How to Improve your Ingres Performance

May 2008
About the Presenter

- Chip Nickolett has been using Ingres since 1986 as a Developer, DBA, and Consultant
- He has performed numerous benchmarks for hardware vendors and customers
  - In 2003 he conducted benchmarks with Ingres Engineering that led to architectural and configuration improvements in Ingres 2.6
- He is currently the Director of Consulting Services in the Americas and can be reached via email at chip.nickolett@ingres.com
Agenda

- Defining Performance
- Areas for Performance Improvement
- The Importance of Gathering Metrics
- Identification of Performance Issues
- Gathering & Analyzing Print Query Information
- Causes of Query Performance Problems
- Storage Structure and Indexing Tips
- Statistics
- Summary
- Q&A
Defining Performance - Goals

- Performance means different things to different people, so it is important to have a clear understanding of the goals before you start
  - What are the specific goals, and are they reasonable?
  - Get specific examples of response time issues
  - Understand what “normal” looks like for the system and try to identify trends and patterns
  - Lock waits and concurrency issues are often identified as performance problems
  - Understand the operating environment
Defining Performance - Issues

- Can you develop a profile of the problem?
  - Does it only happen certain times of the day or days of the week?
  - Do issues exist all the time or just under load?
  - Do you see other symptoms? The errlog.log is a great place to look and to correlate with those symptoms.
  - Does it only happen to certain applications or technologies?
  - Is it a client, server, or both issue?
The Importance of Gathering Metrics

- Relevant metrics will allow you to:
  - Perform trend analysis
  - Correlate activity with symptoms
  - Have quantifiable measurements to demonstrate both performance improvements and degradation
  - Proactively anticipate future needs (capacity management) and potential problems
  - Develop performance profiles
Areas for Performance Improvement

- Look for the quick fixes first
  - DMF Cache Configuration: The “large” setting in “cbf” is a good starting point for most installations
  - Disk I/O: Are there hotspots? These can often be eliminated by distributing tables and indexes equally across multiple Ingres data locations
  - Locking: Locks per transaction, maxlocks, and lock level are easy changes to make
    - Rules of Thumb: 1,500-2,000 locks/tx; 1,000-1,200 maxlocks; explicitly set “level=page” if not row locking
Identification of Performance Issues

- Try to gather data on both Ingres and operating system performance at the same time
  - Are symptoms or issues identified related, or does one contribute to the other?
- Use “ipm” or Visual DBA to identify lock waits and find the session holding the locks
  - Work to find the blocking session
  - Often that session will have a long lock list
  - Transaction and Query design review should be performed on the problem session
Identification of Performance Issues

- Use the trace point “dm420” to compare the level of “reads” (single buffer reads) to “greads” (group buffer reads) to identify table scanning activity
  - In most environments the ratio of reads : greads should be highly skewed towards single buffer reads
  - Table scans are a very common performance issue
  - This is actually more important than the cache hit ratio at first, but once the DMF cache sizing and table scan issues are resolved the cache hit ratio should be evaluated
Identification of Performance Issues (cont.)

- Look for long running queries
  - White paper and monitoring tool download available on the Ingres.com website
- Try to isolate activity during spikes in activity and/or performance slowdowns
  - Automated data collection ideal
- Identify people having the ability to run ad hoc queries in a production environment
- **GOAL:** Identify suspect programs and queries
Identification of Performance Issues (cont.)

- The Ingres print query transaction logging tool is a great way to quickly identify areas for improvement
  - Transaction design issues that lead to concurrency problems
  - Query design issues that lead to performance problems
  - Candidates for “repeated” queries
  - Best of all, it doesn’t require programming changes in order to gather the data
Gathering Print Query Information

- Define the environment variable “II_EMBED_SET” to have the value “printqry”
  - A file named "iiprtqry.log" will be created in the directory a “program” is started
  - You need write permission to that directory or else the program will fail
  - Often an unsupported trace point (SC902) can be set to write this information to the Ingres errlog.log file from a client system
    - This has the potential to fill-up a necessary filesystem and should be monitored closely if implemented
Gathering Print Query Information (cont.)

- In an ODBC environment
  - Specify a log file for with the environment variable "II_API_LOG"
  - Specify the trace level of "5" in the environment variable "II_API_TRACE"
  - Will need to restart the Ingres/Net processes to take effect
  - The output will be slightly different than the examples shown
  - This can significantly degrade performance!
If you want the output file to have a different name or reside elsewhere:

- Define "II_EMBED_SET" to have a value of "printqry; printtrace; tracefile /tmp/iitrace.$$_.log; qryfile /tmp/iiqry.$$_.log;" (path and filenames can be changed to suit your needs and environment)

- This method is especially useful when there are multiple instances of an application running, such as in a production environment
Gathering Print Query Information (cont.)

