

Project Report

Benchmarking Graph Databases: Neo4j vs ArangoDB vs MongoDB vs OrientDB

By Tian Yang & John Li
ITEC6220 M - Advanced Information Management
(Winter 2018-2019)
Instructor: Prof. Xiaohui Yu

It's All about Performance

We are using the same datasets, same query workload, and same environment cross validation of result each graph database's benchmark.

- Orientdb Community Edition 3.0.17
- Neo4j Community Edition 3.4.12
- ArangoDB Community Edition 3.4.4
- MongoDB Community Edition version v4.0.8

Contents

- 1.Summary of the Research
- 2.Introduction
- 3.Test Setup
- 4.Description of Tests
- 5.Results
- 6.Conclusion
- 7.Future works
- 8.Appendix – Details about results

Summary of the Research

- (1) The highly-connected structure of many natural phenomena, such as road, biological, and social networks make graphs an obvious choice in modelling.
- (2) We notice the who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin OTC.
- (3) The following queries are commonly used, Create, Read, Update, and Delete (CRUD) operations
- (4) Graph databases are used for Shortest Path (such as Dijkstra's algorithm)[1] is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks, users' ratings.

Summary of the Research cont'd

- For comparison, we only used the leading graph database systems. We selected operational graph databases (OrientDB and Neo4j) and Multi-Modal Graphs (ArangoDB, MongoDB) (Datanami, A Look at the Graph Database Landscape, Yu Xu, November 30, 2017)
- We benchmarked them to against each other
- In the application domains where relationships are of importance, GDBMS increasingly gains popularity since the relationships can be explicitly modeled and easily visualized in a graph data model.

Summary of the Research cont'd

To select a standard dataset we decided to use Bitcoin OTC trust weighted signed network:

S. Kumar, F. Spezzano, V.S. Subrahmanian, C. Faloutsos.
[Edge Weight Prediction in Weighted Signed Networks](#). IEEE International Conference on Data Mining (ICDM), 2016.

S. Kumar, B. Hooi, D. Makhija, M. Kumar, V.S. Subrahmanian, C. Faloutsos. [REV2: Fraudulent User Prediction in Rating Platforms](#). 11th ACM International Conference on Web Search and Data Mining (WSDM), 2018.

Summary of the Research cont'd

As an addition, full-text search is important for most of the databases. We choice Infrastructure Canada data (10M, consider we are running a free VMs)

This dataset contains a list of infrastructure projects across Canada that have been approved by Infrastructure Canada. The project information listed is based on current information.

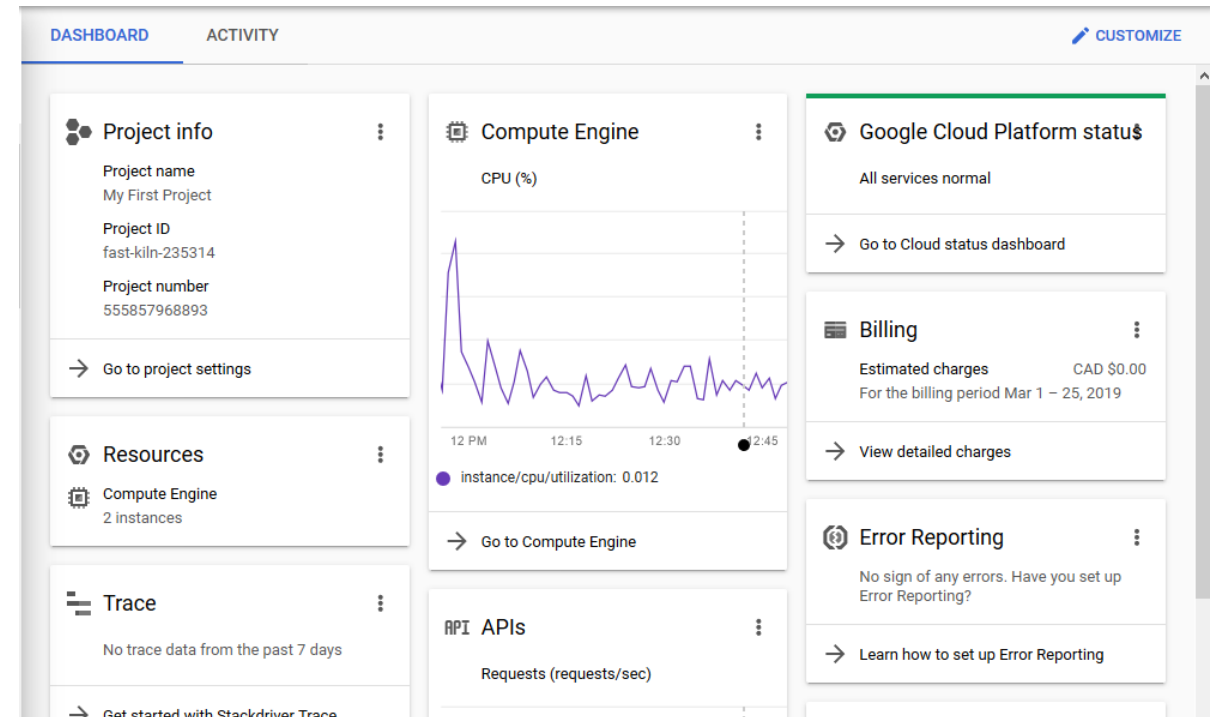
Research Motivation and Aim

For comparison, we used the leading single-model database systems: Neo4j vs ArangoDB vs MongoDB vs OrientDB .

