LDBC benchmarks: three aspects of graph processing

Gábor Szárnyas
szarnyas@mit.bme.hu

11th TUC meeting
Austin, TX
Mission statement

LDBC is a non-profit organization dedicated to establishing benchmarks, benchmark practices and benchmark results for graph data management SW.

LDBC’s Social Network Benchmark is an industrial and academic initiative, formed by principal actors in the field of graph-like data management.
Graph processing landscape

Three key aspects
<table>
<thead>
<tr>
<th>OLTP</th>
<th>local queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLAP</td>
<td>global queries</td>
</tr>
<tr>
<td>analytics</td>
<td>global computations</td>
</tr>
</tbody>
</table>
Graph processing landscape

<table>
<thead>
<tr>
<th>OLTP</th>
<th>local queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example:</td>
<td>“Friends’ recent likes”</td>
</tr>
<tr>
<td>MATCH</td>
<td>(u:User {id: $uID})-[FRIEND]-(f:User)-[LIKES]-&gt;(p:Post)</td>
</tr>
<tr>
<td>RETURN</td>
<td>f, p</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>1.timestamp DESC</td>
</tr>
<tr>
<td>LIMIT</td>
<td>10</td>
</tr>
</tbody>
</table>

<p>| OLAP      | global queries |
| analytics | global computations |</p>
<table>
<thead>
<tr>
<th>OLTP</th>
<th>local queries</th>
<th>limited data</th>
<th>frequent up.</th>
</tr>
</thead>
</table>

Orri Erling et al.,
The LDBC Social Network Benchmark: Interactive Workload, SIGMOD 2015

14 complex reads, 7 simple reads, 8 updates
Queries explore the graph around a given node

<table>
<thead>
<tr>
<th>OLAP</th>
<th>global queries</th>
</tr>
</thead>
</table>

analytics | global computations |
### Graph processing landscape

<table>
<thead>
<tr>
<th>OLTP</th>
<th>local queries</th>
<th>limited data</th>
<th>frequent up.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLAP</td>
<td>global queries</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example: “One-sided friendships”**

MATCH (u1:User)-[:FRIEND]-(u2:User)-[:LIKES]->(p:Post),
    (u1)-[:AUTHOR_OF]->(p)
WITH u1, u2, count(l) AS likes
WHERE likes > 10
    AND NOT (u1)-[:LIKES]->(:Post)<-[:AUTHOR_OF]-(u2)
RETURN u1, u2

| analytics  | global computations   |              |              |
### Graph processing landscape

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>local</td>
<td>global</td>
</tr>
<tr>
<td>Data</td>
<td>limited</td>
<td>lots of</td>
</tr>
<tr>
<td>Updates</td>
<td>frequent</td>
<td>infrequent</td>
</tr>
</tbody>
</table>

Gábor Szárnyas et al.,
*An early look at the LDBC Social Network Benchmark’s Business Intelligence Workload,*
GRADES-NDA 2018

25 queries with infrequent executions
Queries explore a large portion of the graph
Graph processing landscape

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>local queries</td>
<td>global queries</td>
</tr>
<tr>
<td>Data</td>
<td>limited data</td>
<td>lots of data</td>
</tr>
<tr>
<td>Updates</td>
<td>frequent up.</td>
<td>infrequent up.</td>
</tr>
</tbody>
</table>

analytics | global computations

Example: “Find the most central individuals.”

- **BFS**: breadth-first search
- **PR**: PageRank
- **CDLP**: community detection by label propagation
- **WCC**: weakly connected components
- **LCC**: local clustering coefficient
- **SSSP**: single-source shortest path
# Graph processing landscape

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>OLAP</th>
<th>Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>local queries</td>
<td>global queries</td>
<td>global computations</td>
</tr>
<tr>
<td>Data</td>
<td>limited data</td>
<td>lots of data</td>
<td>all data</td>
</tr>
<tr>
<td>Updates</td>
<td>frequent up.</td>
<td>infrequent up.</td>
<td>no updates</td>
</tr>
</tbody>
</table>

Alexandru Iosup et al., *LDBC Graphalytics: A Benchmark for Large-Scale Graph Analysis on Parallel and Distributed Platforms*, VLDB 2016

One-time execution
No updates
<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>local queries</td>
<td>global queries</td>
</tr>
<tr>
<td>Data</td>
<td>limited data</td>
<td>lots of data</td>
</tr>
<tr>
<td>Updates</td>
<td>frequent up.</td>
<td>infrequent up.</td>
</tr>
</tbody>
</table>

Established solutions for relational data:

- Indexing
- Materialized views
- Column stores
- Data warehouses
Challenges
What makes graph queries difficult?
Choke points

• Choke point: a challenging aspect of query processing [QOPT/QEXE]
• Allows systematic benchmark design

CP-2.1: [QOPT] Rich join order optimization

This choke-point tests the ability of the query optimizer to find optimal join orders. A graph can be traversed in different ways. In the relational model, this is equivalent as different join orders. The execution time of these orders may differ by orders of magnitude. Therefore, finding an efficient join (traversal) order is important, which in general, requires enumeration of all the possibilities. The enumeration is complicated by operators that are not freely re-orderable like semi-, anti-, and outer-joins. Because of this difficulty most join enumeration algorithms do not enumerate all possible plans, and therefore can miss the optimal join order. Therefore, these chokepoint tests the ability of the query optimizer to find optimal join (traversal) orders.

