Benchmarking
Graph Data Management Systems
EDBT Summer School 2015

Peter Boncz
boncz@cwi.nl

1. **LDBC Social Network Benchmark**
   - **Tuesday:** LDBC & SNB introduction
   - **Friday:** SNB in depth

2. **SNB Programming Challenge**
   - www.cwi.nl/~boncz/snb-challenge
   - **Tuesday:** what it is about & hardware properties & tips
   - **Friday:** the solution space & winners
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

TPC-A
- price/perf trend 37% per year

TPC-C
- price/perf trend 37% per year

$/tx

© Jim Gray, 2005
What is the LDBC?

Linked Data Benchmark Council = LDBC

- Industry entity similar to TPC (www.tpc.org)
- Focusing on graph and RDF store benchmarking
LDBC Organization (non-profit)

+ non-profit members (FORTH, STI2) & personal members
+ Task Forces, volunteers developing benchmarks
+ TUC: Technical User Community (6 workshops, 36 graph and RDF user case studies, 12 vendor presentations)
What does a benchmark consist of?

• Four main elements:
  – *data schema*: defines the structure of the data
  – *workloads*: defines the set of operations to perform
  – *performance metrics*: used to measure (quantitatively) the performance of the systems
  – *execution rules*: defined to assure that the results from different executions of the benchmark are valid and comparable

• Software as Open Source (GitHub)
  – data generator, query drivers, validation tools, ...
LDBC Task Forces

- Semantic Publishing Benchmark Task Force
  - Develops industry-grade RDF benchmark

- Social Network Benchmark Task Force
  - Develops benchmark for graph data management systems
  - Broad coverage: three workloads

- Graph Analytics Task Force
  - Spin-off from the SNB task force

- Graph Query Language Task Force
  - Not strictly about benchmarking
  - Studies features of graph database query languages
Semantic Publishing Benchmark (SPB)

Bulgaria

Team GB's Campbell secures medal

Luke Campbell is guaranteed an Olympic medal after beating Bulgaria's Detelin Dalakliev in his bantamweight semi-final.
5 Aug 12

Bulgaria beat GB volleyball men
MEN'S VOLLEYBALL
29 Jul 12

Great Britain’s men produce a battling display on their Olympic debut but are beaten in straight sets by Bulgaria of Evandro Court.

Medal Table

<table>
<thead>
<tr>
<th>Rank</th>
<th>Country</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>United States</td>
<td>29</td>
<td>27</td>
<td>36</td>
<td>104</td>
</tr>
<tr>
<td>2</td>
<td>China</td>
<td>23</td>
<td>17</td>
<td>38</td>
<td>88</td>
</tr>
<tr>
<td>3</td>
<td>Great Britain &amp; N Ireland</td>
<td>19</td>
<td>17</td>
<td>29</td>
<td>66</td>
</tr>
<tr>
<td>63</td>
<td>Bulgaria</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Bulgaria Medallists

Bronze
Taneda Pircev
Men's Heavyweight (81kg)

Silver
Stanka Zlateva Hristova
Women’s Freestyle 72kg
SPB scope

• The scenario involves a media/publisher organization that maintains semantic metadata about its Journalistic assets (articles, photos, videos, papers, books, etc), also called Creative Works

• The Semantic Publishing Benchmark simulates:
  – Consumption of RDF metadata (Creative Works)
  – Updates of RDF metadata, related to Annotations

• Aims to be an industrially mature RDF database benchmark (SPARQL1.1, some reasoning, text and GIS queries, backup&restore)
Social Network Benchmark (SNB)

- Intuitive: everybody knows what a SN is
  - Facebook, Twitter, LinkedIn, ...
- SNs can be easily represented as a graph
  - Entities are the nodes (Person, Group, Tag, Post, ...)
  - Relationships are the edges (Friend, Likes, Follows, ...)
- Different scales: from small to very large SNs
  - Up to billions of nodes and edges
- Multiple query needs:
  - interactive, analytical, transactional
- Multiple types of uses:
  - marketing, recommendation, social interactions, fraud detection, ...
Audience

- For **developers** facing graph processing tasks
  - recognizable scenario to compare merits of different products and technologies
- For **vendors** of graph database technology
  - checklist of features and performance characteristics
- For **researchers**, both industrial and academic
  - challenges in multiple choke-point areas such as graph query optimization and (distributed) graph analysis
Data Schema