Test Setup

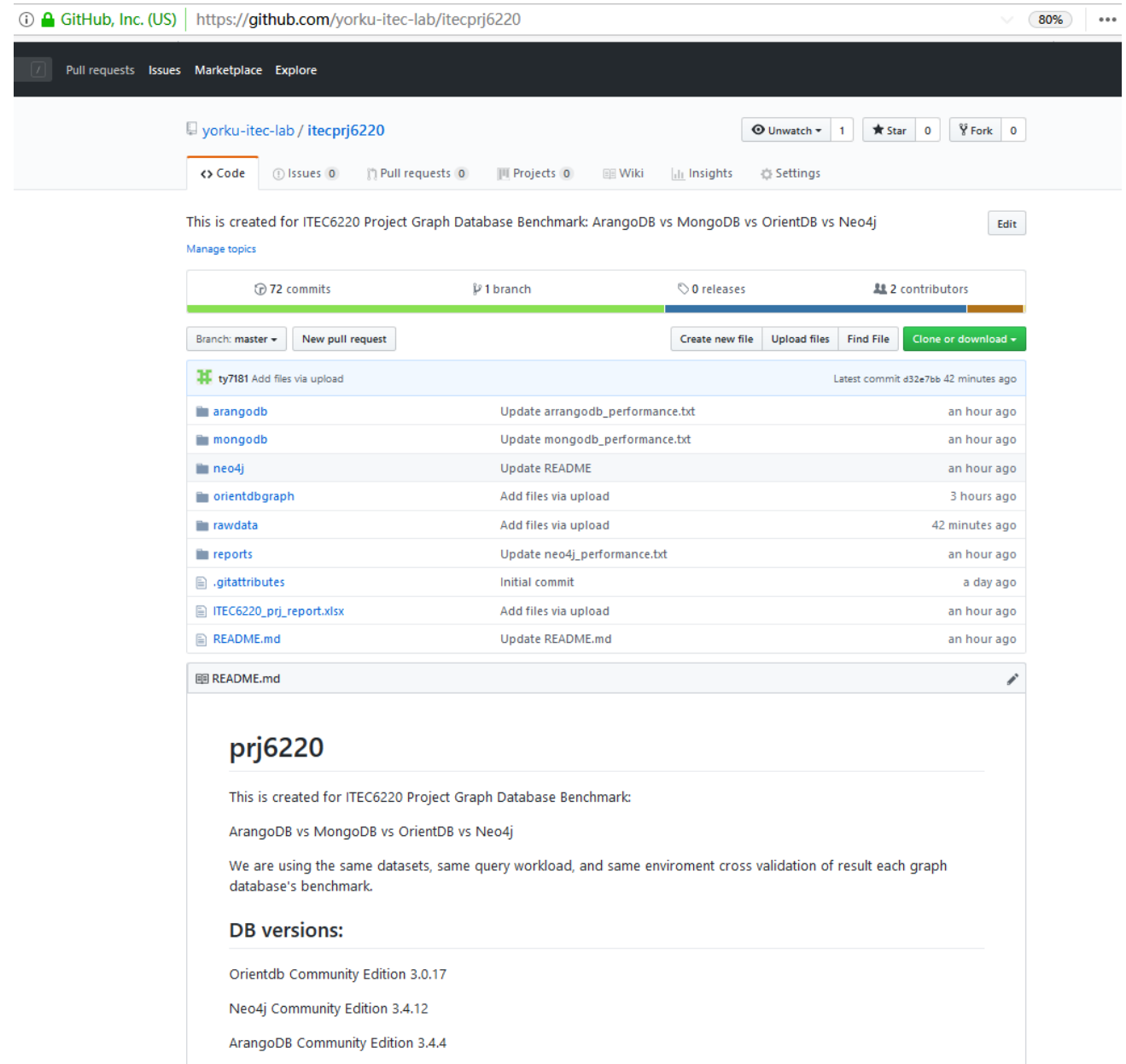
We used a simple google cloud VM setup and instances google recommends for both relational and non-relational databases.

- Machine type custom (4 vCPU, 20 GB memory)
Intel Haswell
Ubuntu 18.04 LTS amd64 bionic image built



Test Setup cont'd

We used a Github repository to host all development code and raw data we got from Stanford University (Stanford Network Analysis Project)



The screenshot shows a GitHub repository page for 'yorku-itec-lab / itecprj6220'. The repository is created for the ITEC6220 Project Graph Database Benchmark, comparing ArangoDB, MongoDB, OrientDB, and Neo4j. It features 72 commits, 1 branch, 0 releases, and 2 contributors. The repository includes folders for 'arangodb', 'mongodb', 'neo4j', 'orientdbgraph', 'rawdata', 'reports', and files for '.gitattributes', 'ITEC6220_prj_report.xlsx', and 'README.md'. The README file is open, showing the project title 'prj6220' and a description of the benchmark setup, including the use of the same datasets, query workload, and environment for cross-validation. It also lists the database versions used: OrientDB Community Edition 3.0.17, Neo4j Community Edition 3.4.12, and ArangoDB Community Edition 3.4.4.

GitHub, Inc. (US) | <https://github.com/yorku-itec-lab/itecprj6220> 80%

Pull requests Issues Marketplace Explore

yorku-itec-lab / itecprj6220 Unwatch 1 Star 0 Fork 0

<> Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

This is created for ITEC6220 Project Graph Database Benchmark: ArangoDB vs MongoDB vs OrientDB vs Neo4j Edit

Manage topics

72 commits 1 branch 0 releases 2 contributors

Branch: master New pull request Create new file Upload files Find File Clone or download

ty7181 Add files via upload Latest commit d32e7bb 42 minutes ago

arangodb	Update arangodb_performance.txt	an hour ago
mongodb	Update mongodb_performance.txt	an hour ago
neo4j	Update README	an hour ago
orientdbgraph	Add files via upload	3 hours ago
rawdata	Add files via upload	42 minutes ago
reports	Update neo4j_performance.txt	an hour ago
.gitattributes	Initial commit	a day ago
ITEC6220_prj_report.xlsx	Add files via upload	an hour ago
README.md	Update README.md	an hour ago

README.md

prj6220

This is created for ITEC6220 Project Graph Database Benchmark:

ArangoDB vs MongoDB vs OrientDB vs Neo4j

We are using the same datasets, same query workload, and same enviroment cross validation of result each graph database's benchmark.

DB versions:

Orientdb Community Edition 3.0.17

Neo4j Community Edition 3.4.12

ArangoDB Community Edition 3.4.4

Test Setup cont'd

The dataset is soc-sign-bitcoinotc.csv.gz

Weighted Signed Directed Bitcoin OTC web of trust network

- Each line has one rating, sorted by time, with the following format: SOURCE, TARGET, RATING, TIME
 - SOURCE: node id of source, i.e., rater
 - TARGET: node id of target, i.e., ratee
 - RATING: the source's rating for the target, ranging from -10 to +10 in steps of 1
 - TIME: the time of the rating, measured as seconds since Epoch.

Dataset statistics

Nodes	5,881
Edges	35,592
Range of edge weight	-10 to +10
Percentage of positive edges	89%

Test Setup cont'd

We used the latest GA versions (as of March 10, 2019) of all database systems and not to include the RC versions. Below are a list of the databases we used for each product:

- Neo4j
- ArangoDB
- MongoDB
- OrientDB
- For this benchmark we used NodeJS 8.9.4 Node.js is an open source server environment, and it allows us to run JavaScript on the server.

