Build Your Own Benchmark Center

Ready to Use Methodology & Tools

Randall Parman
Database Architect
DineEquity, Inc.

Doug Ebel
Benchmark & POC Center of Expertise
Teradata Corporation
“People who claim an investment yielded an improvement either measure it, guess it, or lie.”
-- Doug Ebel

- How benchmarks contribute to good EDW practice
- Lessons learned from different benchmark approaches
- How to defining a good benchmark of the DBMS
- Tools for selecting Queries and Tables from DBQL
- Query Driver
  - DBQL and Resuse Table Usage
  - Query Setup
  - Test setup
  - Monitoring and reporting results
  - Benchmark Wrap-up – Saving results
- Conclusion

Case Study: Benchmark at DineEquity

- Compare 2-node 5500C with 4-node 2580
- Looking for wall clock time comparisons
- How queries selected
  - Known long running queries
  - Gut feel for complexity
- Run methods
  - Serial
  - All in parallel
  - Multiple parallel streams
- Data generated for projected growth
Serial Exec.

12 Concurrent
24 Concurrent

60 Concurrent
Results of Benchmarking At DineEquity

- Provided a reasonable picture of performance
- Easy to control and manipulate
- Found that starting all queries at the same time did not provide a true picture of workload performance.
- Method for choosing queries was very random.
- Did “stumble” upon some workload management nuggets.
- Overall, process was good and led to request for doing this presentation.

How Benchmarks Contribute To Good EDW Practice

- Clearly understand impact of new release on all users ...
  - Case study: 5400 TD 6.2 vs 2550 TD 12
    - 2,132 production queries captured over 6 hr
    - 2,013 completed in 25% of the time of 5400
    - 16 ran longer (complex, redundant syntax, short duration)
  - Case study: SLES 9 -> 10, TD 12 -> 12.0.3, 18 ->33 nodes
    - Did SLES upgrade impact system? .. Or change in activity?
    - Concern that node upgrades may hide performance problem
    - 1400 actual queries extracted from DBQL
    - Proved SLES change < 1% performance
    - Proved no negative impact of software service pack
- Forecast impact before going to production
- Provide cost justification for expense of upgrade
- Adjustment of capacity plan – might defer upgrade
- Inexpensive validation that patches are performance neutral
- Identify impacts of physical data model tuning
Lessons Learned From Different Benchmark Approaches

- "Take all tables" approach: "It’s too difficult to determine which tables are needed"
  - Excess effort, space requirements, system processing
- Engineered queries: "Make that puppy pant"
  - Not representative of real workload
  - Success in artificial benchmark ≠ Success in production
- Manually executed from BI tool: "Everyone hit the Enter key at the count of three"
  - Not reproducible
  - Takes setup of more infrastructure (BI tool, network)
  - Hides DBMS performance differences with the additional overhead of network and client platform, client disks
- Execute same queries multiple times
  - Database gets faster after the first one
  - Not representative of real world

What Impacts Benchmark Performance?

1. Client disk speed
2. Client server speed
3. Client application design
4. **DBMS load utility used**
5. Client LAN bandwidth
6. Network congestion, propagation
7. Public LAN packet size
8. LAN bandwidth to node
9. **Node speed**
10. **Node O/S**
11. **DBMS, DBMS release**
12. **BYNET speed**

Private LAN provides multiple connections + Jumbo packet size
Too Often, When Client BI/ETL Application Is Used, DBMS Has Minor Time Impact In Timing

Conclusion? Teradata good at being idle?