• Specified in UML for portability
  – Classes
  – associations between classes
  – Attributes for classes and associations
• Some of the relationships represent dimensions
  – Time (Y, QT, Month, Day)
  – Geography (Continent, Country, Place)
• Data Formats
  – CSV
  – RDF (Turtle + N3)
Social Network Benchmark: schema
Benchmark Workloads

- **Interactive**: tests throughput running short queries while consistently handling concurrent updates
  - *Show all photos posted by my friends that I was tagged in*

- **Business Intelligence**: consists of complex structured queries for analyzing online behavior
  - *Influential people the topic of open source development?*

- **Graph Analytics**: tests the functionality and scalability on most of the data as a single operation
  - *PageRank, Shortest Path(s), Community Detection*

More details will follow in the second lecture

Draft queries available on ldbcouncil.org website (deliverable D2.2.4) & github

---

GRADES2015 “Graphalytics: A Big Data Benchmark for Graph-Processing Platforms” - Mihai Capota, Tim Hegeman, Alexandru Iosup (TU Delft); Arnau Prat (UPC), Orri Erling (OpenLink Technologies), Peter Boncz (CWI)
Interactive (On-line) Workload

• Test online ACID features and scalability
• The system under test is expected to run in a steady state, providing durable storage
• Updates are typically small
• Updates will conflict a small percentage of the time
• Queries typically touch a small fraction of the database
Q1. Extract description of friends with a given name. Given a person’s **firstName**, return up to 20 people with the same first name, sorted by increasing distance (max 3) from a given **person**, and for people within the same distance sorted by last name. Results should include the list of workplaces and places of study.

Q2. Find the newest 20 posts and comments from your friends. Given a start **Person**, find (most recent) Posts and Comments from all of that Person’s friends, that were created before (and including) a given **Date**. Return the top 20 Posts/Comments, and the **Person** that created each of them. Sort results descending by creation date, and then ascending by Post identifier.

Q3. Friends within 2 steps that have recently traveled to countries X and Y. Find friends and friends of friends of a given **Person** who have made a post or a comment in the foreign **CountryX** and **CountryY** within a specified period of **DurationInDays** after a **startDate**. Return top 20 Persons, sorted descending by total number of posts.

Q4. New Topics. Given a start **Person**, find the top 10 most popular Tags (by total number of posts with the tag) that are attached to Posts that were created by that Person’s friends. Only include Tags that were attached to Posts created within a given **time interval**, and that were never attached to Posts created before this interval.

Q5. New groups. Given a start **Person**, find the top 20 **Forums** which that Person’s friends and friends of friends became members of after a given **Date**. Sort results descending by the number of Posts in each Forum that were created by any of these **Persons**.

Q7. Recent likes. For the specified **Person** get the most recent likes of any of the person’s posts, and the latency between the corresponding post and the like. Flag Likes from outside the direct connections. Return top 20 Likes, ordered descending by creation date of the like.

Q8. Most recent replies. This query retrieves the 20 most recent reply comments to all the posts and comments of **Person**, ordered descending by creation date.

Q9. Latest Posts. Find the most recent 20 posts and comments from all friends, or friends-of-friends of **Person**, but created before a **Date**. Return posts, their creators and creation dates, sort descending by creation date.

Q10. Friend recommendation. Find a friend of a friend who posts much about the interests of **Person** and little about topics that are not in the interests of the user. The search is restricted by the candidate’s **horoscopeSign**. Returns 10 Persons for whom the difference between the total number of their posts about the interests of the specified user and the total number of their posts that are not in the interests of the user, is as large as possible. Sort the result descending by this difference.

Q11. Job referral. Find a friend of the specified **Person**, or a friend of her friend (excluding the specified person), who has long worked in a company in a specified **Country**. Sort ascending by start date, and then ascending by person identifier. Top 10 result should be shown.

Q12. Expert Search. Find friends of a **Person** who have replied the most to posts with a tag in a given **TagCategory**. Count the number of these reply Comments, and collect the Tags that were attached to the Posts they replied to. Return top 20 persons, sorted descending by number of replies.

Q13. Single shortest path. Given **PersonX** and **PersonY**, find the shortest path between them in the subgraph induced by the Knows relationships. Return the length of this path.