Descriptions of Tests

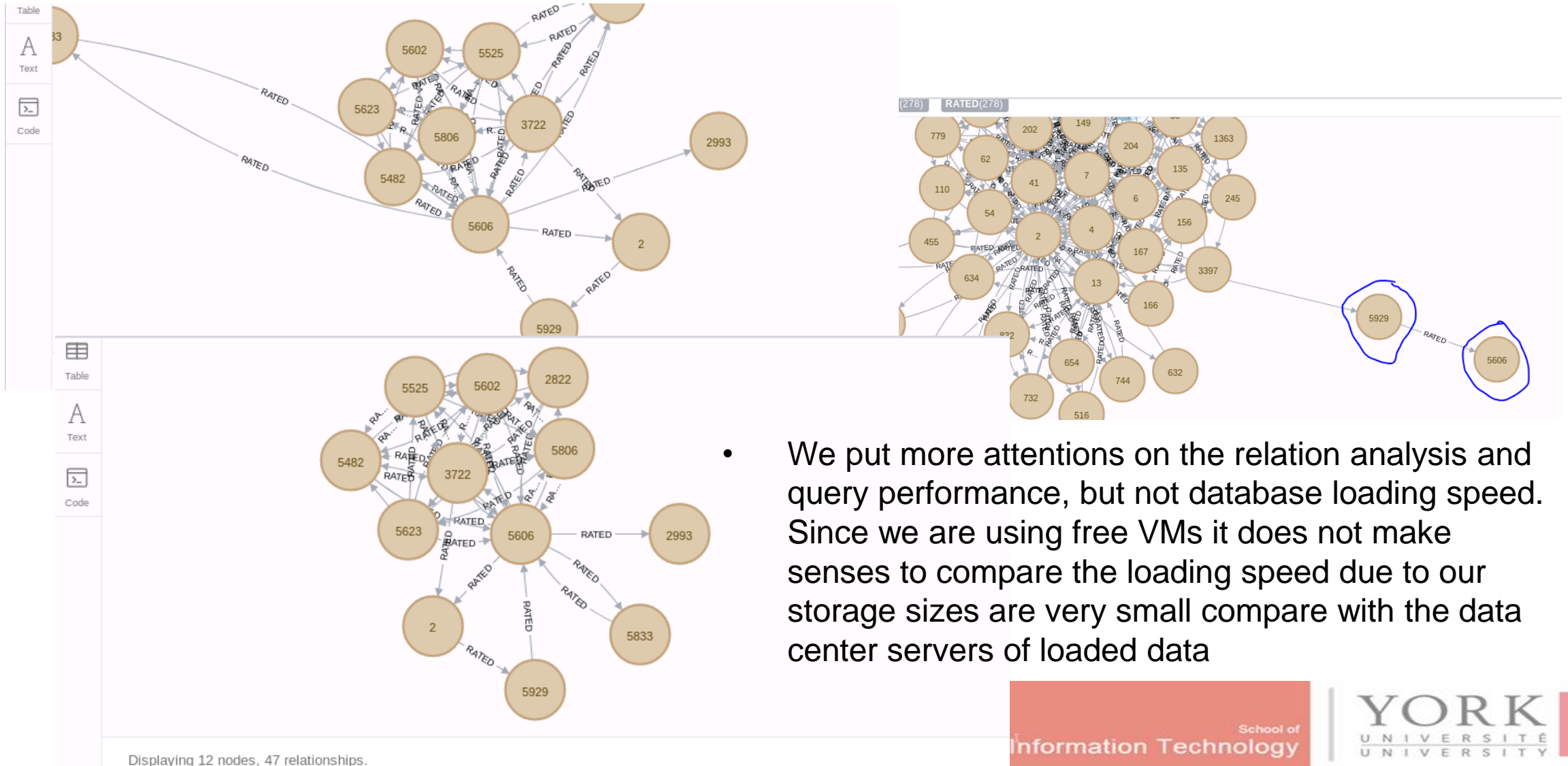
The goal of the benchmark is to measure the performance of each database system when there is no query cache used. To be assured of this, we disabled the query cache for each software that offered one. For our tests we ran the workloads five times, averaging the results. Each test starts with an individual warm-up phase that allows the database systems to load data in memory.

Descriptions of Tests cont'd

The following test cases have been included, as far as the database system was capable of performing the query:

- insert and remove
 - simple CRUD, with multiple records retrieving and or updating
- selection and sort
 - Simple CRUD selecting and sorting
- neighbors
 - Finding neighbors nearest: finding (distinct) direct neighbors, returning IDs
- path finding
 - Finding the path length between 2 nodes, Returning all pathes' parts with length 2 and 3, start vertex S and target vertex E:
- Full-text search (on different dataset)

Descriptions of Tests cont'd



Descriptions of Tests cont'd

The screenshot shows the Neo4j Browser interface in a Mozilla Firefox browser. The address bar shows 'localhost:7474/browser/'. The left sidebar contains 'Database Information' with sections for Node Labels (4814 RateUser), Relationship Types (33766 HasRated), Property Keys (RateUserId, rating, timestamp), and Database details (Version: 3.5.4, Edition: Community, Name: graph.db). The main area displays a Cypher query:

```
MATCH (u1:RateUser {RateUserId:387}), (u2:RateUser {RateUserId:104}), p = shortestPath((u1)-[:HasRated*]-(u2)) WHERE ALL (r IN relationships(p) WHERE exists(r.role)) RETURN p
```

. Below the query, the results are shown in a table view with the column 'n'. The results are three JSON objects:

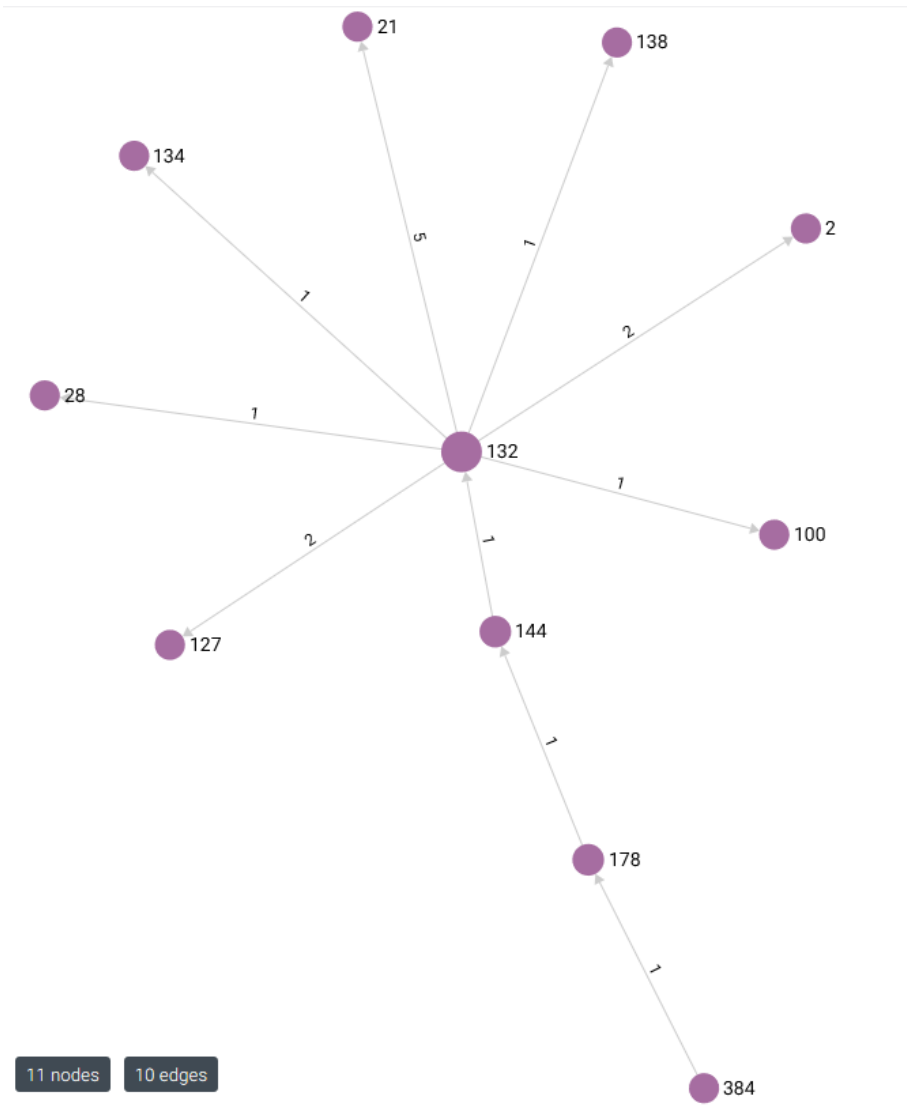
```
{ "RateUserId": "6" }, { "RateUserId": "1" }, { "RateUserId": "4" }
```

.