Peter Boncz, Thomas Neumann, Orri Erling,
TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark, TPCTC 2013
Graph processing challenges / 1

connectedness
the “curse of connectedness”

data structures contemporary computer architectures are good at processing are linear and simple hierarchical structures, such as Lists, Stacks, or Trees

a massive amount of random data access is required […] poor performance since the CPU cache is not in effect for most of the time. […] parallelism is difficult

B. Shao, Y. Li, H. Wang, H. Xia (Microsoft Research), *Trinity Graph Engine and its Applications*, IEEE Data Engineering Bulletin 2017
existing graph query methods [...] focus on the topological structure of graphs and few have considered attributed graphs.

applications of large graph databases would involve querying the graph data (attributes) in addition to the graph topology.

answering queries that involve predicates on the attributes of the graphs in addition to the topological structure [...] makes evaluation and optimization more complex.

S. Sakr, S. Elnikety, Y. He (Microsoft Research),
G-SPARQL: A Hybrid Engine for Querying Large Attributed Graphs,
CIKM 2012
LDBC benchmarks
Timeline

2012
2013
2014
2015
2016
2017
2018

1
2
3
4
5
6
7
8
9
10
11

Interactive SIGMOD 2015
Graphalytics VLDB 2016
BI GRADES-NDA @SIGMOD 2018

EU FP7 project
TUC meetings
Benchmark papers
LDBC benchmarks at a glance

- Interactive
- Business Intelligence
- Graphalytics

Social Network Benchmark (SNB)
LDBC benchmarks at a glance

- Interactive
- Business Intelligence
- Graphalytics

- Amount of data accessed
- Expected execution time
Graphalytics workload
Alexandru Iosup et al.
Graphalytics

- An LDBC benchmark
- Advanced benchmarking harness
- Many classes of algorithms used in practice
- Diverse real and synthetic datasets
- Diverse set of experiments representative for practice
- Renewal process to keep the workload relevant
- Extended toolset for manual choke-point analysis
- Enables comparison of many platforms, community-driven and industrial

[l qosup et al., VLDB’16] [Guo et al., CCGRID’15] [Guo et al., IPDPS’14]

graphalytics.org
ldbccouncil.org/ldbc-graphalytics
Graphalytics Global Competition

• Systematic and periodic comparison of Graph processing systems.
• Register & submit benchmark results at graphalytics.org
Automated Bottleneck Detection and Performance Issue Identification

**Without Grade10:**

- CPU usage < 32 cores (100%)
- No bottleneck visible.. yet

**With Grade10:**

- Average time bottlenecked for Compute/ComputeThread:
  - None: 0 ms (always bottlenecked)
  - Message queue full: 1768 ms
  - Garbage collect: 781 ms
  - CPU: 748 ms

**WorkerSuperstep**

CPU usage < 32 cores (100%)
No bottleneck visible.. yet
Social Network Benchmark
SNB workloads
SNB task force

Arnaud Prat
Sparsity / DAMA-UPC
(Task Force Leader)

Alex Averbuch
Neo4j

Gábor Szárnyas
BME / MTA-BME

Vlad Haprian
Oracle Labs

Marcus Paradies
DLR
LDBC benchmarks at a glance

- **Graphalytics**
- **Business Intelligence**
- **Interactive**

- **SNB workloads**
  - same data generator,
  - same choke points
Data generator

github.com/ldbc/ldbc_snb_datagen
Social network graph

Realistic generator:
• DATAGEN
• Increasing scale factors (SFs)

Nodes:
• Collection attributes
• Type inheritance

Edges:
• Attributes
• Edges between similar nodes
  • Network of Persons
  • Reply tree of Posts/Comments
Workload specifications

github.com/ldbc/ldbc_snbb_docs
Choke points [execution]

- Graph-specific challenges:
  - Cache-unfriendliness, difficult to index, difficult to parallelize

CP-3.3: [QEXE] Scattered index access patterns

This choke-point tests the performance of indexes when scattered accesses are performed. The efficiency of index lookup is very different depending on the locality of keys coming to the indexed access. Techniques like vectoring non-local index accesses by simply missing the cache in parallel on multiple lookups vectored on the same thread may have high impact. Also detecting absence of locality should turn off any locality dependent optimizations if these are costly when there is no locality. A graph neighborhood traversal is an example of an operation with random access without predictable locality.