Q14. Weighted paths. Given **PersonX** and **PersonY**, find all weighted paths of the shortest length between them in the subgraph induced by the Knows relationship. The weight of the path takes into consideration amount of Posts/Comments exchanged.
select ?group count (*)
where {
    {select distinct ?fr
     where {
         {%Person% snvoc:knows ?fr.} union
         {%Person% snvoc:knows ?fr2.
         ?fr2 snvoc:knows ?fr. filter (?fr != %Person%)}
     }
    }

    ?mem snvoc:joinDate ?date . filter (?date >= "%Date0%"^^xsd:date) .
}
group by ?group
order by desc(2) ?group
limit 20
Example: Q5 - Cypher

MATCH (person:Person)-[:KNOWS*1..2]-(friend:Person)
WHERE person.id={person_id}
MATCH (friend)<-[membership:HAS_MEMBER]-(forum:Forum)
WHERE membership.joinDate>{join_date}
MATCH (friend)<-[HAS_CREATOR]-(post:Post)<-[CONTAINER_OF]-(forum)
RETURN forum.title AS forum, count(post) AS postCount
ORDER BY postCount DESC

MATCH (person:Person)-[:KNOWS*1..2]-(friend:Person)
WHERE person.id={person_id}
MATCH (friend)<-[membership:HAS_MEMBER]-(forum:Forum)
WHERE membership.joinDate>{join_date}
MATCH (friend)<-[HAS_CREATOR]-(comment:Comment)
WHERE (comment)-[:REPLY_OF*0..]->(comment)-[REPLY_OF]->(post)<-[CONTAINER_OF]-(forum)
RETURN forum.title AS forum, count(comment) AS commentCount
ORDER BY commentCount DESC
Example: Q5 - Sparksee

```java
v.setLongVoid(personId);
long personOID = graph.findObject(personId, v);
Objects friends = graph.neighbors(personOID, knows, EdgesDirection.Outgoing);
Objects allFriends = graph.neighbors(friends, knows, EdgesDirection.Outgoing);
allFriends.union(friends);
allFriends.remove(personOID);
friends.close();
Objects members = graph.explode(allFriends, hasMember, EdgesDirection.Ingoing);
v.setTimestampVoid(date);
Objects candidate = graph.select(joinDate, Condition.GreaterEqual, v, members);
Objects finalSelection = graph.tails(candidate);
candidate.close();
members.close();
Objects posts = graph.neighbors(allFriends, hasCreator, EdgesDirection.Ingoing);
ObjectsIterator iterator = finalSelection.iterator();
while (iterator.hasNext()) {
    long oid = iterator.next();
    Container c = new Container();
    Objects postsGroup = graph.neighbors(oid, containerOf, EdgesDirection.Outgoing);
    Objects moderators = graph.neighbors(oid, hasModerator, EdgesDirection.Outgoing);
    long moderatorOid = moderators.any();
    moderators.close();
    Objects postsModerator = graph.neighbors(moderatorOid, hasCreator, EdgesDirection.Ingoing);
    postsGroup.difference(postsModerator);
    postsModerator.close();
    postsGroup.intersection(posts);
    long count = postsGroup.size();
    if (count > 0) {
        graph.getAttribute(oid, forumId, v);
        c.row[0] = db.getForumURI(v.getLong());
        c.compare2 = String.valueOf(v.getLong());
        c.row[1] = String.valueOf(count);
        c.compare = count;
        results.add(c);
    }
}
```
Business Intelligence Workload

• The workload stresses query execution and optimization
• Queries typically touch a large fraction of the data
• The queries are concurrent with trickle load
• The queries touch more data as the database grows
Graph Analytics Workload (Graphalytics)

- The analytics is done on most of the data in the graph as a single operation
- The analysis itself produces large intermediate results
- The analysis transactional: no need for isolation from possible concurrent updates
Graphalytics Architecture

GRADES 2015: “Graphalytics: A Big Data Benchmark for Graph-Processing Platforms”
Graphalytics Algorithms

• general statistics (STATS)
  – counts the numbers of vertices and edges in the graph and computes the mean local clustering coefficients

• breadth-first search (BFS)
  – traverses the graph starting from a seed vertex, visiting first all the neighbors of a vertex before moving to the neighbors of the neighbors.

• connected components (CONN) algorithm
  – determines for each vertex the connected component it belongs to.

