TPC-H Analyzed

Hidden Messages and Lessons Learned from an Influential Benchmark

Peter Boncz (CWI)
Thomas Neumann (TUM)
Orri Erling (Openlink Systems)

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

Why Read This Paper

- “TPC-H cheat sheet for DBMS architects”
  - based on years of experience of three database system design lead architects, who have optimized their systems for TPC-H
  - in-depth explanation of 28 crucial challenges in the benchmark, with pointers to address these
- Inspire a benchmark design methodology
  - “choke point” based

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

Database Benchmark Design

Desirable properties:

- Relevant.
- Representative.
- Understandable.
- Economical.
- Accepted.
- Scalable.
- Portable.
- Fair.
- Evolvable.
- Public.

Jim Gray (1991) *The Benchmark Handbook for Database and Transaction Processing Systems*

Dina Bitton, David J. DeWitt, Carolyn Turbyfill (1993) *Benchmarking Database Systems: A Systematic Approach*

Multiple TPCTC papers, e.g.:
Karl Huppler (2009) *The Art of Building a Good Benchmark*

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

Stimulating Technical Progress

- An aspect of ‘Relevant’
- The benchmark metric
  - depends on,
  - or, rewards:

solving certain
technical challenges

(not commonly solved by technology at benchmark design time)

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

Benchmark Design with Choke Points

Choke-Point = well-chosen difficulty in the workload

- "difficulties in the workloads"
  - arise from Data (distribs)+Query+Workload
  - there may be different technical solutions to address the choke point
    - or, there may not yet exist optimizations (but should not be NP hard to do so)
    - the impact of the choke point may differ among systems

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

Benchmark Design with Choke Points

Choke-Point = well-chosen difficulty in the workload

- “difficulties in the workloads”
- “well-chosen”
  - the majority of actual systems do not handle the choke point very well
  - the choke point occurs or is likely to occur in actual or near-future workloads

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

This Paper: TPC-H choke points

- Even though TPC-D was designed without specific choke point analysis
  - more informal SQL query contribution process
- It contains a whole lot of them!
  - many more than SSB
  - considerably more than XMark
  - not sure about TPC-DS (yet)

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

TPC-H choke point areas (1/3)

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22
CP1 Aggregation Performance. Performance of aggregate calculations.	■	▒	■	■	▒	▒	▒	▒	▒	■	▒	▒	■	▒	▒	▒	▒	■	▒	■	■	▒
CP2 Join Performance. Voluminous joins, with or without selections.		▒	▒	▒	▒		▒	▒	■	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒	▒
CP3 Data Access Locality. Non-full-scan access to (correlated) table data.			■	■	■	▒	■	■		■		■		▒	▒					▒		

CP1 Aggregation Performance. Performance of aggregate calculations.

CP1.1 QEXE: Ordered Aggregation.
CP1.2 QOPT: Interesting Orders.
CP1.3 QOPT: Small Group-by Keys (array lookup).
CP1.4 QEXE: Dependent Group-By Keys (removal of).

CP2 Join Performance. Voluminous joins, with or without selections.

CP2.1 QEXE: Large Joins (out-of-core).
CP2.2 QEXE: Sparse Foreign Key Joins (bloom filters).
CP2.3 QOPT: Rich Join Order Optimization.
CP2.4 QOPT: Late Projection (column stores).

CP3 Data Access Locality. Non-full-scan access to (correlated) table data.

CP3.1 STORAGE: Columnar Locality (favors column storage).
CP3.2 STORAGE: Physical Locality by Key (clustered index, partitioning).
CP3.3 QOPT: Detecting Correlation (ZoneMap,MinMax,multi-attribute histograms).

TPC-H choke point areas (2/3)

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22

CP4 Expression Calculation. Efficiency in evaluating (complex) expressions.

CP4.1 Raw Expression Arithmetic.
CP4.1a QEXE: Arithmetic Operation Performance.
CP4.1b QEXE: Overflow Handling (in arithmetic operations).
CP4.1c QEXE: Compressed Execution.
CP4.1d QEXE: Interpreter Overhead (vectorization; CPU/GPU/FPGA JIT compil.).

CP4.2 Complex Boolean Expressions in Joins and Selections.
CP4.2a QOPT: Common Subexpression Elimination (CSE).
CP4.2b QOPT: Join-Dependent Expression Filter Pushdown.
CP4.2c QOPT: Large IN Clauses (invisible join).
CP4.2d QEXE: Evaluation Order in Conjunctions and Disjunctions.