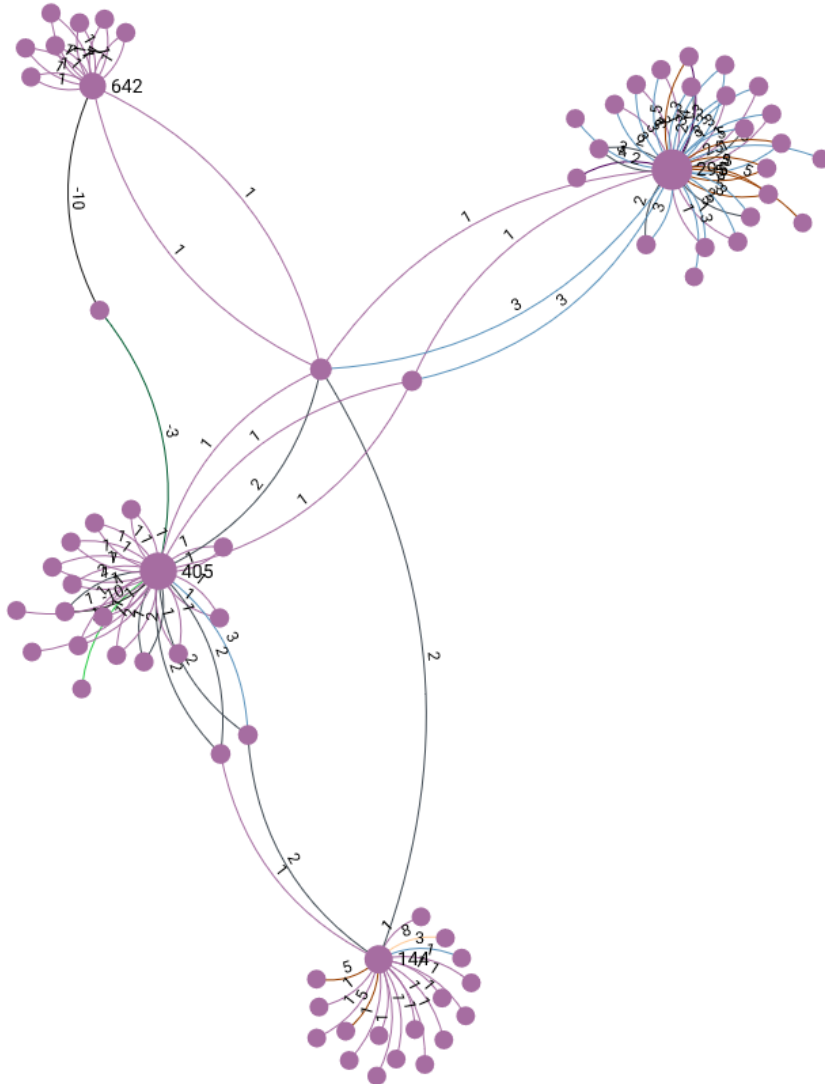
- We think the graph database performance aggregations and result sets is important.
- We measured the average execution times of the proposed shortest path, full-text search and CRUD operations on bitcoin data (Stanford Research dataset) by the benchmark with a reasonable number of nodes.
- The graphs were stored as graphs in graph databases
- For most CRUD operations including the graph structure queries in our benchmark, Arango was faster than others.

Descriptions of Tests cont'd

Finding Shortest Paths involves starting from a source node and successively exploring its outgoing edges. We are currently working on extending our benchmark to larger datasets as well as on including more queries.



Descriptions of Tests cont'd



Depth- and Breadth-First Search Algorithms

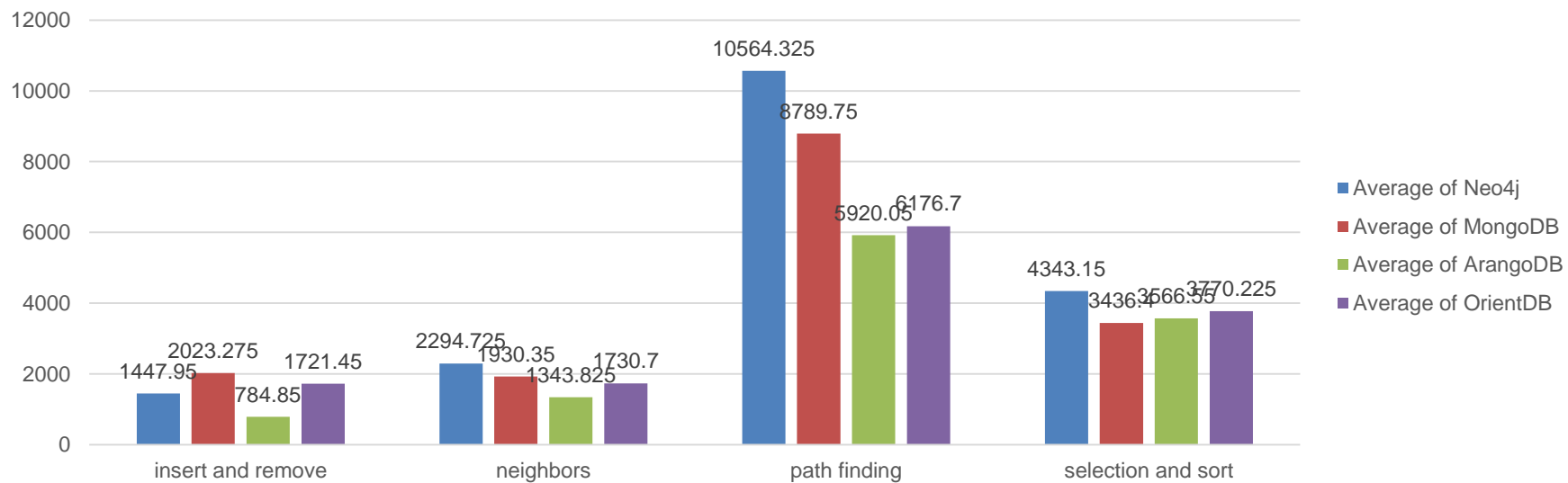
There are two basic types of graph search algorithms: depth-first and breadth-first.