ETL Test ran 32 Minutes

How To Focus On DBMS Measurement

- Data Loading
  - Execute data load utilities against data prepared by ETL application separate from ETL application
  - Use dedicated server (or TD Managed Server)
  - Use dedicated LAN, configure for Jumbo Packets
  - Use multiple connections from ETL server to nodes
- Data Retrieval
  - Capture queries from BI tool. Run without BI tool
  - Set Retlimit and RetCancel in BTEQ to limit rows sent back to client platform
  - Measure performance from Query StartTime to FirstRespTime on DBQL (resolution to .01 seconds)
Distilling Benchmark Objectives And Scope

- Benchmark for DW should be a subset of your vision or production
- **Focus on the DBMS**
  - Minimize the client software: BI/ETL tools

How To Distill Benchmark Objectives And Scope

- A DBMS Benchmark should be the **DBMS related subset** of your vision or production
  > Should be smaller in scale than a full production parallel
  > Ideally execute in 15-60 minutes to allow re-testing

- Make benchmark **representative** of your production
  > #’s of heavy, medium and light users
  > Queries selected from real production
  > Queries modified for each execution to represent variety of interests (geographies, products, time periods)
  > Should cover a few major tables plus their supporting lookups or dimensions.
Overview Of The Selection Methodology

0. Turn on DBQL logging for queries, objects
1. ID Tables of key interest
2. Sample queries that use those tables
3. ID other tables needed by Queries
4. Eliminate tables with minor use and their queries
5. Run tests and/or export DDL & Data

Identifying Data And Queries Via DBQL

Call SelectTable('db','tablepattern');
Call SelectQueryUsingTables('querypattern', mintablecount, samplesize);
Call SelectTablesUsedByQueries();
Call SelectTableUsage(); \(\text{Counts of queries using each table}\)
Call ReduceMinorTablesPreview(\text{MinUsage});
Call ReduceMinorTables(\text{MinUsage});
Call QueryRenumber();
Call SelectNeededObjects(); \(\text{Show table ...}\)
\(\text{Show view ...}\)
Benchmark Query Driver: TdBenchV5

• Features:
  > Tracks each test using TrackingTable + DBQL
  > Supports single stream and multi-stream tests
  > Has reporting views for analysis of results from DBQL and ResUsage by selecting a RunID
  > Supports mixed workloads of heavy, medium, light queries and update processes
  > Organizes DBQL, Resusage, and output log files

• Requirements:
  > Runs on Windows and Windows Server platforms
  > Implemented in DOS batch files, however, no skills required in Batch commands
  > Uses stored procedures. (For Teradata express DBMS on PC, need to be able to create stored procedures.)
Tracking Each Test – The TestTracking Table

- When test initiated, you are prompted for:
  - File to run (.SQL, .BTQ, .BAT, or .LST) in the Scripts Directory
  - Number of sessions
  - Prefix of the Worker logon ID’s to be used
  - Description of the test
  - Duration of the test.
- Unique RunID is assigned from TestTracking
  - Marks beginning & ending Timestamps for reporting from DBQL & ResUsage

Most Useful DBQL Tables For Benchmarks

- **DBC.QryLog:**
  - One row for every query execution. Each query identified by a QueryID and provides start/stop times, CPU usage, I/O, and information about skew.
  - This is the source for all results reporting.
- **DBC.QryLogObjects:**
  - For each QueryID, one row for every database object referenced.
  - This is the essential cross reference of tables ↔ queries.
- **DBC.QryLogExplanations:**
  - For each QueryID, captures explain text in a Large Object text field.
  - Before and after analysis can explain performance changes.
- **DBC.QryLogSteps:**
  - Provides detailed CPU, IO and duration for each step executed within the query.
  - Helps to understand where time is being spent and whether estimates might be improved by statistics.
- **DBC.QryLogSQL:**
  - Contains complete SQL for the QueryID. Default for DBC.QryLog is to contain first 200 characters.
Other Log Tables For Benchmark

- Resusage: enabled via CTL utility
  > Recommend 30 second interval, 30 second collect
  > ResUsage tables supported:
    - **DBC.ResUsageSpma**: *(Recommended)*
      - System-wide node information provides a summary of overall system utilization incorporating the essential information from most of the other tables
    - **DBC.ResUsageSawt**: *(Optional)*
      - Data specific to AMP worker tasks. Use this when you want to monitor the utilization of the AWTs and determine if work is backing up because the AWTs are all being used
    - **DBC.ResUsageSvpr**: *(Optional)*
      - Data specific to each virtual processor and its file system.
- **DBC.EventLog**: *(Always on)*
  > LogonOff information for finding interference to benchmark.