CP4.3 String Matching Performance.
CP4.3a QOPT: Rewrite LIKE(X%) into a Range Query.
CP4.3b QEXE: Raw String Matching Performance (e.g. using SSE4.2).
CP4.3c QEXE: Regular Expression Compilation (JIT/FSA generation).

TPC-H choke point areas (3/3)

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22

CP5 Correlated Subqueries. Efficiently handling dependent subqueries.

CP5.1 QOPT: Flattening Subqueries (into join plans).
CP5.2 QOPT: Moving Predicates into a Subquery.
CP5.3 QEXE: Overlap between Outer- and Subquery.

CP6 Parallelism and Concurrency. Making use of parallel computing resources.

CP6.1 QOPT: Query Plan Parallelization.
CP6.2 QEXE: Workload Management.
CP6.3 QEXE: Result Re-use.

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

CP1.4 Dependent GroupBy Keys

Q10

```
SELECT c_custkey,  c_name, c_acctbal,
   sum(l_extendedprice * (1 - l_discount)) as revenue,
   n_name,  c_address,  c_phone, c_comment
FROM  customer, orders,  lineitem,  nation
WHERE  c_custkey = o_custkey and l_orderkey = o_orderkey
   and o_orderdate >= date '[DATE]'
   and o_orderdate < date '[DATE]' + interval '3' month
   and l_returnflag = 'R` and c_nationkey = n_nationkey
GROUP BY
   c_custkey, c_name,   c_acctbal,  c_phone,  n_name,
   c_address, c_comment
ORDER BY revenue DESC
```

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

CP1.4 Dependent GroupBy Keys

Q10

```
SELECT c_custkey,  c_name, c_acctbal,
   sum(l_extendedprice * (1 - l_discount)) as revenue,
   n_name,  c_address,  c_phone, c_comment
FROM  customer, orders,  lineitem,  nation
WHERE  c_custkey = o_custkey and l_orderkey = o_orderkey
   and o_orderdate >= date '[DATE]'
   and o_orderdate < date '[DATE]' + interval '3' month
   and l_returnflag = 'R` and c_nationkey = n_nationkey
GROUP BY
   c_custkey, c_name,   c_acctbal,  c_phone,
   c_address, c_comment, n_name
ORDER BY revenue DESC
```

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

CP1.4 Dependent GroupBy Keys

- Functional dependencies:

  `c_custkey → c_name, c_acctbal, c_phone, c_address, c_comment, c_nationkey → n_name`

- Group-by hash table should exclude the colored attrs → less CPU+ mem footprint
- in TPC-H, one can choose to declare primary and foreign keys (all or nothing)
  - this optimization requires declared keys
  - Key checking slows down RF (insert/delete)

Exasol:
"foreign key check" phase after load

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

CP2.2 Sparse Joins

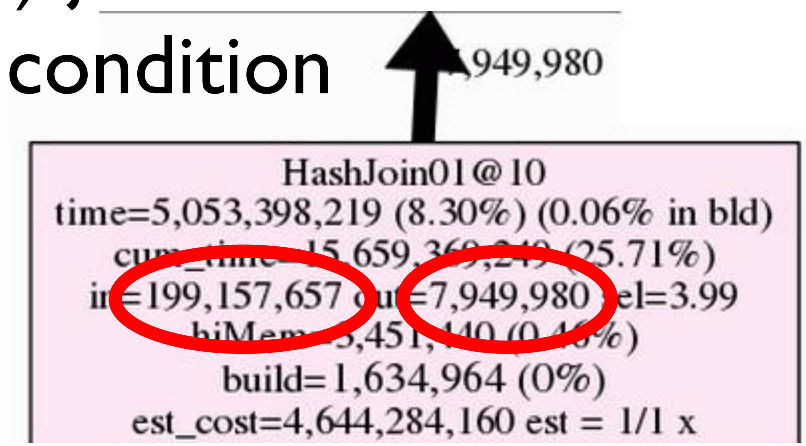
- Foreign key (N:1) joins towards a relation with a selection condition
  - Most tuples will *not* find a match
  - Probing (index, hash) is the most expensive activity in TPC-H
- Can we do better?
  - Bloom filters!

