Industry-strength benchmarks for Graph and RDF Data Management

Peter Boncz
Why Benchmarking?

- make competing products comparable
- accelerate progress, make technology viable

- TPC price/perf trend 1990-2005: improved 58% per year, prices have declined 37%/y

© Jim Gray, 2005
What is the LDBC?

**Linked Data Benchmark Council = LDBC**

- Industry entity similar to TPC ([www.tpc.org](http://www.tpc.org))
- Focusing on graph and RDF store benchmarking

Kick-started by an EU project

- Runs from September 2012 – March 2015
- 9 project partners:

  ![Project Partner Logos]

- Will continue independently after the EU project
LDBC Benchmark Design

Developed by so-called “task forces”

- Requirements analysis and use case selection.
  - Technical User Community (TUC)
- Benchmark specification.
  - data generator
  - query workload
  - metrics
  - reporting format
- Benchmark implementation.
  - tools (query drivers, data generation, validation)
  - test evaluations
- Auditing
  - auditing guide
  - auditor training
LDBC: what systems?

Benchmarks for:

- RDF stores (SPARQL speaking)
  - Virtuoso, OWLIM, BigData, Allegrograph, …

- Graph Database systems
  - Neo4j, DEX, InfiniteGraph, …

- Graph Programming Frameworks
  - Giraph, Green Marl, Grappa, GraphLab, …

- Relational Database systems
LDBC: functionality

Benchmarks for:

- Transactional updates in (RDF) graphs
- Business Intelligence queries over graphs
- Graph Analytics (e.g. graph clustering)
- Complex RDF workload, e.g. including reasoning, or for data integration

Anything relevant for RDF and graph data management systems
Roadmap for the Keynote

Choke-point based benchmark design

- What are Choke-points?
  - examples from good-old TPC-H
  - relational database benchmarking

- A Graph benchmark Choke-Point, in-depth:
  - Structural Correlation in Graphs
  - and what we do about it in LDBC

- Wrap up
Database Benchmark Design

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


Multiple TPCTC papers, e.g.:
Karl Huppler (2009) *The Art of Building a Good Benchmark*
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)
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
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
Example: TPC-H choke points

- Even though it was designed without specific choke point analysis
- TPC-H contained a lot of interesting challenges
  - many more than Star Schema Benchmark
  - considerably more than Xmark (XML DB benchmark)
  - not sure about TPC-DS (yet)

“TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark”
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

“TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark”
TPC-H choke point areas (2/3)
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
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
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

“TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark”
CP1.4 Dependent GroupBy Keys

- Functional dependencies:
  
  \[
  c_{\text{custkey}} \rightarrow c_{\text{name}}, c_{\text{acctbal}}, c_{\text{phone}},
  c_{\text{address}}, c_{\text{comment}}, c_{\text{nationkey}} \rightarrow n_{\text{name}}
  \]

- Group-by hash table should exclude the colored attrs \(\rightarrow\) 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
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!

“TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark”
CP2.2 Sparse Joins

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

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

  - 1.5G cycles    200M probes
  - 85% eliminated

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

Vectorwise:
- TPC-H joins typically accelerate 4x
- Queries accelerate 2x
**CP5.2 Subquery Rewrite**

Q17

```sql
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

```sql
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.
Choke Points

- Hidden challenges in a benchmark
  - Influence database system design, e.g. TPC-H
    - Functional Dependency Analysis in aggregation
    - Bloom Filters for sparse joins
    - Subquery predicate propagation

- LDBC explicitly designs benchmarks looking at choke-point “coverage”
  - Requires access to database kernel architects
Roadmap for the Keynote

**Choke-point** based benchmark design

- What are Choke-points?
  - examples from good-old TPC-H

- Graph benchmark Choke-Point, in-depth:
  - *Structural Correlation in Graphs*
  - and what we do about it in LDBC

- Wrap up
Data correlations between attributes

SELECT personID from person
WHERE firstName = 'Joachim' AND addressCountry = 'Germany'

SELECT personID from person
WHERE firstName = 'Cesare' AND addressCountry = 'Italy'