On the contrary, dealing with semantically rich graph databases allows for informed searches, which conduct an early termination of a search if nodes with no compatible outgoing relationships are found. As a result, informed searches also have lower execution times.

As our test results, the ArangoDB performance is the best at this field.

Overall Results

The graph below shows the overall results of our performance benchmark. We noticed the ArangoDB and MongoDB is performance better when request larger amount of IOs, but Neo4j is returning better results when we retrieved small size data.

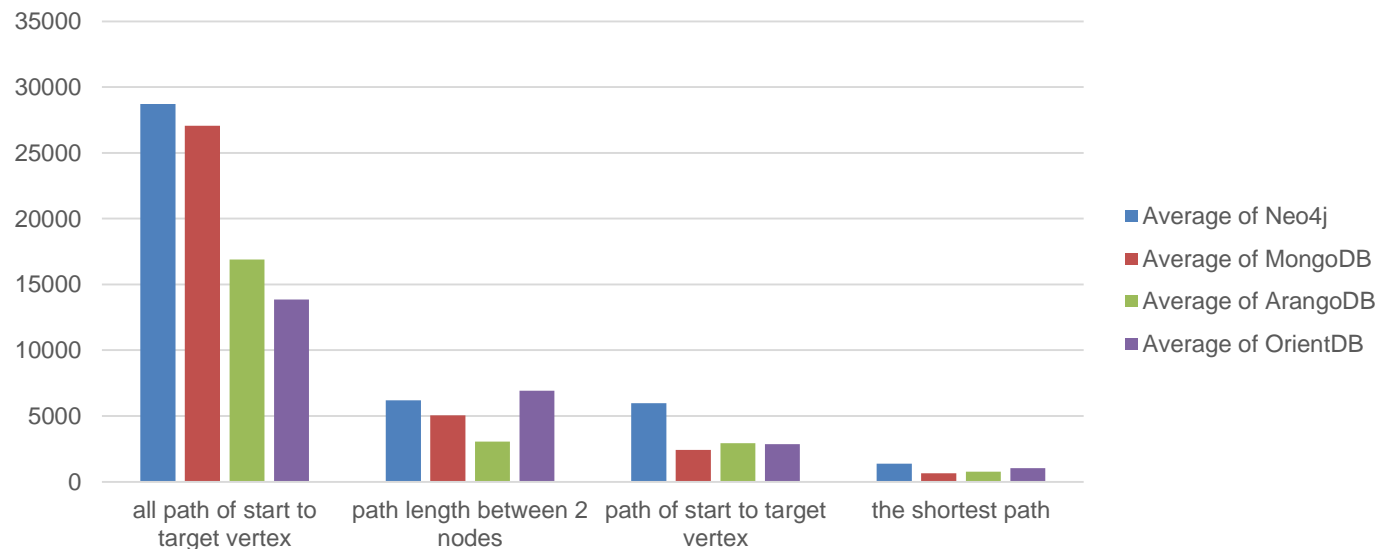


Overall Results cont'd

- In fundamental queries like single-read, single-write, as well as single-write sync Neo4j showed better results in single read/write tests
- These are just the results. To appreciate and understand them, we'll need look a little deeper into the individual results and focus on the more complex queries like aggregations and graphy functionalities.

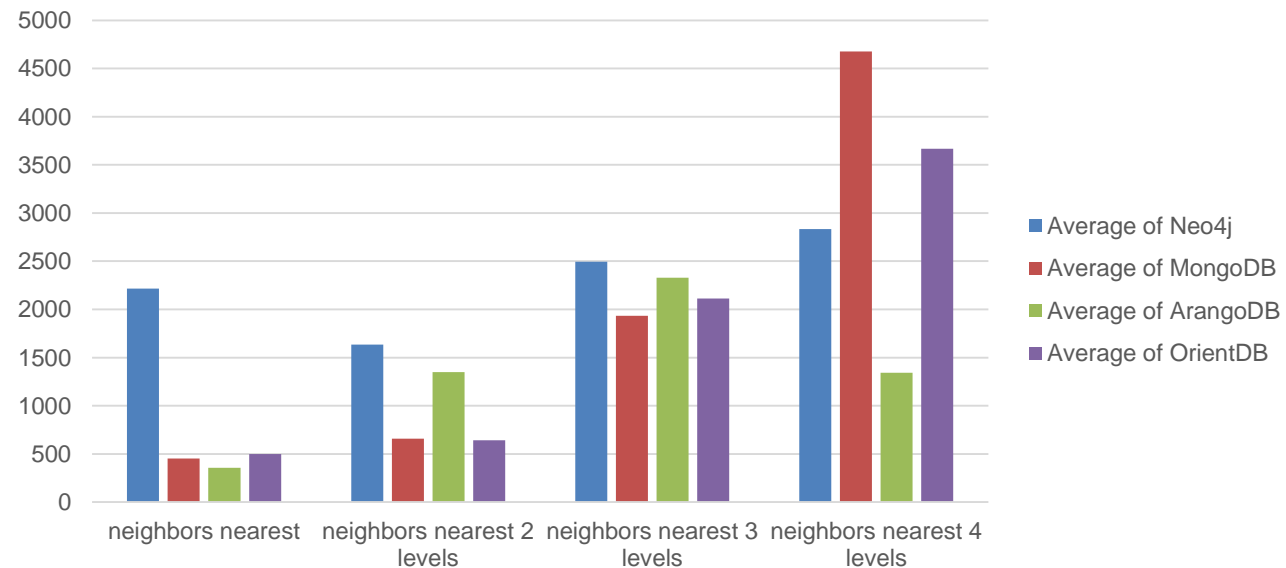
Test results for path finding (core performance for graph database)

Group	path finding			
Row Labels	Average of Neo4j	Average of MongoDB	Average of ArangoDB	Average of OrientDB
all path of start to target vertex	28717.8	27061.1	16906.3	13866.1
path length between 2 nodes	6189	5040.3	3063.2	6933.1
path of start to target vertex	5982.4	2416.7	2943.8	2864.9
the shortest path	1368.1	640.9	766.9	1042.7
Grand Total	10564.325	8789.75	5920.05	6176.7



