Managing DB2 Performance in a Heterogeneous Environment

Jim Wankowski
Quest Software
Agenda

• Defining Performance
• Monitoring Methods
• Identifying/Resolving Performance Issues
  ✓ Memory Management
  ✓ Physical Design Considerations
  ✓ Space Management
• SQL Design
  ✓ Coding Tips
  ✓ Optimization
  ✓ Tuning Examples
What is Performance?

Performance: *noun*

“The fulfillment of a claim, promise, or request”

- How does your company define performance?
  - System availability
  - Transaction throughput
  - Minimum response times (SLAs)
Subsystem/Instance Monitoring

All aspects of the DB2 subsystem or instance need to be monitored

- Take a look at the big picture
  - Think of DB2 as an ecosystem

- Do not tune for the sake of tuning!
  - Where are your bottlenecks?
Monitoring Methods

z/OS

- **Instrumentation Facility Component (IFC)**
  - **Statistics**
    - Global statistical data for overall health of subsystem
  - **Accounting**
    - Thread level statistics
    - Good for monitoring specific applications
  - **Performance**
    - Most detailed $$$
    - Only use for short periods

LUW

- **Snapshot Monitor**
  - Show status of database for an instant in time
    - Sort
    - Locks
    - Table activity
    - BP activity
    - UOW
    - SQL
  - Low overhead (~5%)

- **Event Monitor**
  - Historical collection of data
  - More overhead (~10-20%)
  - Main focus on application statistics
    - Tables, Deadlocks, Tablespaces, BP, connections, Statements, Transactions
Memory Management

**z/OS**
- EDM Pool
- RID Pool
- Sort Pool
- Buffer Pool

**LUW**
- Catalog Cache
- Package Cache
- Sort Heap
- Buffer Pool
- Locklist

Minimizing the amount of disk access should be a key performance objective
Memory Usage

z/OS

EDM Pool
- “System Bufferpool”
  - Minimizes I/O against catalog and directory
  - 3 pools- EDM,DBD, Statement
- Contains
  - DBD (Database Descriptor)
  - CT (Cursor Table)
  - PT (Package Table)
  - SKCT (Skeleton Cursor Table)
  - SKPT (Skeleton Package Table)
  - Plan/Package authorization Cache Dynamic SQL skeletons (Dynamic SQL caching active)

LUW

Catalog Cache
- Minimizes I/O against catalog
- Contains
  - SYSTABLES information
  - Authorization information
    - SYSDBAUTH
    - Execute privileges for routines
  - Real time statistics cache (9.5)

Package Cache
- Minimizes I/O against catalog
  - Loading packages
  - Having to prepare Dynamic SQL
Performance Implications

**z/OS**

**EDM Pool**
- Increased I/O activity against DSNDB01
  - SCT02
  - SPT01
  - DBD01
- Increased response times due to loading the SKCTs, SKPTs and DBDs
- Re-preparation of Dynamic SQL
- Fewer threads used concurrently, due to a lack of storage
- Resource unavailable “-904”

**LUW**

**Catalog Cache**
- Increased bind times
- Increased compile times
- Increased time to check DB and execution privileges

**Package Cache**
- Slower response time with Dynamic SQL
What to Monitor — z/OS

• EDM Pool hit ratio should be at least 80%
  – PT/CT’s
    • 80-90%
  – DBD requests
    • 100%
  – Pages used for Package/Cursor tables <50% of pool
What to monitor – LUW

- Catalog Cache hit ratio
- Catalog Cache lookups
  - $\text{Cat\_cache\_lookups}$
- Catalog Cache Inserts
  - $\text{Cat\_cache\_inserts}$
- Catalog Cache Overflows
  - $\text{Cat\_cache\_overflows}$

- Package cache hit ratio
- Package cache overflows
  - $\text{Pkg\_cache\_num\_overflows}$
- Package cache lookups
  - $\text{Pkg\_cache\_lookups}$
- Package cache inserts
  - $\text{Pkg\_cache\_inserts}$
- Package cache high water mark
  - $\text{pkg\_cache\_size\_top}$
Sorting

**z/OS**

**Sort Pool**
- Memory area for all sort activity
- Sort pools should be made as large as possible
  - 240K – 256MB
    - The larger the sort pool, the more efficient the sort
    - Default to 1MB
- Minimizes externalizing to physical sort files
  - DSNDB07