Queries

- BI 4, BI 5, BI 7, BI 8, BI 15, BI 16, BI 19, BI 21, BI 22, BI 23, BI 25, IC 5, IC 7, IC 8, IC 9
- IC 10, IC 11, IC 12, IC 13, IC 14
Choke points [language]

New choke points to cover *language features*

- CP-8.1: Complex patterns
- CP-8.2: Complex aggregations
- CP-8.3: Ranking-style queries
  - “arg min”-style queries, `OVER` and `rank()` in PostgreSQL
- CP-8.4: Query composition
  - Focal point of G-CORE
- CP-8.5: Dates and times
  - Recent advancement in openCypher and Neo4j
- CP-8.6: Handling paths
  - Focal point of G-CORE
Choke points [language]: Paths

1. Path unwinding
   • Higher-order queries
     • e.g. for a given path, calculate a score for each edge and summarize them

2. Matching semantics ~ walks vs. trails vs. simple paths
   • Homomorphism-based
   • Isomorphism-based
     • No-repeated-anything
     • No-repeated-node semantics
     • No-repeated-edge semantics

3. Regular path queries (RPQs)

R. Angles et al.,
*Foundations of Modern Query Languages for Graph Databases*,
ACM Computing Surveys, 2017
Choke points [language]: Paths

CP-8.6: [LANG] Handling paths

Handling paths as first-class citizens is one of the key distinguishing features of graph database systems [3]. Hence, additionally to reachability-style checks, a language should be able to perform path unwinding [1], i.e. express queries that operate on elements of a path such as calculating a score for each edge of a path. Also, some use cases specify uniqueness constraints on paths, e.g. that a certain path must not have repeated nodes (referred to as “walks” in graph theory) or not have repeated edges (“trails” in graph theory). Following the definitions of paper [1], homomorphism-based semantics (no constraints on repetitions) and multiple flavours of isomorphism-based semantics (no-repeated-node, no-repeated-edge, and no-repeated-anything).

Cypher. Cypher uses no-repeated-edge matching semantics (in return, this semantics is sometimes dubbed as cyphermorphism). Configurable matching semantics (e.g. MATCH ALL WALKS) were proposed in the openCypher language. RPQs are also proposed in the openCypher language as path patterns.

G-CORE. G-CORE treats paths as first-order citizens: its path property graph data model can store paths in the graph model itself. However, the language only supports shortest path semantics (for tractability reasons) and does not allow enumeration of all paths. G-CORE uses homomorphism-based matching semantics.

SPARQL. SPARQL uses homomorphism-based matching semantics and supports RPQs as property paths. Isomorphism-based matching semantics can be expressed by introducing custom filtering condition on predicates, e.g. FILTER ( ?e1 != ?e2 ).
<table>
<thead>
<tr>
<th>Interactive workload</th>
<th>Complex reads</th>
<th>Short reads</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>C3</td>
<td>C8</td>
<td>U1-2</td>
</tr>
<tr>
<td>C2</td>
<td>C4</td>
<td>C9</td>
<td>U3-5</td>
</tr>
<tr>
<td>C3</td>
<td>C5</td>
<td>C10</td>
<td>U6</td>
</tr>
<tr>
<td>C4</td>
<td>C6</td>
<td>C11</td>
<td>U7-8</td>
</tr>
<tr>
<td>C5</td>
<td>C7</td>
<td>C12</td>
<td></td>
</tr>
</tbody>
</table>
**query**  
Interactive / complex / 2

**title**  
Recent posts and comments by your friends

**pattern**

<table>
<thead>
<tr>
<th>Person</th>
<th>knows</th>
<th>person: Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>id = $id</td>
<td></td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>firstName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lastName</td>
</tr>
<tr>
<td></td>
<td>hasCreator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>content / imageFile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>creationDate</td>
</tr>
</tbody>
</table>

**desc.**
Given a start Person, find (most recent) Messages from all of that Person’s friends, that were created before (and including) a given date.