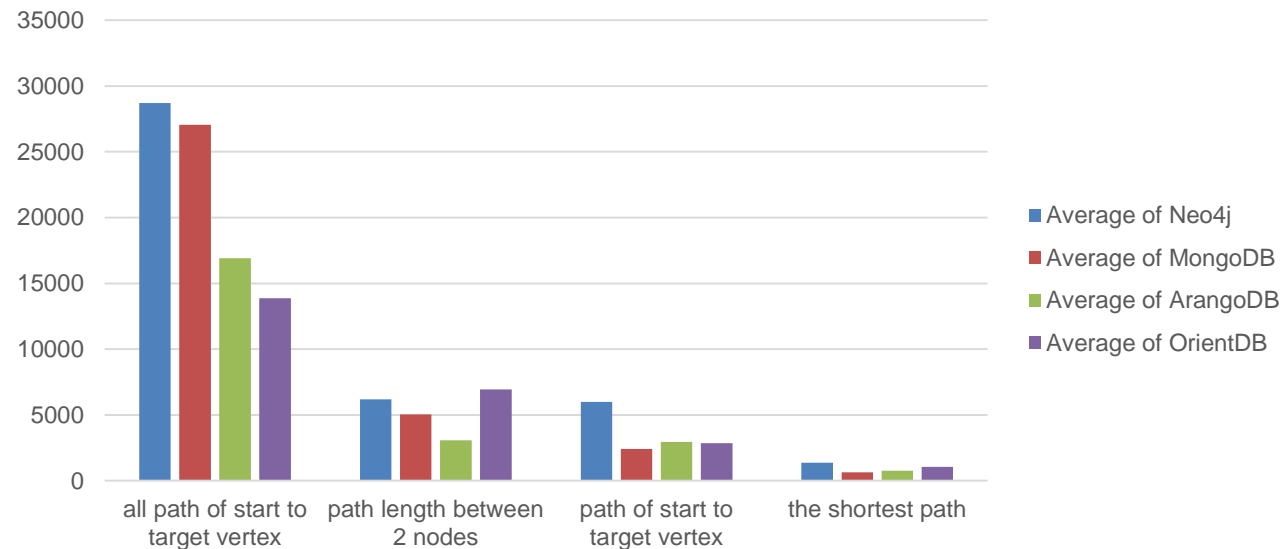
Test results for neighbors finding

Row Labels	Average of Neo4j	Average of MongoDB	Average of ArangoDB	Average of OrientDB
neighbors nearest	2216.5	452.8	357.4	499.9
neighbors nearest 2 levels	1634	658.2	1347.7	642.9
neighbors nearest 3 levels	2494.9	1933.7	2328.9	2113.6
neighbors nearest 4 levels	2833.5	4676.7	1341.3	3666.4
Grand Total	2294.725	1930.35	1343.825	1730.7



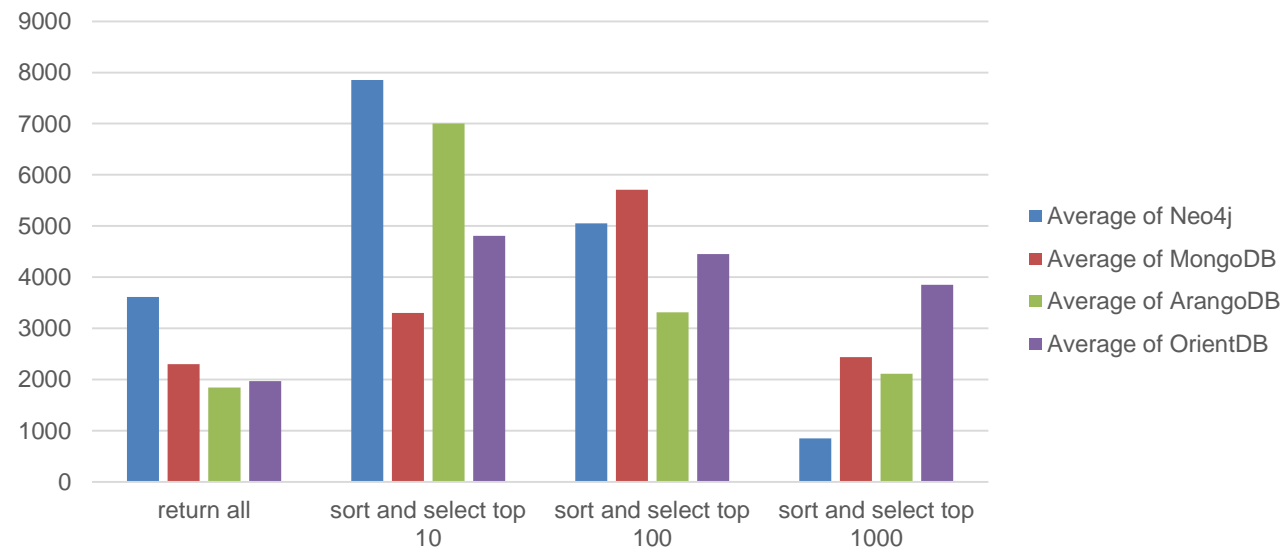
Test results for path finding

Row Labels	Average of Neo4j	Average of MongoDB	Average of ArangoDB	Average of OrientDB
all path of start to target vertex	28717.8	27061.1	16906.3	13866.1
path length between 2 nodes	6189	5040.3	3063.2	6933.1
path of start to target vertex	5982.4	2416.7	2943.8	2864.9
the shortest path	1368.1	640.9	766.9	1042.7
Grand Total	10564.325	8789.75	5920.05	6176.7



Test results for selections (simple and complex)

Row Labels	Average of Neo4j	Average of MongoDB	Average of ArangoDB	Average of OrientDB
return all	3615.4	2297.4	1841.4	1968.7
sort and select top 10	7855.4	3302	7000.6	4809.5
sort and select top 100	5050.5	5708.5	3314.8	4451.7
sort and select top 1000	851.3	2437.7	2109.4	3851
Grand Total	4343.15	3436.4	3566.55	3770.225



Conclusion

- When doing benchmark tests that different hardware can produce different results.
- The performance needs may vary and the requirements may differ.
- In this benchmark we could show, that ArangoDB and MongoDB can compete with the leading single-model database systems on their home turf.
- In conclusion, the excellent performance of a ArangoDB is a key advantage of others

