Benchmarking Using Basic DBMS Operations

Alain Crolotte and Ahmad Ghazal
Teradata Corporation

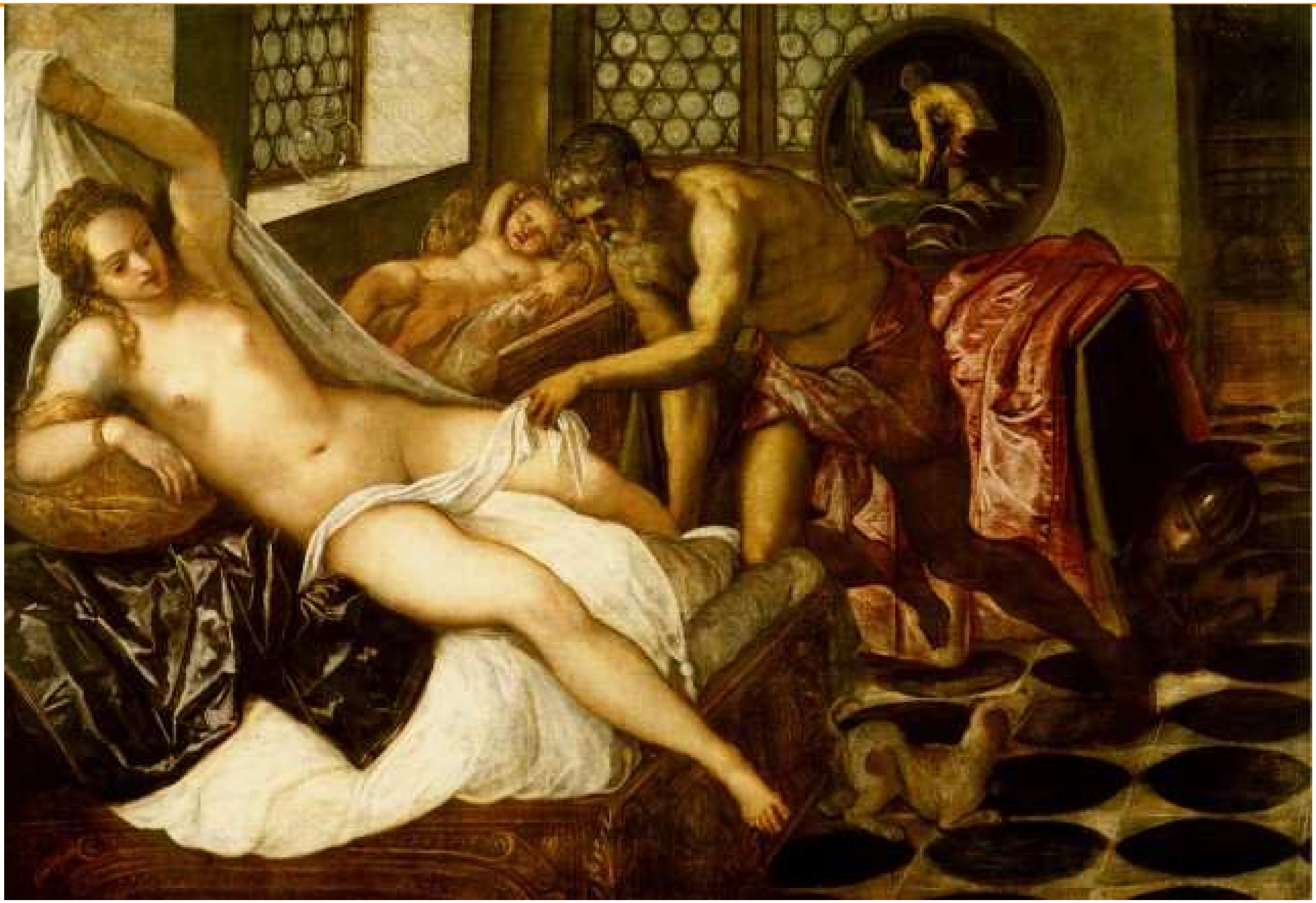
TPCTC 2010
Singapore

Raising Intelligence

Outline

- Background
- Approach
- Benchmark Description
- Example
- Conclusion

Mars and Venus Surprised by Vulcan
Painting by Tintoretto



Background

- TPC-H is a successful benchmark with a well understood schema and an excellent data generator, but the benchmark is showing its age
- Because of its age many new fancy techniques go around TPC-H strict rules by sliding “under the covers”
- What techniques are used can be inferred only though painful investigation
- TPC-H metrics not reflective of true ad-hoc performance
- Problem: How do you construct a benchmark allowing evaluation of hardware through the DBMS layer that is simple and yet interesting?