**params**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Person.id</td>
<td>ID</td>
</tr>
<tr>
<td>2</td>
<td>date</td>
<td>DateTime</td>
</tr>
</tbody>
</table>

**result**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Message.hasCreator-&gt;Person.id</td>
<td>ID</td>
</tr>
<tr>
<td>2</td>
<td>Message.hasCreator-&gt;Person.firstName</td>
<td>String</td>
</tr>
<tr>
<td>3</td>
<td>Message.hasCreator-&gt;Person.lastName</td>
<td>String</td>
</tr>
<tr>
<td>4</td>
<td>Message.id</td>
<td>ID</td>
</tr>
<tr>
<td>5</td>
<td>Message.content or Post.imageFile</td>
<td>String</td>
</tr>
<tr>
<td>6</td>
<td>Message.creationDate</td>
<td>DateTime</td>
</tr>
</tbody>
</table>
BI workload
<table>
<thead>
<tr>
<th>query</th>
<th>BI / read / 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Related topics</td>
</tr>
<tr>
<td>pattern</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>desc.</td>
<td>Find all Messages that have a given Tag. Find the related Tags attached to replies of these Messages (direct relation not transitive). but only of those replies that do not have the given Tag. Group the Tags by name, and get the count of replies in each group.</td>
</tr>
<tr>
<td>params</td>
<td>tag 32-bit Integer</td>
</tr>
<tr>
<td>result</td>
<td>relatedTag.name String R</td>
</tr>
<tr>
<td></td>
<td>count 32-bit Integer R</td>
</tr>
<tr>
<td>sort</td>
<td>count ↓</td>
</tr>
<tr>
<td></td>
<td>relatedTag.name ↑</td>
</tr>
<tr>
<td>limit</td>
<td>100</td>
</tr>
<tr>
<td>CPs</td>
<td>1.6, 3.3, 5.2</td>
</tr>
</tbody>
</table>
Driver and implementations

github.com/ldbc/ldbc_snb_driver
github.com/ldbc/ldbc_snb_implementations
Implementing an SNB workload

1. Get / generate data set
2. Implement loader
3. Implement queries and driver adapter

Validation
1. Get / generate validation data sets
2. Cross-validate for multiple SFs
3. If required, fix issues and go to 2.

Validation is very time consuming, but…

• Even after 2 validated tools, there were bugs in both implementations
• Even after 3 validated tools, there were ambiguities in the spec
The SIGMOD 2015 paper had implementations for Virtuoso and Sparksee.

Current implementations:

- PostgreSQL
- Sparksee
- SPARQL (some fixes by students of Tomer Sagi @ University of Haifa)

Next up:

- Cypher
- ?
Implementations / BI workload

Cross-validated implementations:

• Cypher    Neo4j     25/25  
• SPARQL     Stardog   24/25  
• SQL        PostgreSQL 25/25  
• Imperative (C++) Sparksee 25/25  
• PGQL       Oracle Labs PGX 10/25  

Next up:
• Spark SQL
• Cypher for Apache Spark
• ?
Incremental View Maintenance (IVM)

LDBC BI queries helped identify challenges for IVM on graphs:
• Complex aggregations
• Nested data structures
• Higher-order queries (path unwinding)

Results:
• Rules to transform queries to nested relational algebra and to flat RA
• Open-source prototype (ingraph/openCypher), supports ~15/25 BI queries
• Incremental higher-order queries are an open problem

Gábor Szárnyas et al.,
Reducing Property Graph Queries to Relational Algebra for Incremental View Maintenance, arXiv preprint
Progress and roadmap
SNB progress report: 10\textsuperscript{th} vs. 11\textsuperscript{th} TUC

pre-10\textsuperscript{th} TUC
• 54 Trello cards
• Specification
  • 180+ commits
• DATAGEN
  • 40+ commits
• “Close to publication”

10\textsuperscript{th} – 11\textsuperscript{th} TUC
• 67 Trello cards
• Specification
  • 250+ commits
• DATAGEN
  • 50+ commits
• Driver and implementations
  • 600+ commits
Roadmap – 10th TUC

- Implement & validate for Neo4j, PostgreSQL and Sparksee ✓
- Publish a subset of the benchmark in a workshop ✓
  - GraphQ @ EDBT (late Nov)
  - GRADES @ SIGMOD (late March) ✓
- Gather feedback & refine ✓
- Define update operations ✗
- We are recruiting! ✓
Roadmap – 11th TUC

• Social Network Benchmark workloads
  • Goal: publish the BI workload as an industry track conference paper
  • Help industry adoption
  • Define update operations: insertions and deletes (cf. GDPR)

• Graphalytics
  • Goal: establish Graphalytics 2.0
  • Run global competition

• We are still recruiting!
Acknowledgements

Gábor Szárnyas was partially supported by NSERC RGPIN-04573-16 (Canada) and the MTA-BME Lendület Cyber-Physical Systems Research Group (Hungary).

DAMA-UPC research was supported by the grant TIN2017-89244-R from MINECO (Ministerio de Economia, Industria y Competitividad) and the recognition 2017SGR-856 (MACDA) from AGAUR (Generalitat de Catalunya).

Sparsity thanks the EU H2020 for funding the Uniserver project (ICT-04-2015-688540).