---

Begin With The End In Mind – You’re Going To Be Analyzing The Queries In DBQL

- Which DBQL QueryText easier to parse for reporting?
  
  **#1:**
  > SELECT DISTINCT a18.HOTEL_PROD_ID (NAMED HOTEL_PROD_ID ) ,
  > a18.HOTEL_DESC (NAMED HOTEL_DESC ) , pa19.nsearch_count ... 
  > SELECT a14.DAY_CODE (NAMED DAY_CODE ) ,
  > a15.PRICE_TEST_VER_ID (NAMED PRICE_TEST_VER_ID ) , ...
  > SELECT a13.HOTEL_ID (NAMED HOTEL_ID ) , MAXIMUM ( 
  > a13.HOTEL_DESC ) (NAMED HOTEL_DESC ) , ...

  **Or, #2:**
  > Exec xxx_Benchmark.Query01;
  > Exec xxx_Benchmark.Query02;
  > Exec xxx_Benchmark.Query03;
Avoid Benchmark Constructs That Aren’t Realistic

- **Which query sequence gives more realistic test? #1?**
  - Exec xxx_Benchmark.Query15;
  - Exec xxx_Benchmark.Query15;
  - Exec xxx_Benchmark.Query15;
  - Exec xxx_Benchmark.Query15;

- **Or, #2?**
  - Exec xxx_Benchmark.Query15 ('District21','2010-03');
  - Exec xxx_Benchmark.Query15 ('District14','2010-02');
  - Exec xxx_Benchmark.Query15 ('District08','2010-04');
  - Exec xxx_Benchmark.Query15 ('District31','2010-03');

Preparing Queries For Benchmarks

- **Best:** convert to Macros
  - Allows for simple parameterization
  - Identify with simple query number, e.g.: .Query01, .Query02
  - Store in the xxx_Benchmark database, give Select + Grant to app. DB’s
  - Doesn’t work if query is a script with multiple DDL

- **Next Best:** Stored procedures
  - Allows for multiple DDL statements
  - Result queries need to be moved to top as cursors
  - Declare procedure with 1 or more result sets, instead of final query, open cursor and leave open as you exit

- **Also supported:** External scripts
  - Recommend adding /* .Querynnn */ marker after select
  - If multiple steps, use fractions: /* .Querynnn.yyy */
  - Ensure every identifier is unique

- **Zero pad all identifiers so they sort correctly, easier to parse**
  - E.g. .Query01, .Query02 ... .Query10, or .Query01.01, .Query01.02
  - not .Query1, .Query10, .Query11, ... .Query2, .Query20, ...
Testing Models

- **Serial Execution:**
  - First execution can be against empty tables to validate setup and get cross reference of column reference to focus tuning.
  - Provides a baseline of each query’s performance characteristics. Repeat if tuning needed (Skewed, slow, etc).

- **Concurrent Sessions:**
  - Increase # streams until Queries/Hour drop off.

- **Baseline Load:**
  - Export normal unit(s) of update (e.g. a day or several days).
  - Delete normal unit(s) of data.
  - Load (each) normal unit of data.

- **Mixed Workload:**
  - Prepare by deleting unit(s) of data (e.g. a day or days).
  - Start Concurrent test, after a minute, start the load(s) of data.

Serial Execution: First Benchmark Test

- **Serial:**
  - Simple execution of all queries.
  - Will provide best possible execution time: Stand-alone.
  - Pay attention to skewing. Multiple queries with same skew will be much worse when run together.
  - CPU seconds used over duration allows theoretical calculation of how many can run together without slowing (e.g. 33% Cpu => 3).
Avoid Benchmark Constructs That Aren’t Realistic

- Which workload model gives more realistic test? #1?
- Or, #2?
- Which is easier for DBMS workload management? #1 or #2?