- To just get the Query Execution Plans (QEPs) do the following (below).
  - By default this will generate a file named "iiprtttrc.log"
  - This approach collects the QEP information and the QE90 trace point information
  - define **ING_SET** to “set qep; set printqry; set trace point qe90;”
  - define **II_EMBED_SET** to “printtrace” (Note: “tracefile /PATH/FILE;" can be optionally be set)
Analyzing Print Query Information

- The “grep” command (Unix, Linux, or Cygwin shell on Windows) makes this quick and easy
  - grep “^Query [SR]” iiprtqry.log | more

- The output will look like this:

  - Query Send Time:          Thu May 15 11:29:38 2008
  - Query Send Time:          Thu May 15 11:29:38 2008
  - Query Send Time:          Thu May 15 11:29:38 2008
Analyzing Print Query Information (cont.)

- Look at the time between the “Query Send” and the “Query Response” as this is your execution time
  - Expect anywhere from 25-100+ queries per second
  - If you see a several second delay between the last “Query Response” and the next “Query Send” look at the program to try to understand what is going on between queries as it could be a performance issue
  - Identify areas where only a few queries run per second, or queries that take longer than a second to run
- Your eyes will quickly learn to spot these patterns
Once you have identified queries that appear to be slow, gather and analyze the Query Execution Plan (QEP) for those queries

- See the Ingres DBA Guide or the white paper “Finding and Fixing Troublesome Long-Running Ingres Queries” for more information on analyzing QEPs (both available for free download at ingres.com)

This will quickly help you target queries that are taking “too long” to execute
Causes of Query Performance Issues

- Disjoint Query
- Applying functions (such as “char” or “left”) to a key column
- Base table storage structure
  - Important to differentiate access versus uniqueness in some cases
- Secondary indexes
  - Missing or poorly designed
- Statistics
Storage Structure & Indexing Tips

- **BTREEs do work best in most cases**
  - Structure might be keyed on a **unique value** (such as “Emp_No”) or **common access method** (such as “Ord_No, Line_No”)
  - Structure can be used to **cluster similar data** to minimize I/O (e.g., clustering an Insurance claim table by “Region, LOB, Branch, Agent, Policy”, and then having indexes on “Claim_No” (probably unique) and “Policy”)
Storage Structure & Indexing Tips (cont.)

- Have as much “leading-edge” granularity in your keys (primary and secondary) when used as an access method
  - Reversing the order of what is often a natural data grouping can improve performance

- Try to satisfy the columns commonly referenced in the “where” clause with indexes
  - Note: If the query is fully satisfied using the index a visit to the base table will not be required, so adding columns commonly selected might help performance
Storage Structure & Indexing Tips (cont.)

- Don’t create a secondary index that has the same order as the base table structure
  - The optimizer will likely ignore it

- Indexing Rules of Thumb:
  - Index size should be 20% or less of the table size
  - Try to maintain 7 or less secondary indexes on a table
    (1-3 is ideal, 4-7 work well in an OLTP environment, 8+ indexes typically add 30% or more write (I/O) overhead and increase the time spent in the optimizer due to the number of plan choices)
It is very important to note that Ingres will only select a single access method per table. This means:

- No more than one index will be used as an access method
- Indexes need to be designed to support as much of the query as possible (single-column indexes just don’t do that)
- Reasonable overlap (i.e., redundancy) between indexes is fine as long as the leading-edge columns are not in the same order
Statistics

- **Good statistics usually help performance**
  - Default assumptions in lieu of statistics often estimate too large of a result set
  - Default optimization flags do not create enough histogram cells
    - Improved in Ingres 2006 r3
    - Use “optimizedb –zk –zu200 –zr200” flags instead
  - Statistics showing little granularity (e.g., having a count per value equal to 20% or more of the table volume) can lead to table scans
The trace point “QE90” will provide actual versus estimated effort for the query
- Ignore the “C” values (CPU) and focus on I/O
- Look at the “tups” (row) count estimates. Do they seem reasonable? Are the actual values and estimates close? Good statistics improve accuracy.
- Is there excessive amounts of “D” (disk I/O) for the volume of rows retrieved? This could indicate a table scan and the need to review the statistics for those tables.
Summary

- Most of the Ingres performance issues I’ve found center around the common issues and mistakes mentioned in this presentation.
- It is possible to quickly identify simple changes that will have an immediate positive impact.
- Once you have exhausted the simple changes, the next step is usually query, transaction and application design optimization.
- There are several helpful white papers and tools at Ingres.com, and more coming every month!
Questions?