CP2.2 Sparse Joins

- Foreign key (N:1) joins towards a relation with a selection condition

Q21

probed: 200M tuples
result: 8M tuples
→ 1:25 join hit ratio



Vectorwise:
TPC-H joins typically accelerate 4x
Queries accelerate 2x

2G cycles 29M probes → cost would have been 14G cycles ~= 7 sec

```
#PROF 2021162220   OWN 28950172  9.8avg rdtsc  307565 calls vht_lookup_keys() "vht_lookup_keys" in con
#PROF 1575739535   OWN 199097581  7.9avg rdtsc  307534 calls sel_bitfiltercheck_uchr_col_slng_val_sint_
```

1.5G cycles 200M probes → 85% eliminated

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

CP3.2 Physical Locality By Key

- most frequent selection in TPC-H is range predicate between date columns
- there is correlation between these

```
l_shipdate = o_orderdate + random[1:121]
l_commitdate = o_orderdate + random[30:90]
l_receiptdate = l_shipdate + random[1:30]
```

- techniques to use:
  - clustered index
  - partitioned table (by range)

CP3.2 Physical Locality By Key

- can the optimizer derive a range on l_commitdate from l_shipdate?
  - supposing a clustered index on l_shipdate
  - ➔ e.g. Zone Maps, MinMax indices, Small Materialized Aggregates
- can the optimizer derive a range on o_orderdate from l_shipdate?

Q3

```
SELECT l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue,
   o_orderdate, , o_shippriority
FROM customer, orders,  lineitem
WHERE
   c_mktsegment = '[SEGMENT]` and c_custkey = o_custkey
   and l_orderkey = o_orderkey
   and o_orderdate < date '[DATE]`
   and l_shipdate > date '[DATE]'
GROUP BY l_orderkey,  o_orderdate,  o_shippriority
ORDER BY revenue DESC  o_orderdate;
```

Microsoft SQLserver magic flag
DATE_CORRELATION_OPTIMIZATION

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

CP4.1 Raw Expression Arithmetic

How fast is a query processor in computing, e.g.

- Numerical Arithmetic
- Aggregates
- String Matching

Q1

```
SELECT
   l_returnflag, l_linestatus, count(*),
   sum(l_quantity),sum(l_extendedprice),
   sum(l_extendedprice*(1-l_discount)),
   sum(l_extendedprice*(1-l_discount)*(1+l_tax)),
   avg(l_quantity),avg(l_extendedprice),avg(l_discount),
FROM lineitem
```

SIMD? Interpreter Overhead?
Vectorwise, Virtuoso, SQLserver cstore ➔ vectorized execution
Hyper, Netteza, ParAccel ➔ JIT query compilation
Kickfire, ParStream ➔ hardware compilation (FPGA/GPU)

CP5.2 Subquery Rewrite

Q17

```
SELECT sum(l_extendedprice) / 7.0 as avg_yearly
FROM lineitem,  part
WHERE p_partkey = l_partkey
   and p_brand = '[BRAND]'
   and p_container = '[CONTAINER]'
   and l_quantity <( SELECT 0.2 * avg(l_quantity)
                     FROM lineitem
                     WHERE l_partkey = p_partkey)
```

This subquery can be extended with restrictions from the outer query.

Hyper:
CP5.1+CP5.2+CP5.3
results in 500x faster
Q17

```
SELECT 0.2 * avg(l_quantity)
FROM lineitem
WHERE l_partkey = p_partkey
   and p_brand = '[BRAND]'
   and p_container = '[CONTAINER]'
```

+ CP5.3 Overlap between Outer- and Subquery.

CP6.3: Re-Use

- For the Throughput score
  - RF del/ins streams may be run in advance
  - Subsequently, concurrent query streams
    - Read-only system state
    - Limited # parameter bindings

      ➔ Duplicate queries, Overlapping queries

Query Result Caching Opportunity
Oracle ➔ previous runs used a query cache
MonetDB ➔ Recycling, partial query re-use

TPC does not tolerate query caching options/directives

Conclusion

- Choke Points: a concept in Benchmark Design
  - trying to create relevant queries
  - instrument to steer towards certain breakthroughs
- Full Analysis for TPC-H
  - "cheat sheet" for improving systems on TPC-H
  - 28 choke points
    - have influenced many systems

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf

Thanks! / Questions?

Peter Boncz (CWI)
Thomas Neumann (TUM)
Orri Erling (Openlink Systems)

www.cwi.nl/~boncz/tpctc2013_boncz_neumann_erling.pdf