Extra Exam - Text search – MongoDB vs. ArangoDB

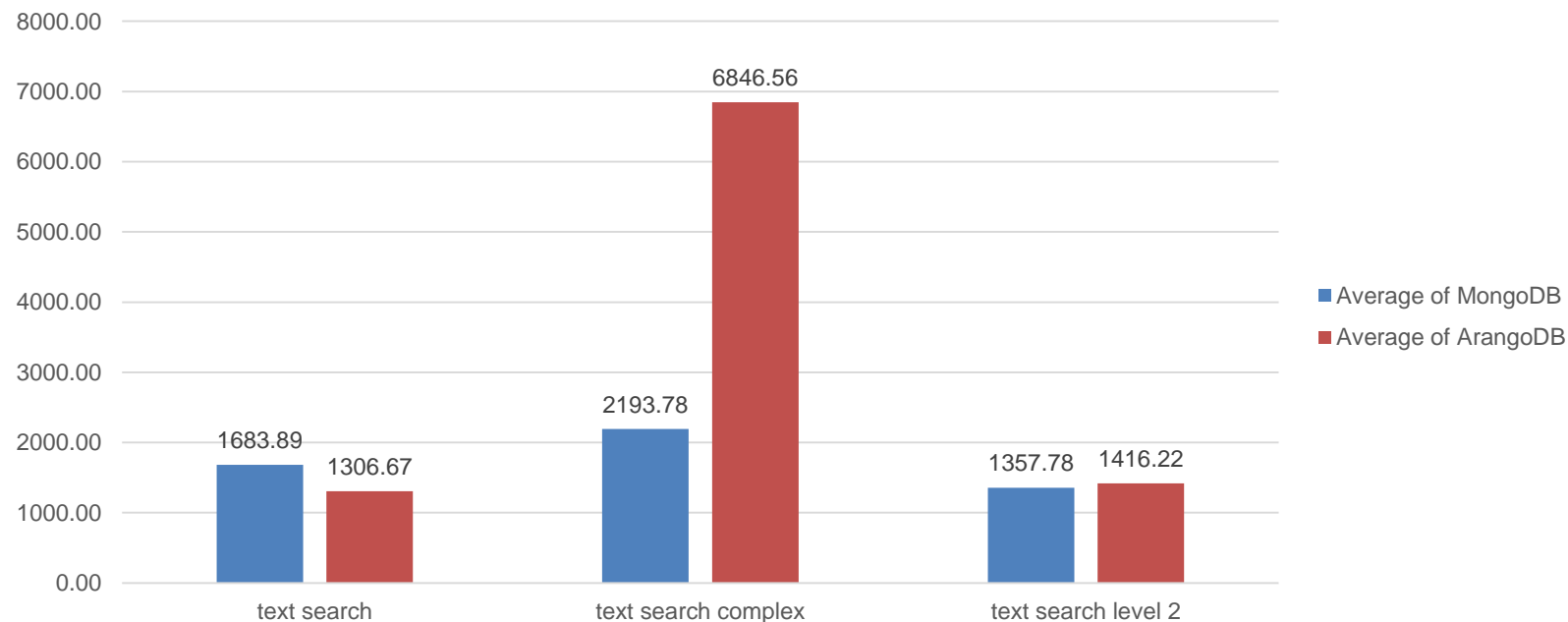
Due to the data complexity, we only completed the text search performance comparing between MongoDB and ArangoDB

We are using Infrastructure Canada Projects data

Index and search text by using Infrastructure Canada Projects Data. The publisher - Current Organization Name: Infrastructure Canada, This dataset contains a list of infrastructure projects across Canada that have been approved by Infrastructure Canada. The project information listed is based on current information.

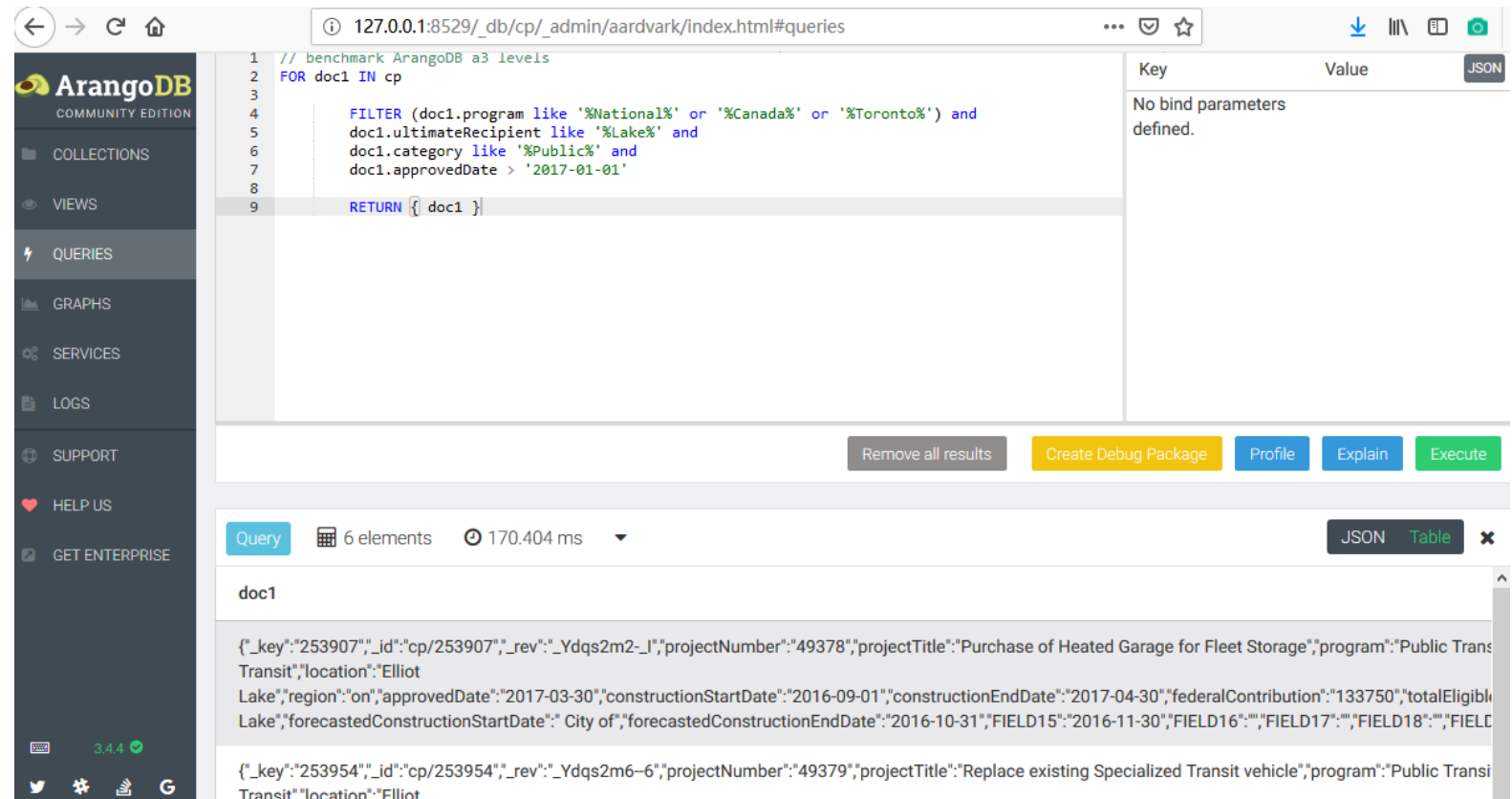
Text search – MongoDB vs. ArangoDB cont'd

As the results show, the MongoDB text search performance much better when have complex conditions, but ArangoDB performance better when search simple key words.



Text search – MongoDB vs. ArangoDB cont'd

Optimizing the storage of facts that refer to the same entity is an important aspect of the graph database enabling fast queries and inference.