- Query optimizers may underestimate or overestimate the result size of conjunctive predicates
SELECT COUNT(*)
FROM paper pa1 JOIN conferences cn1 ON pa1.journal = jn1.ID
  paper pa2 JOIN conferences cn2 ON pa2.journal = jn2.ID
WHERE pa1.author = pa2.author AND
  cn1.name = 'VLDB' AND cn2.name = 'SIGMOD'

Data correlations **between attributes**
Data correlations **over joins**

SELECT COUNT(*)
FROM paper pa1 JOIN conferences cn1 ON pa1.journal = cn1.ID
  paper pa2 JOIN conferences cn2 ON pa2.journal = cn2.ID
WHERE pa1.author = pa2.author AND
  cn1.name = 'VLDB' AND cn2.name = 'SIGMOD'

- A challenge to the optimizers to adjust estimated join hit ratio
  pa1.author = pa2.author
  depending on other predicates

**Correlated predicates are still a frontier area in database research**
LDBC Social Network Benchmark (SNB)
Handling Correlation: a choke point for Graph DBs

- What makes graphs interesting are the connectivity patterns
  - who is connected to who?
  - structure typically depends on the (values) attributes of nodes
- Structural Correlation (choke point)
  - amount of common friends
  - shortest path between two persons
  - search complexity in a social network varies wildly between
    - two random persons
    - e.g. colleagues at the same company
- No existing graph benchmark specifically tests for the effects of correlations
- Synthetic graphs used for benchmarking do not have structural correlations

Need a data generator generating synthetic graph with data/structure correlations

“S3G2: A Scalable Structure-correlated Social Graph Generator”
Generating **Correlated** Property Values

- How do data generators generate values?  
  E.g. FirstName

TPCTC 2012:  
“S3G2: A Scalable Structure-correlated Social Graph Generator”
Generating Property Values

- How do data generators generate values?  E.g. FirstName

- **Value** Dictionary \(D()\)
  - a fixed set of values, e.g.,
    
    \{“Andrea”, “Anna”, “Cesare”, “Camilla”, “Duc”, “Joachim”, .. \}

- **Probability** density function \(F()\)
  - steers how the generator chooses values
    - cumulative distribution over dictionary entries determines which value to pick
  - could be anything: uniform, binomial, geometric, etc...
    - geometric (discrete exponential) seems to explain many natural phenomena

“S3G2: A Scalable Structure-correlated Social Graph Generator”
Generating Correlated Property Values

- How do data generators generate values? E.g. FirstName

- **Value** Dictionary \( D() \)

- **Probability** density function \( F() \)

- **Ranking** Function \( R() \)
  - Gives each value a unique rank between one and \(|D|\)
    - determines which value gets which probability
  - Depends on some parameters (parameterized function)
    - value frequency distribution becomes correlated by the parameters or \( R() \)

“S3G2: A Scalable Structure-correlated Social Graph Generator”
Generating **Correlated** Property Values

How do data generators generate values? E.g. **FirstName**

- **Value Dictionary**

- **Probability density function**
  geometric distribution

- **Ranking Function**
  \[ R(\text{gender}, \text{country}, \text{birthyear}) \]
  \begin{align*}
  \cdot \text{gender, country, birthyear} & \Rightarrow \text{correlation parameters}
  \end{align*}

**Solution:**
- Just store the rank of the top-N values, not all |D|
- Assign the rank of the other dictionary values randomly
Compact Correlated Property Value Generation

Using geometric distribution for function $F()$
Correlated Value Property in LDBC SNB

- Main source of dictionary values from DBpedia (http://dbpedia.org)

- Various realistic property value correlations (⇒)
  
  e.g.,
  
  (person.location, person.gender, person.birthDay) ⇒ person.firstName
  person.location ⇒ person.lastName
  person.location ⇒ person.university
  person.createdDate ⇒ person.photoAlbum.createdDate
  ....

“S3G2: A Scalable Structure-correlated Social Graph Generator”
Correlated Edge Generation

- Student: "Anna" (University of Leipzig, Germany, 1990)
- Student: "Laura" (University of Leipzig, 1990)
- Student: Britney Spears (University of Leipzig, University of Amsterdam, Netherlands, 1990).
Correlated Edge Generation

“Student”

“Anna”

“University of Leipzig”

“Laura”

“1990”

“Britney Spears”

“University of Leipzig”

“1990”

“University of Leipzig”

“1990”

“University of Amsterdam”

“Germany”

“Netherlands”
Simple approach

- Compute **similarity** of two nodes based on their (correlated) **properties**.
- Use a **probability density function** wrt to this similarity for connecting nodes.