Approach

- Identify main DBMS areas:
  - Scans
  - Aggregations
  - Joins
  - CPU Intensive
  - Indexes
- Build on existing structures:
  - TPC-H schema
  - data generator (dbgen)
- Design queries exercising only specific individual sub-functions in the main DBMS areas. (For instance a scan query should scan the entire table but yet return no or very few rows.)

TERADATA
Raising Intelligence

Benchmark Description - XMarq

- Xmarq consists of basic SQL queries:
  - > Scan Queries
  - > Aggregate Queries
  - > Join Queries
  - > CPU Intensive Queries
  - > Indexed Queries

SCAN QUERIES

- ST - LARGE TABLE SCAN

  Retrieve all fields from lineitem using a condition unsatisfied or returning very few rows:

  SELECT * FROM LINEITEM WHERE L_LINENUMBER < 0;

  SELECT * FROM LINEITEM WHERE L_RECEIPTDATE < 920110; (another possibility)

- SI - MATCHING INSERT/SELECT

  Insert into an empty table PARTX with DDL identical to PART (distributed by p_partkey)

  INSERT INTO PARTX SELECT * FROM PART;

- SN - NON-MATCHING INSERT/SELECT

  Same as matching insert/select but PARTX is distributed differently from PART e.g. (p_size ,p_brand, p_container) or a more unique combination of fields. On MPP systems PARTX rows will be located on different nodes resulting in a different performance. The SQL is the same.

TERADATA
Raising Intelligence

SCAN QUERIES (cont.)

- SU - SCAN/UPDATE 4% OF THE ROWS

This query utilizes the fact that column p_brand in PARTX has only 25 distinct vales so that, due to the uniform distribution of data produced by dbgen, any specific value of p_brand will be represented in about 4% of the rows.

```
UPDATE PARTX SET P_RETAILPRICE = (P_RETAILPRICE + 1) WHERE P_BRAND =
'Brand#23';
```

- SP - SCAN/UPDATE 20% OF THE ROWS

This query utilizes the fact that p_mfgr has only 5 distinct values. As a result a particular value of p_mfgr will involve about 20% of the rows. At scale factor 1000 exactly 40,006,935 rows out of 200 millions are updated.

```
UPDATE PARTX SET P_RETAILPRICE = (P_RETAILPRICE + 1) WHERE P_MFGR
='Manufacturer#5';
```

TERADATA
Raising Intelligence

AGGREGATE QUERIES

- AR – ROW COUNT

  This query does a simple count of the rows in the largest table in the database, LINEITEM. The number of rows returned is well-documented in the TPC-H spec. Some products use a simple file system lookup or can take advantage of an index for this query

```
SELECT COUNT(*) FROM LINEITEM;
```

- AD – DISTINCT COUNT

  This query counts the distinct values of the column l_quantity in the LINEITEM table. A single row with a value of 50 should be returned at all scale factors and l_quantity must not be indexed for this query.

```
SELECT COUNT (DISTINCT L_QUANTITY) FROM LINEITEM;
```

- AS – 15-GROUP AGGREGATE

  This query utilizes ORDERS and the fact that the combination o_ordertatus, o_orderpriority has only 15 distinct combinations at all scale factors.

```
 SELECT O_ORDERSTATUS, O_ORDERPRIORITY, AVERAGE (O_TOTALPRICE FROM ORDERS
GROUP BY 1, 2;
```

TERADATA
Raising Intelligence

AGGREGATE QUERIES (cont.)

