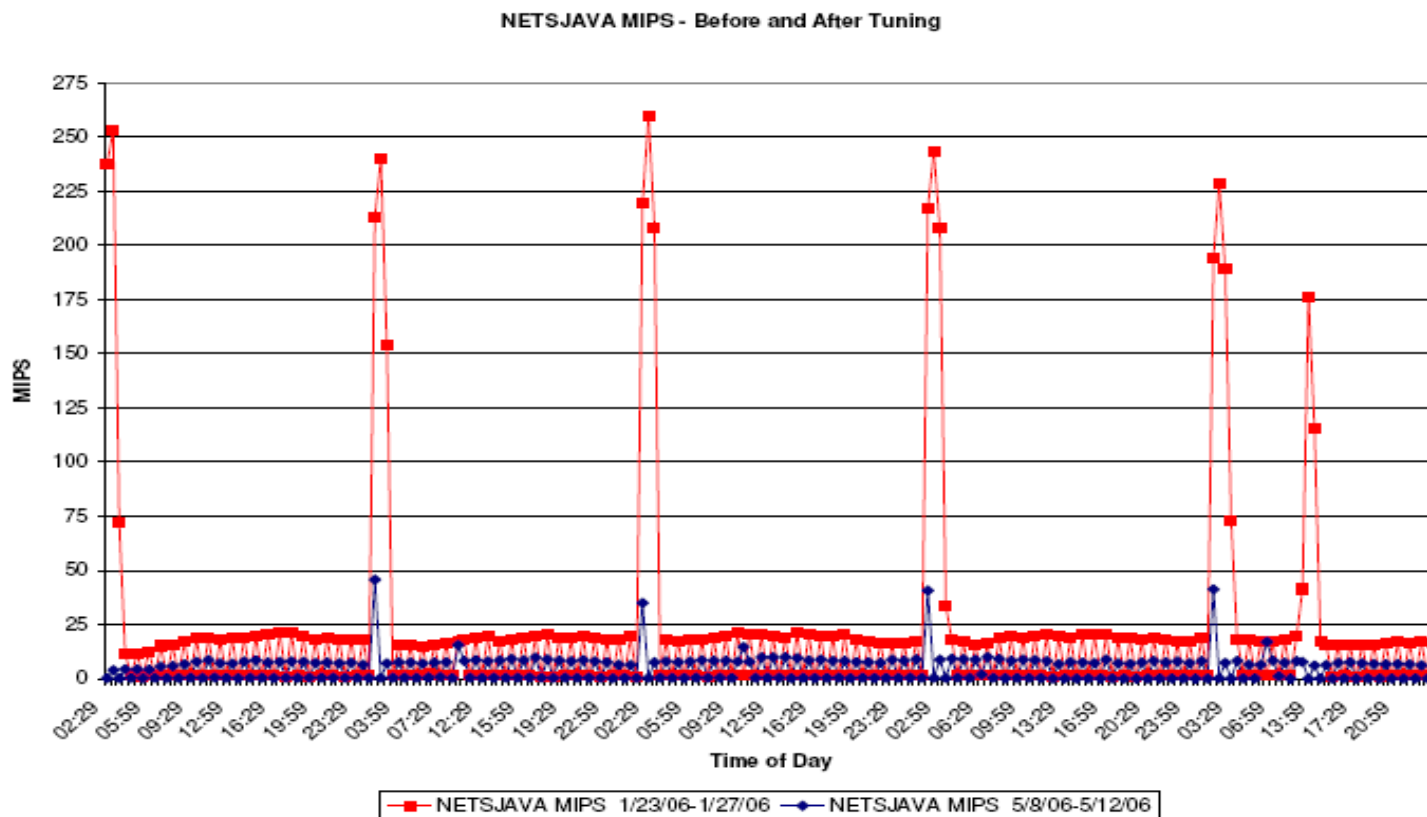
Status of the Analysis

At this time we have made great progress in terms of trying to save CPU:

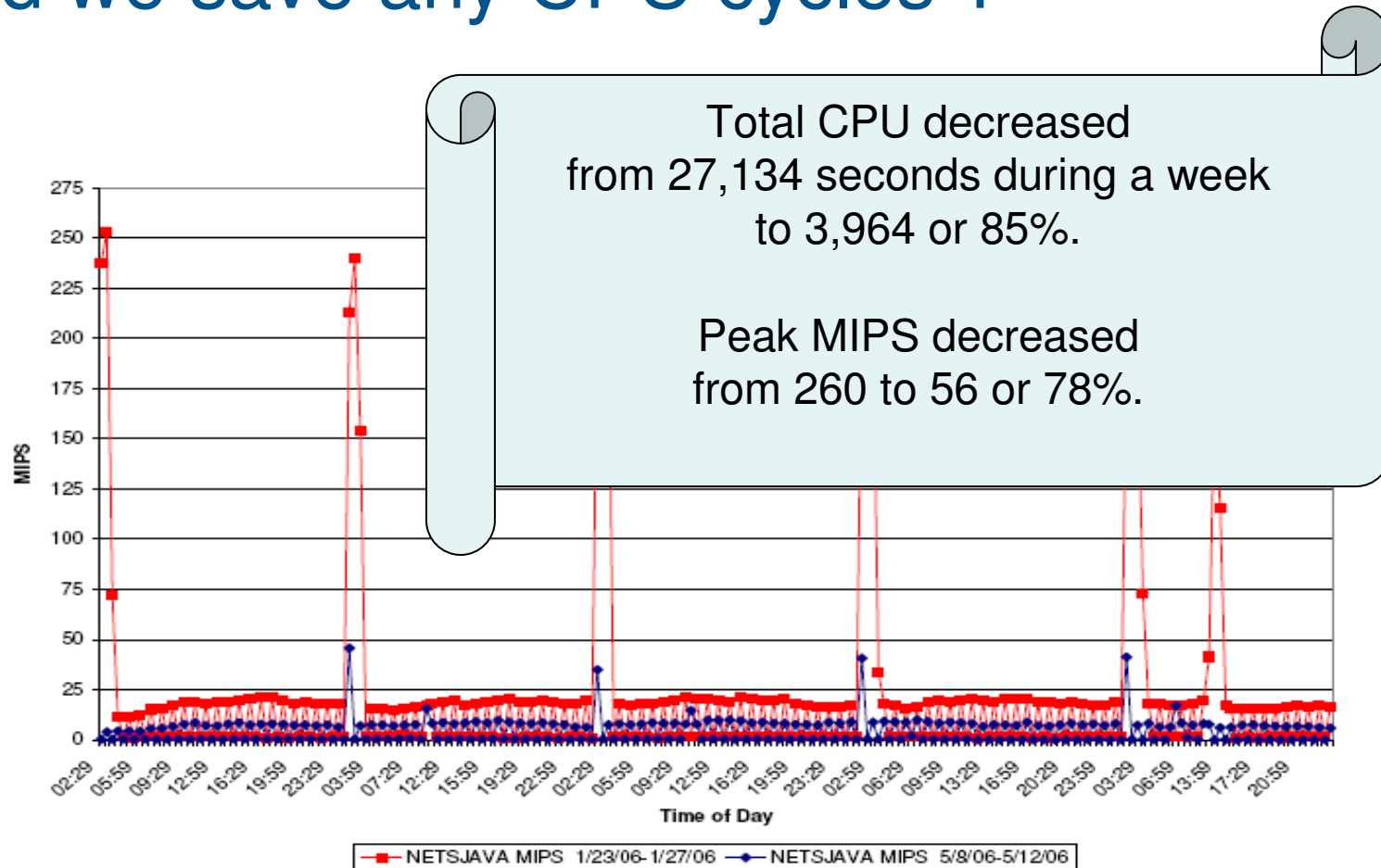
- 1) Removing RI resulted in about 40% CPU
- 2) Lock avoidance gave a few additional percent CPU savings
- 3) The third initiative was redesigning the Inserts to use parameter markers instead of literals/constants

What was the total savings –
CPU as well as MIPS ?

Did we save any CPU cycles ?



Did we save any CPU cycles ?



Did we save any CPU cycles ?

- What can be done to save more ?
 - Convert SELECT statements to use parameter markers – impact not checked yet, but looking at the Insert performance (considering the Insert savings)
 - Compress tablespaces ?
 - Reorg avoidance
 - Isolate objects in own buffer pool to gain better hit ratio

Agenda

- Performance complaints – where to start
- Analysis of executed SQL
- Referential Integrity analysis
- Locking analysis
- Dynamic SQL analysis
- **Compression analysis**
- Reorg avoidance / considerations
- Wrap-up and Steen's philosophy

Tablespace Compression

- History
 - Compression was introduced many years ago (DB2 V3.2 ?)
 - Adoption was very slow the first couple of years
 - Select statements performed pretty well
 - Overhead too big for Insert, Update and Delete
 - CPU overhead could not always be justified by space savings
 - Compression dictionary took up space in EDM pool
 - Performance has improved a lot over the past decade
- Impact
 - Until recently it hasn't been possible to unload from an incremental image copy since compression dictionary was not included in image copy
 - Log records are compressed for compressed objects – might impact Log Tools in use

Tablespace Compression

- Why compress ?
 - More rows in a page (don't forget MAXROWS 255 parameter)
 - More rows might result in fewer GETP requests
 - Fewer GETP requests might result in better BP hit ratio
- But
 - Small tablespaces might grow due to compression dictionary
 - If small row size, compression might not save space
 - Compressed data can result in higher processing cost
 - Compressing a row is more expensive than decompressing
 - Random access might not benefit from compression