Danger: this is very expensive to compute on a large graph! (quadratic, random access)
Our observation

Probabilty that two nodes are connected is skewed w.r.t. the similarity between the nodes (due to probability distr.)
Correlation Dimensions

Similarity metric + Probability function

- **Similar metric**
  
  Sort nodes on similarity (similar nodes are brought near each other)

  
  ![node positions](image)

  <Ranking along the “Having study together” dimension>
  
  we use **space filling curves** (e.g. Z-order) to get a linear dimension

- **Probability function**
  
  Pick edge between two nodes based on their ranked distance
  
  (e.g. geometric distribution, again)
Generate edges along correlation dimensions

- Sort nodes using **MapReduce** on similarity metric
- Reduce function keeps a window of nodes to generate edges
  - Keep low memory usage (sliding window approach)
- Slide the window for **multiple passes**, each pass corresponds to one correlation dimension (multiple MapReduce jobs)
  - for each node we choose **degree** per pass (also using a prob. function)
    - steers how many edges are picked in the window for that node

“S3G2: A Scalable Structure-correlated Social Graph Generator”
Correlation Dimensions in LDBC SNB

- Having studied together
- Having common interests (hobbies)
- Random dimension
  - motivation: not all friendships are explainable (…)

(of course, these two correlation dimensions are still a gross simplification of reality, but this provides some interesting material for benchmark queries)

“S3G2: A Scalable Structure-correlated Social Graph Generator”
Evaluation (… see the TPCTC 2012 paper)

- **Social graph characteristics**
  - Output graph has similar characteristics as observed in real social network (i.e., “small-world network” characteristics)
    - Power-law social degree distribution
    - Low average path-length
    - High clustering coefficient

- **Scalability**
  - Generates up to 1.2 TB of data (1.2 million users) in half an hour
    - Runs on a cluster of 16 nodes
      (part of the SciLens cluster, www.scilens.org)
  - Scales out linearly

“S3G2: A Scalable Structure-correlated Social Graph Generator”
Summary

- correlation between values ("properties") and connection pattern in graphs affects many real-world data management tasks
  - use as a choke point in the Social Network Benchmark

- generating huge correlated graphs is hard!
  - MapReduce algorithm that approximates correlation probabilities with windowed-approach

See: for more info
- https://github.com/ldbc
- SNB task-force wiki http://www.ldbc.eu:8090/display/TUC
Roadmap for the Keynote

**Choke-point** based benchmark design

- What are Choke-points?
  - examples from good-old TPC-H

- Graph Choke-Point In depth
  - Structural Correlation in Graphs
  - And what we do about it in LDBC

- **Wrap up**
LDBC Benchmark Status

- **Social Network Benchmark**
  - **Interactive Workload**
    - Lookup queries + updates
    - Navigation between friends and posts
    - Graph DB, RDF DB, Relational DB
  - **Business Intelligence Workload**
    - Heavy Joins, Group-By + navigation!
    - Graph DB, RDF DB, Relational DB
  - **Graph Analytics**
    - Graph Diameter, Graph Clustering, etc.
    - Graph Programming Frageworks, Graph DB (RDF DB?, Relational DB?)
LDBC Benchmark Status

- Social Network Benchmark
- Semantic Publishing Benchmark
  - BBC use case (BBC data + queries)
    - Continuous updates
    - Aggregation queries
    - Light-weight RDF reasoning
LDBC Next Steps

- Benchmark Interim Reports
  - November 2013
  - SNB and Semantic Publishing

- Meet LDBC @ GraphConnect
  - 3rd Technical User Community (TUC) meeting
  - London, November 19, 2013
Conclusion

- LDBC: a new graph/RDF benchmarking initiative
  - EU initiated, Industry supported
  - benchmarks under development (SNB, SPB)
    - more to follow

- Choke-point based benchmark development
  - Graph Correlation
thank you very much. Questions?