**LUW**

**Sort Heap**
- Number of pages available for private or shared sorts
  - Used by Optimizer for determining access paths
    - Sorting
    - Hash Joins
    - Index ANDing
    - Dynamic bitmaps
Optimizing Sort

z/OS

- **DSNDB07**
  - Do not use BP0!
  - 4 –5 equally sized datasets
  - Keep in primary space
  - Separate volumes
  - Separate BP’s
  - Also used for:
    - View, temp table,
    - Nested table expression materializations
    - Non-correlated in-lists

LUW

- **Avoid Sort Overflow**
  - If `sortheap` too small, sort will overflow into temp database tables

- **Avoid Non-Piped Sorts**
  - If sorted information must be stored in a temporary table vs. memory (`sortheap`)
  - Determined at optimization
Optimizing Sort – All Platforms

- Proper indexing can minimize sorting
- Avoid ORDER by, GROUP By, DISTINCT
- Avoid sorting VARCHARs
- Only select required columns
What to Monitor - LUW

- **Sort Heap Overflow**
  - *Sortheap*

- **Sort Heap Threshold**
  - *Sheapthres*
RID Pool – z/OS

“Record Identifier” Pool
- Enforces unique keys during multi-row updates
- Used for storing and sorting RID’s for:
  - List Prefetch
  - Multiple index access
  - Hybrid Joins

• Performance Implication
  - If RID pool is too small above access paths revert to TS scans
What to Monitor – z/OS

- **Insufficient Pool Size**
  - RID pool too small
  - Recalculate size

- **RDS Limit**
  - RID list > 25% #rows in table
    - Prefetch turned off and TS scan results
  - Determined at Bind time
  - Have tables grown since BIND?
    - Make sure stats are accurate
      - RUNSTATS/REBIND

- **DM Limit**
  - RIDs req’d to satisfy query > 16M
  - TS scan results
  - Is TS Scan best access?
    - Re-evaluate indexes
    - Add additional filtering

---

### SQL statistics

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Per sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commits</td>
<td>18,836,894</td>
<td>49.71</td>
</tr>
<tr>
<td>Rollbacks</td>
<td>4,792</td>
<td>0.01</td>
</tr>
<tr>
<td>Incremental binds</td>
<td>70,285</td>
<td>0.19</td>
</tr>
<tr>
<td>Runtime reoptimizes</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Direct row success</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Direct row, index use</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Direct row, tbs scan L</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>RID list success</td>
<td>520,010,971</td>
<td>1,372.21</td>
</tr>
<tr>
<td>RID failure, storage</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>RID failure, RDS limit</td>
<td>926</td>
<td>0.00</td>
</tr>
<tr>
<td>RID failure, DM limit</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>RID failure, size limit</td>
<td>1</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Bufferpools

**z/OS**
- 80 bufferpools
  - 50 – 4K
  - 10- 8k, 16k, & 32K respectively
- Share across subsystem

**LUW**
- **IBMDEFAULTBP** automatically created with database
  - Additional pools created with DDL
- Hidden Bufferpools
  - 4k, 8k, 16k, & 32K
- Share only within individual databases
What to Monitor

**z/OS**

- Bufferpool hit ratio
  - \((\text{GETPAGES} - \text{pages read})/\text{GETPAGES}\)

- Page externalization
  - Excessive writes to disk

**LUW**

- Overall hit ratio
  - Total # data/IX reads by BP

- Data hit rate

- Index hit rate

- Asynchronous page cleaners
  - \(\text{Num_iocleaners}\)
Effective Use of Bufferpools

Separate bufferpools for:

- Catalog and Directory (BP0+)
- DSNDB07
- Tablespaces
- Indexes
  - Large VPSIZE
  - More than one
- Small, read-only tables
- Large tablespaces w/random access
- Small frequently updated tables and IX’s
- Test environment for isolating test cases

The single biggest performance mistake is to stick everything in BP0!

z/OS

Separate bufferpools for:

- System Catalog
- Sequentially scanned tables
- Temporary table spaces
- Small frequently updated tables
- Small read-only tables
- Large tables w/random access

LUW
Locklist

Amount of storage allocated to a database for locking

Possible Performance Implications:

- Lock Escalations
  - Decrease in concurrency
  - Degradation of performance due to lock waits
- Deadlocking
- SQLCODE -912
  - Maximum # locks reached in database

How to avoid:

- Frequent COMMITs
- Consider using LOCK TABLE for application performing large number of updates
- Use CURSOR STABILITY
What to Monitor

- **Lock List**
  - *Locklist*
- **Maximum Locks**
  - *Maxlocks*
- **Lock Escalations**
  - *lock_escals*
Automated Tuning - LUW