Tablespace Compression

- This scenario:
 - The largest table in this scenario had a row size 50
 - Compression ratio of 58%
 - Lots of inserts statements
 - A few expensive select statements hitting many but not all rows in the single largest table
- Inserts after compression turned on
 - GETP requests decreased a little (a few percent)
 - CPU costs went up a little (a few percent)
- Selects after compression turned on
 - First analysis of heavy select statements showed decrease in GETP access (not a surprise)
 - First analysis illustrated CPU increase (a surprise to me)
 - Still positive I could find some savings

Tablespace Compression

- IBM Reorg output after compression turned on for the same but much larger tablespace (learn from the output !)

```

DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = RASST02.RASST02R
DSNUGUTC - REORG TABLESPACE NMJAVAXX.NUSPACE UNLOAD CONTINUE SORTDEVT SYSDA S
DSNURULD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=7123669 FOR TA
DSNURULD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:49
!M71A DSNURBDC - DICTIONARY WITH 4096 ENTRIES HAS BEEN SUCCESSFULLY BUILT FROM
!M71A DSNURWT - COMPRESSION REPORT FOR TABLE SPACE NMJAVAXX.NUSPACE

333922 KB WITHOUT COMPRESSION
138748 KB WITH COMPRESSION
    58 PERCENT OF THE BYTES SAVED FROM COMPRESSED DATA ROWS

    100 PERCENT OF THE LOADED ROWS WERE COMPRESSED

    50 BYTES FOR AVERAGE UNCOMPRESSED ROW LENGTH
    22 BYTES FOR AVERAGE COMPRESSED ROW LENGTH

    87429 PAGES REQUIRED WITHOUT COMPRESSION
    38533 PAGES REQUIRED WITH COMPRESSION
    55 PERCENT OF THE DB2 DATA PAGES SAVED USING COMPRESSED DATA

!M71A DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=7123669 FOR TABLE
    
```

Tablespace scan queries showed an increase in CPU of about 04 % but a huge 60% decrease in GETP requests – does DASD savings justify CPU increase ???

Tablespace Compression

- Conclusion
 - Never just “DO IT”
 - Set up a scenario and measure before and after
 - Make sure the entire workload is executed before a decision is made
 - Need to compare the increased CPU cost from the insert statements with the CPU savings from the selects
 - If Buffer Pool shortage is an issue – compression can be great
 - Try to execute your benchmarks in a live environment – not a sanitized environment (“things” might behave differently under “unusual” conditions
 - It is very easy to make the wrong decision

Agenda

- Performance complaints – where to start
- Analysis of executed SQL
- Referential Integrity analysis
- Locking analysis
- Dynamic SQL analysis
- Compression analysis
- **Reorg avoidance / considerations**
- Wrap-up and Steen's philosophy

Reorg Avoidance

- Should we really care about Reorg ?
 - It's almost 100% online
 - It is getting faster and faster
 - We will benefit from well organized data and index when not using random access
- We should care in most cases – think about WHY a reorg is executed !!
 - We reorg to keep data in a predefined / desired order
 - If reorg isn't executed, performance might decrease for some applications / SQL statements
 - So – when the reorg is executed – the impact might already be noticed by the users and applications, meaning – performance is degrading and costs \$\$\$
 - Monitor the CPU usage and/or GETP activity before and after a reorg to see if the reorg really matters

Reorg Avoidance

- Bottom line – the best reorganization is the one never executed !
 - Keep a log of which objects/partitions are being reorg'ed
 - Log everything in a DB2 table including the date
 - Log the reason why the reorganization was executed
 - Extents
 - Clusterratio
 - Pseudo-deleted entries
 - Free space gone or too much free space
 - Farindref or Nearindref
 - Etc.
 - Once a month or once every quarter – review which objects have been reorganized more than twice
 - Adjust parameters in order to avoid some of the reorg's

Agenda

- Performance complaints – where to start
- Analysis of executed SQL
- Referential Integrity analysis
- Locking analysis
- Dynamic SQL analysis
- Compression analysis
- Reorg avoidance / considerations
- **Wrap-up and Steen's philosophy**

A Tuning Approach

- ✓ Most important – find “your way” and be comfortable
- ✓ Spend time and get to know the application / environment
- ✓ Ask questions
- ✓ Every time you “see something” – make a note (I like yellow stickers in my notebook)
- ✓ Create “simulation” environment or duplicate objects etc. to test your theories prior to production implementation
 - ✓ (compress is a good example from this case)
- ✓ One measurement isn’t enough – like benchmarks, execute the SQL or job multiple times to get the average values
- ✓ Only change one thing at the time and measure before/after – you will learn a lot and can benefit from your findings later
 - ✓ If I had changed COMPRESS, removed the FK’s, changed the literals to parameter markers – how could I find out where the “biggest bang for the bucks” were – AND – duplicate my findings / efforts elsewhere

May 6-10, 2007

San Jose Convention Center

San Jose, California, USA

Any Questions ?

IDUG® 2007

North America



GoFurther

How to be a Hero by Taking Small Steps

Steen Rasmussen

CA

steen.rasmussen@ca.com