Models For Concurrent Workload

- Decay:
  > Start all queries at once and time until last finishes
  > Not realistic but often used because it is simple to set up
  > Weakness: After small queries finish, long running queries run with little concurrency – mostly another serial test
Models For Concurrent Workload

- **Fixed period:**
  > Start # sessions of heavy, medium, light in proportion to real users running queries repeatedly for fixed time period
  > Need to run long enough so longest queries complete

- **Fixed work:**
  > Line up a number of heavy, medium and light queries in proportion to real workload and initiate different #’s of sessions to execute the queries
  > Need to have enough medium/light queries so heavy queries run with concurrency
How Does TdBench Work? Queue Table Usage

- Queue table used to stage work for Worker sessions
- Contributes a few seconds of CPU per hour
- **Overhead?** ... Duration
  - Adds .05 seconds between queries for local PC/Server
  - .4 to .5 seconds from PC in Cincinnati to San Diego.
- **Benefit?** Allows adding workers for more concurrency without needing to change scripts

![Diagram of TdBench queue tables](image)

**TdBench Queue Tables:**

- **QueryQueue**
  - Default queue of queries for worker sessions
  - Workers logon and wait for queries to run
  - Control session populates table from QueryHold to release the test
  - For fixed period tests, last query in QueryQueue repopulates the QueryQueue

- **QueryQueuexxx**
  - Optional to separate queues of work for heavy, medium, light queries, or separate queues for different user types

- **QueryHold**
  - Populated by Control session prior to test
  - Holds queries for all queues
  - Allows quick population or repopulation of queue table(s)
Different Queue Table Usages

- **Serial Execution:**
  > 1 Queue, 1 Worker

- **Concurrent Option #1**
  > 1 Queue, Multiple workers
  > Use for “Fixed Work”
  > For Decay test, use 
  #workers = #queries

- **Concurrent Option #2**
  > Multiple workers, multiple queues
  > Use for “Fixed Period”

Other Test Implementations Supported

- **BTEQ Script**
  > May be simplest conversion of scripts from BI tools
  > Difficult to vary parameters for multiple sessions
  > Can be used to call a different load utility via .os command and return to BTEQ for insert/select from staging to core
  > Can contain .hang command to delay start of load activity
  > Initiated after all of the worker sessions logged on

- **Batch Script** (Windows .bat file)
  > Most flexibility, but also most technical skill to write
  > Passed parameters about RunID, Directory of Log path, etc to support coordination with other test outputs.
TD benchmarks behavior based on type of script

1. `.SQL`
   - Each line is a complete query ending with a semicolon
   - Often an exec of a macro or call to stored procedure
   - All lines will be loaded to a queue table and read by Workers
   - 4000 character/line limit (arbitrary, but must be a single line)

2. `.BTQ`
   - Any valid BTEQ commands or SQL statements
   - Can use .OS command to launch processes (e.g. Fastload)
   - Logon executed by Query Driver
   - Each session runs in its own Window
   - Should exec TestStop at end to help mark end of test.

3. `.Bat`
   - Passed RunID, Log path, Session count, Logon ID, Query Queue,
     Repeat count, Seq# of this session
   - Should run BTEQ at end to execute TestStop to help mark end of test.

---

TD benchmarks behavior based on type of script
(2 of 2) – The .LST file

- Used to run multiple types of scripts or multiple queues
- When used, TDBench only prompts for description of this run and duration
- `.LST` file tokens:
  - 1. Name of a `.SQL`, `.BTQ`, or `.Bat` file. (Specify directory)
  - 2. Number of sessions
  - 3. User logon name pattern
  - 4. Name of the query queue
  - 5. Repeat count (Warning: .Repeat only does next SQL command. E.g. .Run is a BTEQ command, not SQL)

- Example:
  - scripts\New200Hvy.sql 6 Usr_HVY% QueryQueueHvy 1
  - scripts\New200Med.sql 10 Usr_Med% QueryQueueMed 1
  - scripts\New200Lte.sql 24 Usr_Tac% QueryQueueTac 1
Directory Structure For Tdbench