- **Self Tuning Memory Manager**
  - Bufferpools
  - Locklist
  - Package Cache
  - Sort Heap
  - Database Shared Memory

- **DB2 automatically adjusts allocations based on workloads**
Physical Design Considerations

**z/OS**

- **Simple**
  - Deprecated in V9

- **Segmented**
  - Best for most applications
  - Superior I/O performance over simple

- **Partitioned**
  - Best for large volumes of data
    - Manageable w/partition independence

- **Universal (V9)**

- **Large**
  - LOB, CLOB, BLOB

**LUW**

- **Regular**
  - Extent size
    - 2-256 pages
  - Space allocation
    - SMS vs. DMS

- **Temporary**

- **Large**
  - Much easier to manage than z/OS
    - No auxiliary tables

- **Automatic (V9.5)**
Index Design Considerations — All Platforms

- Define primary keys with unique indexes
- Use indexes on foreign keys
- Define unique indexes with include columns
  - For index only access
- Define Indexes on join columns
- Use indexes on highly sorted columns
  - Save clustering index for this purpose
- Minimize use of indexes on heavily updated or inserted tables
  - Avoid use of data tablespace (LUW)
Maintenance

- Proper maintenance is critical for optimal performance
  - Reorganization
  - Statistics Collection
Reorg

• What causes fragmentation?
  – Insert/Update
    • Check PCTFREE and FREEPAGE
  – VARCHAR fields being updated
Monitoring for TS and Table REORGs

**z/OS**

- **Cluster Ratio < 95%**
  - SYSINDEXES
    - CLUSTERATIO < 95%
  - SYSINDEXPART
    - FAROFFPOS > 10% of CARD
- **Excessive row relocation**
  - SYSTABLEPART
    - NEARINDREF+FARINDREF > 10% of CARD
- **Excessive extents (>50)**
- **Excessive drop space**
  - Simple TS only
    - PERCDROP > 10%
- **LOB tablespaces**
  - SYSLOBSTATS
    - ORGRATIO > 2

**LUW**

- **Cluster Ratio < 90%**
  - SYSCAT.INDEXES
    - CLUSTERRATIO
- **Overflow of Rows**
  - SYSSTAT.TABLES
    - OVERFLOW
- **Fetch Statistics**
  - SYSCAT.INDEXES
    - Small # of
      - AVERAGE_SEQUENCE_FETCH_PAGES
      - Growth of AVERAGE_RANDOM_FETCH_PAGES
- **Empty Pages**
  - SYSCAT.TABLES
    - FPAGES-NPAGES
Statistics

Accurate statistics are a critical factor for performance monitoring and tuning

RUNSTATS provides statistical information for:
   1. Optimization of SQL
   2. Monitoring status of objects
RUNSTATS

• **When to run RUNSTATS:**
  ✓ After LOAD*, REORG and REBUILD IX
  ✓ After creating new index
  ✓ After heavy insert, update, delete activity

• **Rebind After RUNSTATS?**
  ✓ New index created
  ✓ Cluster ratio changes <> 80%
  ✓ Changes more than 20%
    • Cardinality
    • Index leaf distance
Automated Statistics Collection

**z/OS**

- **Real Time Statistics**
  - Stored Procedure
    - DSNACCOR
  - Space repository
  - Monitor for
    - REORG
    - RUNSTATS
    - Image Copy

**LUW**

- **Added in V9**
  - Automatically scheduled during optimal maintenance window
  - Maintenance Policy Definition

- **“Throttled” RUNSTATS**
  - Adjusts resources based on activity
  - Statistics Profiling allows for customization

- **Real-Time Stats (V9.5)**
  - Asynchronously collected prior to SQL execution
Application Design

- SQL design considerations
- Optimization

Efficient application design is the single most important aspect of an efficiently performing subsystem
SQL Coding Factors

• Many ways to write a SQL to return the same data
• Small differences in coding SQL can have great performance implications
• Different SQL versions may produce different access plans
Optimizer

**z/OS**

- Fixed optimization
- **Optimization Hints**
  - Used to maintain old access path
    - Across DB2 versions
    - After Rebinds
  - Must be turned on at install time
  - Need to modify PLAN_TABLE
  - Manual Process
- **New Visual Explain**
  - Utilizes “hidden” plan_tables

**LUW**

- More flexible than z/OS
  - 7 levels of optimization
  - Adjusted based on query complexity
Optimizer Class – LUW