- AM – THOUSAND GROUP AGGREGATE

  This query uses LINEITEM and l_receipdate that has only a limited number of values (2555 at scale factor 1000). While l_shipdate is more predictable (exactly 2406 distinct values at all scale factors) this field plays too central of a role in the TPC-H queries. No index should be placed on l_receiptdate for this query.

  SELECT L_RECEIPTDATE, COUNT (*) FROM LINEITEM GROUP BY 1 ORDER BY 1;

- AL – HUNDRED THOUSAND GROUP AGGREGATE

  This query is meant to build over 100000 aggregate groups. By using the first 15 characters of o_comment one can build exactly 111517 groups at scale factor 1000. To further limit the number of rows actually retrieved we added a limit on o_totalprice.

  SELECT SUBSTRING (O_COMMENT,1,15), ,COUNT(*) FROM ORDERS GROUP BY 1;

TERADATA
Raising Intelligence

JOIN QUERIES

- **JI – IN-PLACE JOIN**

  In this query we join ORDERS and LINEITEM on the key common to both tables without constraints while performing a calculation ensuring that the join is performed but only one row is returned.

  SELECT AVERAGE (L_QUANTITY) FROM LINEITEM, ORDERS WHERE L_ORDERKEY = O_ORDERKEY;

- **JF – PK/FK JOIN**

  This query joins PART and LINEITEM on partkey which is the PK for PART and the FK for LINEITEM while performing an operation involving columns in both tables. No index on l_partkey is allowed for this query.

  SELECT AVERAGE (P_RETAILPRICE*L_QUANTITY) FROM PART, LINEITEM WHERE P_PARTKEY = L_PARTKEY;

- **JA – AD-HOC JOIN**

  This query joins PART and LINEITEM on unrelated columns (p_partkey and l_suppkey) while performing a sum so that only one row is returned. Because of the fact that the join columns are sequential integers there will be some matching.

  SELECT SUM(L_QUANTITY) FROM PART, LINEITEM WHERE P_PARTKEY=L_SUPPKEY;

JOIN QUERIES (cont.)

- JL – LARGE/SMALL JOIN

This query joins CUSTOMER and NATION on nationkey while performing a group by operation. This is also an FK/PK join but the salient feature here is the size discrepancy between the tables since NATION has only 25 rows.

SELECT N_NAME, AVERAGE(C_ACCTBAL) FROM CUSTOMER, NATION WHERE C_NATIONKEY=N_NATIONKEY GROUP BY N_NAME;

- JX – EXCLUSION JOIN

This query calculates the total account balance for the one-third of customers without orders. For those customers the CUSTOMER rows do not have a customer key entry in the ORDER table. The inner query selects customers that do have orders so that by leaving these out we get the customers that do not have orders.

SELECT SUM(C_ACCTBAL) FROM CUSTOMER WHERE C_CUSTKEY NOT IN (SELECT O_CUSTKEY FROM ORDERTBL);

CPU INTENSIVE QUERIES

- **CR – ROLLUP REPORT**

This query covers the group by operator. To accomplish that, the query is applied on a single table with a simple predicate. The query involves 7 columns and 12 aggregations.

```
SELECT L_RETURNFLAG, L_LINESTATUS, L_SHIPMODE, SUBSTRING (L_SHIPINSTRUCT, 1, 1),
SUBSTRING (L_LINESTATUS, 1, 1), ((L_QUANTITY - L_LINENUMBER) + (L_LINENUMBER -
 L_QUANTITY)),
 (L_EXTENDEDPRICE - L_EXTENDEDPRICE), SUM ((1 + L_TAX) * L_EXTENDEDPRICE),
 SUM ((1 - L_DISCOUNT) * L_EXTENDEDPRICE), SUM (L_DISCOUNT / 3),
 SUM (L_EXTENDEDPRICE * (1 - L_DISCOUNT) * (1 + L_TAX)),
 SUM (L_EXTENDEDPRICE - ((1 - L_DISCOUNT) * L_EXTENDEDPRICE)),
 SUM (DATE - L_SHIPDATE + 5), SUM (L_SHIPDATE - L_COMMITDATE),
 SUM (L_RECEIPTDATE - L_SHIPDATE), SUM (L_LINENUMBER + 15 - 14),
 SUM (L_EXTENDEDPRICE / (10 - L_TAX)), SUM ((L_QUANTITY * 2) / (L_LINENUMBER * 3)),
 COUNT (*)
 FROM LINEITEM
 WHERE L_LINENUMBER GT 2
 GROUP BY  L_RETURNFLAG,  L_LINESTATUS,  L_SHIPMODE,
 SUBSTRING (L_SHIPINSTRUCT,1,1), SUBSTRING (L_LINESTATUS,1,1),
 ((L_QUANTITY - L_LINENUMBER) + (L_LINENUMBER - L_QUANTITY)),
 (L_EXTENDEDPRICE - L_EXTENDEDPRICE);
```