- **TDBenchV5 Directory**
  - Batch files for setup & execution
  - Readme.Doc - Instructions
- **Logs Directory**
  - Run001 Directory
    - Logs for RunID 1
  - Run002 Directory
    - Logs for RunID 2
  - Run ...
- **Queries Directory**
  - Optional, 1 query per file
  - Can be converted to macro
- **Scripts Directory**
  - .SQL, .BTQ, .Bat, and .Lst files
- **Setup Directory**
  - Contains source for install

Windows PC/Server

- **TdBench**
  - Dialog to start benchmark
- **Control**
  - Updates TestTracking, Populates QueryHold, Populates QueryQueues, Cancels sessions
- **Workers**
  - Executes queries

Executing And Monitoring The Query Driver

- Use Windows Server for High Performance tests
  - Queries < 1 second
  - Long running tests where you might get disconnected
- Laptop ok for tests with longer running queries
- Either case, Control session should execute $R (Rush)
- Look for impact of Query Monitor on Test
  - Use Windows task manager to look for high CPU use
  - Calculate interval between queries for a single session
- Watch system via Viewpoint, TdManager or PMon
  - Watch for significant number of idle worker sessions
Benchmark On 2 Systems, Laptop Screen 2580, Monitor For 2650

10 Worker Windows + PMon
Monitoring The Test: Comparison Of 2 Machines

Performance Chart - p4H 7/17/2010

#1: max CPU

#2: max I/O

Watch Test For Causes $< 100\%$ CPU

PMon Graph

Less than $100\%$ CPU

PMon Resource Usage

Skewed I/O (All tables used same default)

Node Problem
Analyzing The Results For A Runid

- Views join captured log information against TestTracking
  - Product join to find rows between start/stop timestamps
  - Use with RunID=n or RunID in (n1, n2, ...)
- Views:
  - **RptBenchSummary** – Number of statements by error code
    - Important for quick validation everything worked ok
  - **RptTestDetail** – execution of each statement within test
    - Use for serial tests
  - **RptTestSummary** – # executions & times by query
    - Use for concurrency tests
  - Resusage reports – select data within bounds of test +/- 2 min
    - **RptSpma** – SPMA records, node level summary per interval
    - **RptSawt** – Amp worker summary information per interval
    - **RptSvpr** – Vproc summary information per interval
    - **RptLogonoff** – all logon/logoff’s within test period

Select * From Testtracking;

<table>
<thead>
<tr>
<th>RunId</th>
<th>TestName</th>
<th>StartTime</th>
<th>ReleaseExecTime</th>
<th>ActualStopTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>run_scripts3</td>
<td>2010-07-17 08:53:38</td>
<td>2010-07-17 08:53:38</td>
<td>2010-07-17 09:20:56</td>
</tr>
<tr>
<td>5</td>
<td>run_scripts5</td>
<td>2010-07-17 09:29:15</td>
<td>2010-07-17 09:29:16</td>
<td>2010-07-17 10:14:16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SessionCnt</th>
<th>RunTitle</th>
<th>TestDescription</th>
<th>RunNotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Baseline the 2650</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>baseline with new aj</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Checkout 2 sessions on 2650</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Concurrent 3 sessions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5 concurrent sessions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10 session concurrent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Rerun 5 sessions with rule set 1: 2a, 2b, 3b, 8a only 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Select * From RptTestDetail Where Runid=2