- **DB2 Optimizer Class**
  - Values are between 0 and 9, default is 5
    - Determines the intensity used by the DB2 SQL Compiler when rewriting SQL
    - Dynamic SQL can’t spend time optimizing, use lower class
    - Static SQL optimizes once, use a higher class
    - “dft_queryopt” database setting
    - SET CURRENT QUEREY OPTIMIZATION n

<table>
<thead>
<tr>
<th>Level</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Minimal amount of optimization. Only recommended for very simple SQL accessing well indexed tables. Only nested loop joins and IX scans enabled.</td>
</tr>
<tr>
<td>1</td>
<td>Similar to 0 except Merge Scan and TS scan enabled.</td>
</tr>
<tr>
<td>2</td>
<td>Recommended for very complex queries which are infrequently executed in a decision support or OLAP environment.</td>
</tr>
<tr>
<td>3</td>
<td>Closest to OS/390 optimizer. Recommended for queries with 4 or more joins.</td>
</tr>
<tr>
<td>5</td>
<td>DEFAULT – Most cost effective method for mix of simple and complex queries. Optimization will be automatically reduced for complex dynamic SQL if optimizer determines that the resources are not necessary.</td>
</tr>
<tr>
<td>7</td>
<td>Same as 5 except optimization not reduced for complex dynamic SQL</td>
</tr>
<tr>
<td>9</td>
<td>Used to determine whether more comprehensive optimization can generate better access plan for very complex long running queries using large tables</td>
</tr>
</tbody>
</table>
Optimization Tips – z/OS & LUW

- Make sure statistics are accurate
- Use stage 1 vs. stage 2 predicates
- Only select required columns
  - Avoid SELECT *
- Keep predicates as restrictive as possible
  - Minimize number rows returned
  - Minimize program filtering and let DB2 do the work
- Order predicates from most to least restrictive
- Avoid sorts
  - ASC/DESC indexes can help avoid excessive sorting
- Avoid UNION clause
  - IN, BETWEEN, or CASE

- Use Appropriate Optimizer Class (LUW)
  - 0 through 9 (5 default)
  - Use lower class for Dynamic SQL
  - Higher class for static
- Use “Optimize for n Rows”
  - Minimizes optimization cost
- Use “Fetch First n Rows”
- Avoid Data Type Conversions

EXPLAIN!!!
Tuning Examples
Optimizer Class Scenario – LUW

- **Scenario** – Complicated Join statement performing poorly (2 table join x 10 Unions)
- **Application** – Dynamic SQL
- **Optimizer Class** – 0
- **Possible Solution** – Up the Optimizer Class since preparation only occurs at bind time

<table>
<thead>
<tr>
<th>Optimization Class</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Cost (Timerons)</td>
<td>493416</td>
<td>148563</td>
<td>148563</td>
<td>148563</td>
<td>148563</td>
<td>148563</td>
<td>148563</td>
</tr>
<tr>
<td>I/O Cost</td>
<td>36933</td>
<td>23980</td>
<td>23980</td>
<td>23980</td>
<td>23980</td>
<td>23980</td>
<td>23980</td>
</tr>
<tr>
<td>Elapsed Time</td>
<td>0:0:1.234</td>
<td>0:0:0.891</td>
<td>0:0:0.828</td>
<td><strong>0:0:0.703</strong></td>
<td>0:0:0.750</td>
<td>0:0:0.796</td>
<td>0:0:0.797</td>
</tr>
</tbody>
</table>

35% improvement in elapsed time by bumping optimizer class up to 3
Union vs. IN vs. BETWEEN

- **Scenario** – Complicated Join statement performing poorly (2 table join x 10 Unions)
- **Application** – Dynamic SQL
- **Possible Solution** – Rewrite query with IN or BETWEEN clause

### Processor Cost (ms) Elapsed Time

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th></th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNION</td>
<td>25</td>
<td>0:0:0.266</td>
<td>148563</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>6</td>
<td>0:0:0.157</td>
<td>25875</td>
</tr>
<tr>
<td>IN</td>
<td>12</td>
<td>0:0:0.141</td>
<td>25832</td>
</tr>
</tbody>
</table>

**IN Clause performs best on both platforms**
- **+47% for z/OS**
- **+62% for Windows**
Summary

• Build a strategy for how you will monitor/tune your DB2 environment
  – Try to identify most critical applications and begin focusing on those
  – Set realistic performance expectations
  – Tune, monitor, tune, monitor…
Jim Wankowski

Quest Software

Jim.Wankowski@quest.com

Download whitepapers at:
WWW.QUEST.COM/DB2