CPU INTENSIVE QUERIES (cont.)

- CF – FLOATS & DATES

This query maximizes CPU activity while minimizing disk or interconnect overhead. Floating point conversion is done multiple times for each of the rows, as well as repetitive complex date conversions.

```
SELECT COUNT(*) FROM LINEITEM
WHERE (L_QUANTITY = 1.1E4
OR L_QUANTITY = 2.1E4
OR L_QUANTITY = 3.1E4
OR L_QUANTITY = 4.1E4
OR L_QUANTITY = 5.1E4
OR L_QUANTITY = 6.1E4
OR L_QUANTITY = 7.1E4
OR L_QUANTITY = 8.1E4
OR L_QUANTITY = 9.1E4
OR L_QUANTITY = 50)
AND (DATE - L_SHIPDATE) GT 0
AND (L_COMMITDATE + 5) LT (L_RECEIPTDATE + 5)
AND (L_SHIPDATE + 20) LT (L_COMMITDATE + 20);
```

TERADATA
Raising Intelligence

INDEX QUERIES

- **IP – PRIMARY RANGE SEARCH**

  This query measures the ability of database system to select from a table based on a range of values applied to the table's primary (or clustering) index. In this case the range constraint is on the same column as the partitioning key (clustering, primary index). It is designed to highlight capabilities such as value-ordered or value-sensitive indexes.

  SELECT P_NAME, P_RETAILPRICE FROM PART WHERE P_PARTKEY LT 50000 AND P_RETAILPRICE LT 909.00;

- **IR – SECONDARY RANGE SEARCH**

  In this query, the range constraint column is not the same column as the partitioning key. Secondary index access usually requires traversing a secondary structure which points to the location of the physical rows required. An index must be placed on l_shipdate for this query or l_receiptdate can be used as an alternate.

  SELECT L_ORDERKEY, L_LINENUMBER FROM LINEITEM WHERE L_SHIPDATE LT 981200;

- **IL – LIKE OPERATOR**

  This query uses the text string search capabilities of the LIKE clause, searching for all rows that have a value in their O_CLERK column that begin with that string. This query returns 100 rows at all scale factors.

  SELECT DISTINCT O_CLERK FROM ORDERS WHERE O_CLERK LIKE 'Clerk#0000067%';

TERADATA Raising Intelligence

INDEX QUERIES (cont.)

- IB – BETWEEN OPERATOR

  This query uses yet another type of constraint with an index.

  SELECT DISTINCT L_QUANTITY FROM LINEITEM WHERE L_SHIPDATE BETWEEN 930301 AND 930331;

- II – MULTIPLE INDEX ACCESS

  This query combines both primary and secondary index searches using the OR operator between two constraints: one constraint is on a non-unique index on the Clerk column and the second is on a unique index o_orderkey.

  SELECT COUNT (*) FROM ORDERS WHERE O_CLERK = 'CLERK#000006700' OR O_ORDERKEY = 50500;

- IC – COUNT BY INDEX

  This query counts the distinct discounts via an index on l_discount.

  SELECT L_DISCOUNT, COUNT (*) FROM LINEITEM GROUP BY L_DISCOUNT;

- IM – MULTI-COLUMN INDEX -- LEADING VALUE ONLY

  In this query we have a 2-column index on p_type and p_size but the condition is on p_type only.