• community detection (CD) algorithm
  – detects groups of nodes that are connected to each other stronger than they are connected to the rest of the graph

• graph evolution (EVO)
  – predicts the evolution of the graph according to the “forest fire” model

GRADES 2015: “Graphalytics: A Big Data Benchmark for Graph-Processing Platforms”
Systems

- **Graph database systems**
  - e.g. Neo4j, InfiniteGraph, DEX, Titan
- **Graph programming frameworks**
  - e.g. Giraph, Signal/Collect, Graphlab, Green Marl, Grappa
- **RDF database systems**
  - e.g. OWLIM, Virtuoso, BigData, Jena TDB, Stardog, Allegrograph
- **Relational database systems**
  - e.g. Postgres, MySQL, Oracle, DB2, SQLServer, Virtuoso, MonetDB, Vectorwise, Vertica
- **noSQL database systems**
  - e.g. HBase, REDIS, MongoDB, CouchDB, or even MapReduce systems like Hadoop and Pig
## Workloads by system

<table>
<thead>
<tr>
<th>System</th>
<th>Interactive</th>
<th>Business Intelligence</th>
<th>Graph Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph databases</td>
<td>Yes</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
<tr>
<td>Graph programming</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>frameworks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDF databases</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Relational databases</td>
<td>Yes</td>
<td>Yes</td>
<td>Maybe, by keeping state in temporary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tables, and using the functional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>features of PL-SQL</td>
</tr>
<tr>
<td>NoSQL Key-value</td>
<td>Maybe</td>
<td>Maybe</td>
<td></td>
</tr>
<tr>
<td>NoSQL MapReduce</td>
<td>-</td>
<td>Maybe</td>
<td>Yes</td>
</tr>
</tbody>
</table>
More Information


Blogs
Specifications
Early Result FDRs
Videos of TUC talks
Developer info
Code, Issue Tracking
SNB Challenge: Querying a Social Graph
LDBC SNB Data generator

• Synthetic dataset available in different scale factors
  – SF100 ← for quick testing
  – SF3000 ← the real deal

• Very complex graph
  – Power laws (e.g. degree)
  – Huge Connected Component
  – Small diameter
  – Data correlations
    *Chinese have more Chinese names*
  – Structure correlations
    *Chinese have more Chinese friends*
CSV file schema

- See: http://wikistats.ins.cwi.nl/lsde-data/practical_1
- Counts for sf3000 (total 37GB)

Person (9M)
- PersonId PK
- FirstName
- LastName
- Gender
- Birthday
- CreationDate
- LocationIP
- BrowserUsed
- LocatedIn

Knows (1.3B)
- PersonFrom
- PersonTo

interests (.2B)
- PersonID
- tagID

Tags (16K)
- TagID
- Name
- URL

Place (1.4K)
- PlaceID PK
- URL
- type
The Query

• The marketeers of a social network have been data mining the musical preferences of their users. They have built statistical models which predict given an interest in say artists A2 and A3, that the person would also like A1 (i.e. rules of the form: A2 and A3 $\rightarrow$ A1). Now, they are commercially exploiting this knowledge by selling targeted ads to the management of artists who, in turn, want to sell concert tickets to the public but in the process also want to expand their artists' fanbase.

• The ad is a suggestion for people who already are interested in A1 to buy concert tickets of artist A1 (with a discount!) as a birthday present for a friend ("who we know will love it" - the social network says) who lives in the same city, who is not yet interested in A1 yet, but is interested in other artists A2, A3 and A4 that the data mining model predicts to be correlated with A1.
The Query

For all persons P:
- who have their birthday on or in between D1..D2
- who do not like A1 yet
  we give a score of
    - 1 for liking any of the artists A2, A3 and A4 and
    - 0 if not
  the final score, the sum, hence is a number between 0 and 3.

Further, we look for friends F:
- Where P and F who know each other mutually
- Where P and F live in the same city and
- Where F already likes A1

The answer of the query is a table (score, P, F) with only scores > 0
Binary files

- Created by “loader” program in example github repo
- Total size: 6GB

**Person.bin**
- PersonId  PK
- Birthday
- LocatedIn
- Knows_first
- Knows_n
- Interests_first
- Interests_n

**Knows.bin**
- PersonPos

**interests.bin**
- tagID
What it looks like

- Created by “loader” program in example github repo
- Total size: 6GB

- 2bytes * 204M
- 48bytes * 8.9M
- 4bytes * 1.3B
The Naïve Implementation

The “cruncher” program

Go through the persons $P$ sequentially

• counting how many of the artists $A_2,A_3,A_4$ are liked as the score

for those with score $> 0$:

– visit all persons $F$ known to $P$.