<table>
<thead>
<tr>
<th>stmt</th>
<th>RunId</th>
<th>TestName</th>
<th>SessionCnt</th>
<th>RunSecs</th>
<th>StartSecs</th>
<th>NumResultRows</th>
<th>TotalIOCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>update testtracking set ReleaseExecTime = current_timestamp where runid=0002;</td>
<td>2</td>
<td>script01</td>
<td>1</td>
<td>0.02</td>
<td>0.13</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Query1a:</td>
<td>2</td>
<td>script01</td>
<td>1</td>
<td>13.94</td>
<td>4.39</td>
<td>2,198,113</td>
<td>2,233,291</td>
</tr>
<tr>
<td>Query1b:</td>
<td>2</td>
<td>script01</td>
<td>1</td>
<td>10.65</td>
<td>19.07</td>
<td>428,170</td>
<td>1,552,523</td>
</tr>
<tr>
<td>Query2a:</td>
<td>2</td>
<td>script01</td>
<td>1</td>
<td>190.33</td>
<td>30.32</td>
<td>105,485</td>
<td>23,226,119</td>
</tr>
<tr>
<td>Query2b:</td>
<td>2</td>
<td>script01</td>
<td>1</td>
<td>73.37</td>
<td>221.11</td>
<td>136,315</td>
<td>14,104,267</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AMPCPU</th>
<th>ParserCPU</th>
<th>CpuTime</th>
<th>SpoolUsage</th>
<th>CpuSkew</th>
<th>IoSkew</th>
<th>ErrorCode</th>
<th>QueryID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.01</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>163691810215365766</td>
</tr>
<tr>
<td>1,085.74</td>
<td>1.32</td>
<td>1,087.06</td>
<td>11,870,501,888</td>
<td>0.06</td>
<td>0.06</td>
<td>0</td>
<td>163691810215365766</td>
</tr>
<tr>
<td>985.33</td>
<td>1.26</td>
<td>986.3</td>
<td>11,144,025,088</td>
<td>0.15</td>
<td>0.19</td>
<td>0</td>
<td>163691810215365769</td>
</tr>
<tr>
<td>24,236.50</td>
<td>0.7</td>
<td>24,237.20</td>
<td>244,722,140,672</td>
<td>0.07</td>
<td>0.09</td>
<td>0</td>
<td>163691810215365770</td>
</tr>
<tr>
<td>5,955.01</td>
<td>1.14</td>
<td>5,956.14</td>
<td>96,679,272,448</td>
<td>0.03</td>
<td>0.02</td>
<td>0</td>
<td>163691810215365771</td>
</tr>
</tbody>
</table>
Select * From RptTestSummary Where Runid = 6

Executed by remote laptop

Select Runid, TheTime, NodeId, CpuUexec, CpuUserv, CpuIoWait, CpuIdle from RptSpma where RunId = 21;

Query Output

Excel Pivot Chart
Benchmark Wrap-up – Save Your Results

- Tables provided in xxx_Benchmark for History
  - Hist_DBQLExplainTbl
  - Hist_DBQLObjTbl
  - Hist_DBQLogTbl
  - Hist_DBQLSqlTbl
  - Hist_DBQStepTbl
  - Hist_EventLog
  - Hist_ResUsageSawt
  - Hist_ResUsageSpma
  - Hist_ResUsageSvpr

- Stored procedures supporting history:
  - HistSetCopyRuns
    - Copies all new data from DBC within RunID’s
  - HistSetCopyAll
    - Copies all new data from DBC
  - HistSetHist
    - Changes views in xxx_Benchmark to point to history tables in xxx_Benchmark
  - HistSetDBC
    - Changes views in xxx_Benchmark to point to active tables in DBC.

Concluding Thoughts “Building Your Own Benchmark Center”

- “If you can not measure it, you can not improve it.”
  Lord Kelvin, Mathematician, Physicist, 1824 - 1907

- “You don’t have to be a rocket scientist to do analysis. Find a simple tool and use it to the extent you need it.”
  Randall Parman, Swamped DBA

- There are two possible outcomes: if the result confirms the hypothesis, then you’ve made a measurement. If the result is contrary to the hypothesis, then you’ve made a discovery.
  Enrico Fermi, Nuclear Physicist, 1901 - 1954

- “If you don’t measure the right things, decisions may be wrong, you’ll have no basis for finding problems, and worse yet, if improvements happen, they will go unrecognized.”
  Doug Ebel, Database Geek, 1950 - 20??