  SELECT AVERAGE (P_RETAILPRICE) FROM PART WHERE P_TYPE = 'SMALL PLATED BRASS';

- IT – MULTI-COLUMN INDEX -- TRAILING VALUE ONLY

  A somewhat related need is measured by this query, which provides a value for the second, or trailing value in a 2-column index. Like the previous query the ability to use an index in a way broader than anticipated is test. As was the case for IM a single row is returned

  SELECT AVERAGE (P_RETAILPRICE) FROM PART WHERE P_SIZE = 21;

TERADATA Raising Intelligence

Example – 2 Systems with same Teradata software version B has 4 times as many CPUs as A CPUs and slightly better I/O subsystem (scale factor 1000). The single system metric is the simple mean while the comparison metric is based on the geometric mean.

QUERIES/CATEGORIES	System A	System B	A/B
ST -- LARGE TABLE SCAN	95.9	18.6	5.1
SI -- MATCHING INSERT/SELECT	48.0	8.7	5.5
SN -- NON-MATCHING INSERT/SELECT	174.6	25.0	7.0
SU -- SCAN/UPATE 4% OF THE ROWS	191.5	58.4	3.3
SP -- SCAN/UPATE 20% OF THE ROWS	906.3	244.1	3.7
SCANS	**283.2**	**71.0**	**4.8**
AR -- ROW COUNT	0.8	0.3	3.0
AD -- COUNT DISTINCT VALUES	112.3	29.8	3.8
AS -- 15-GROUP AGGREGATE	51.5	14.4	3.6
AM -- THOUSAND-GROUP AGGREGATE	30.2	9.7	3.1
AL -- HUNDRED-THOUSAND GROUP AGGREGATION	61.1	16.6	3.7
AGGREGATIONS	**51.1**	**14.2**	**3.4**
JI -- IN PLACE JOIN	372.8	123.3	3.0
JF -- FOREIGN KEY TO PRIMARY KEY JOIN	2441.5	340.5	7.2
JA -- AD HOC JOIN	891.9	191.9	4.6
JL -- LARGE/SMALL JOIN	0.1	0.0	11.0
JX -- EXCLUSION JOIN	21.3	19.6	1.1
JOINS	**745.5**	**135.1**	**3.2**

Geometric mean for the category

Simple Average for the category

TERADATA
Raising Intelligence

Example (cont.)

CR -- ROLLUP REPORT	312.3	75.2	4.2
CF -- FLOATS & DATES	96.4	16.5	5.9
CPU INTENSIVE	**204.3**	**45.8**	**4.9**
IP -- PRIMARY RANGE SEARCH	3.1	1.0	3.3
IR -- SECONDARY RANGE SEARCH	10.7	0.3	42.7
IL -- LIKE OPERATOR	29.0	7.7	3.8
IB -- BETWEEN OPERATOR	14.1	0.6	22.1
II -- MULTIPLE INDEX ACCESS	29.0	7.5	3.9
IC -- COUNT BY INDEX	6.5	1.8	3.6
IM -- MULTI-COLUMN INDEX -- LEADING VALUE ONLY	1.9	0.3	5.6
IT -- MULTI-COLUMN INDEX -- TRAILING VALUE ONLY	1.7	0.2	7.3
INDEXED OPERATIONS	**12.0**	**2.4**	**7.1**
overall scores	***236.2***	***48.5***	***4.9***

Simple average of the categories

Geometric mean of the categories

Conclusion

- XMarq meets all of Karl Huppler's criteria for a "good" benchmark

  (Huppler, K. "The Art of Building a Good Benchmark". In Performance Evaluation and Benchmarking: First TPC Technology Conference, TPCTC 2009, Lyon, France)
  - It is *relevant* since it uses all usual paths important in DBMS technology
  - It is *repeatable* since it uses the TPC-H schema
  - It is *fair* because it is simple and does not favor "fancy" optimizers
  - Since it uses existing structures (TPC-H schema, dbgen, subset of TPC-H SQL) it is both *verifiable* and *economical*