For each $F$:

• checks on equal location
• check whether $F$ already likes $A_1$
• check whether $F$ also knows $P$

if all this succeeds $(\text{score},P,F)$ is added to a result table.
Naïve Query Implementation

• “cruncher”

2bytes * 204M

48bytes * 8.9M

4bytes * 1.3B

results
Memory Hierarchy

- Processor
- CPU
  - Processor Register
  - CPU Cache
    - Level 1 (L1) Cache
    - Level 2 (L2) Cache
    - Level 3 (L3) Cache
- Physical Memory
  - Random Access Memory (RAM)
- Solid State Memory
  - Non-volatile Flash-based Memory
- Virtual Memory
  - File-based Memory

EDO, SD-RAM, DDR-SDRAM, RD-RAM and More...

- SSD, Flash Drive
- Mechanical Hard Drives

Super fast, super expensive, tiny capacity;
Faster, expensive, small capacity;
Fast, priced reasonably, average capacity;
Average speed, priced reasonably, average capacity;
Slow, cheap, large capacity.

Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng
Hardware Progress

Microprocessor Transistor Counts 1971-2011 & Moore’s Law

Transistors
CPU performance

Single-Threaded Integer Performance
Based on adjusted SPECint® results

+21% per year
+52% per year

intel Xeon
intel Core
intel Pentium
intel Itanium
intel Celeron
AMD FX
AMD Opteron
AMD Phenom
AMD Athlon
IBM POWER
PowerPC
Fujitsu SPARC
Sun SPARC
DEC Alpha
MIPS
HP PA-RISC
RAM, Disk Improvement Over the Years

RAM

Magnetic Disk
Latency Lags Bandwidth

By David A. Patterson

Recognizing the chronic imbalance between bandwidth and latency, and how to cope with it.

As I review performance trends, I am struck by a consistent theme across many technologies: bandwidth improves much faster than latency.
# Geeks on Latency

## Latency Numbers Every Programmer Should Know

<table>
<thead>
<tr>
<th>Latency Comparison Numbers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5 ns</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5 ns</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7 ns</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25 ns</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100 ns</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000 ns</td>
</tr>
<tr>
<td>Send 1K bytes over 1 Gbps network</td>
<td>10,000 ns 0.01 ms</td>
</tr>
<tr>
<td>Read 4K randomly from SSD*</td>
<td>150,000 ns 0.15 ms</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000 ns 0.25 ms</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000 ns 0.5 ms</td>
</tr>
<tr>
<td>Read 1 MB sequentially from SSD*</td>
<td>1,000,000 ns 1 ms 4X memory</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000 ns 10 ms 20x datacenter roundtrip</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000 ns 20 ms 80x memory, 20X SSD</td>
</tr>
<tr>
<td>Send packet CA→Netherlands→CA</td>
<td>150,000,000 ns 150 ms</td>
</tr>
</tbody>
</table>

### Notes

- 1 ns = 10-9 seconds
- 1 ms = 10-3 seconds
- * Assuming ~1GB/sec SSD
Sequential Access Hides Latency

• Sequential RAM access
  – CPU prefetching: multiple consecutive cache lines being requested concurrently

• Sequential Magnetic Disk Access
  – Disk head moved once
  – Data is streamed as the disk spins under the head

• Sequential Network Access
  – Full network packets
  – Multiple packets in transit concurrently
Consequences For Algorithms

• Analyze the main data structures
  – How big are they?
    • Are they bigger than RAM?
    • Are they bigger than CPU cache (a few MB)?
  – How are they laid out in memory or on disk?
    • One area, multiple areas?

Java Object Data Structure vs memory pages (or cache lines)
Consequences For Algorithms

• Analyze your access patterns
  – Sequential: you’re OK
  – Random: it better fit in cache!
    • What is the access granularity?
    • Is there temporal locality? Is there spatial locality?
Improving Bad Access Patterns

• Minimize Random Memory Access
  – Apply filters first. Less accesses is better.

• Denormalize the Schema
  – Remove joins/lookups, add looked up stuff to the table (but.. makes it bigger)

• Trade Random Access For Sequential Access
  – perform a 100K random key lookups in a large table
    → put 100K keys in a hash table, then
      scan table and lookup keys in hash table

• Try to make the randomly accessed region smaller
  – Remove unused data from the structure
  – Apply data compression
  – Cluster or Partition the data (improve locality) ...hard for social graphs
Naïve Query Implementation

- “cruncher”

- 2 bytes * 204M
- 48 bytes * 8.9M
- 4 bytes * 1.3B