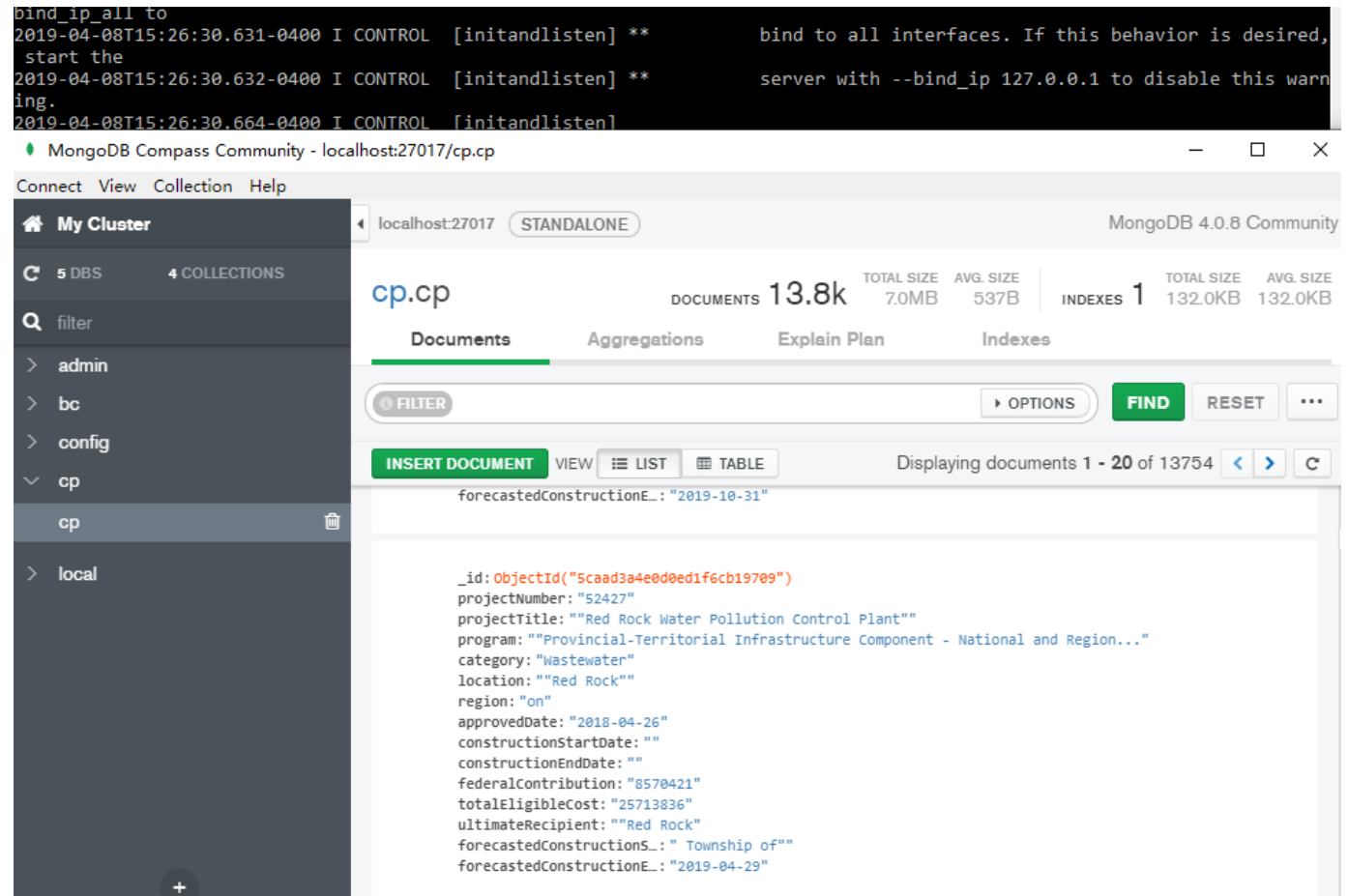
The screenshot displays the ArangoDB web interface. On the left is a sidebar with navigation options: COLLECTIONS, VIEWS, QUERIES (selected), GRAPHS, SERVICES, LOGS, SUPPORT, HELP US, and GET ENTERPRISE. The main area shows a query editor with the following code:

```
1 // benchmark ArangoDB a3 levels
2 FOR doc1 IN cp
3
4   FILTER (doc1.program like '%National%' or '%Canada%' or '%Toronto%') and
5   doc1.ultimateRecipient like '%Lake%' and
6   doc1.category like '%Public%' and
7   doc1.approvedDate > '2017-01-01'
8
9 RETURN { doc1 }
```

Below the query editor, there are buttons: "Remove all results", "Create Debug Package", "Profile", "Explain", and "Execute". The results section shows a summary: "Query", "6 elements", "170.404 ms". Below this, the results are displayed in JSON format, showing two documents (doc1) with various fields like _key, _id, _rev, projectNumber, projectTitle, program, location, etc.

Text search – MongoDB vs. ArangoDB cont'd

It allows us to develop hybrid queries that include semantic facts and full-text search within unstructured data.



Future work

- For this graph database performance benchmark, we used the same dataset and the same hardware to test each database system.
- In the future, need to check or understand better the results under different systems for different set of the data, the equipment, and the software.
- We made sure for each experiment that the database had a chance to load all relevant data into RAM. Therefore, we increased cache sizes where relevant and used full collection scans as a warm-up procedure.

Appendix - Test results - <https://github.com/yorku-itec-lab/itecprj6220>

Search or jump to... Pull requests Issues Marketplace Explore

yorku-itec-lab / itecprj6220

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

This is created for ITEC6220 Project Graph Database Benchmark: ArangoDB vs MongoDB vs OrientDB vs Neo4j

Manage topics

72 commits 1 branch 0 releases 2 contributors

Branch: master New pull request Create new file Upload files Find File Clone or download

ty7181 Add files via upload Latest commit d32e7bb 3 minutes ago

arangodb	Update arangodb_performance.txt	19 minutes ago
mongodb	Update mongodb_performance.txt	18 minutes ago
neo4j	Update README	an hour ago
orientdbgraph	Add files via upload	2 hours ago
rawdata	Add files via upload	3 minutes ago
reports	Update neo4j_performance.txt	33 minutes ago
.gitattributes	Initial commit	a day ago
ITEC6220_prj_report.xlsx	Add files via upload	22 minutes ago
README.md	Update README.md	12 minutes ago

README.md

prj6220

Appendix - Test results cont'd

yorku-itec-lab / itecprj6220

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Branch: master itecprj6220 / reports / neo4j_performance.txt Find file Copy path

ty7181 Update neo4j_performance.txt 6832f9d 2 hours ago

2 contributors

```
162 lines (161 sloc) | 5.08 KB
Raw Blame History

1 Import Execution took: 399466 ms
2 Query 1 Execution took: 962 ms
3 Query 2 Execution took: 1165 ms
4 Query 3 Execution took: 1434 ms
5 Query 4 Execution took: 2897 ms
6 Query 5 Execution took: 2277 ms
7 Query 6 Execution took: 4559 ms
8 Query 7 Execution took: 2275 ms
9 Query 8 Execution took: 2428 ms
10 Query 9 Execution took: 758 ms
11 Query 10 Execution took: 4068 ms
12 Query 11 Execution took: 759 ms
13 Query 12 Execution took: 1217 ms
14 Query 13 Execution took: 2996 ms
15 Query 14 Execution took: 3505 ms
16 Query 15 Execution took: 16018 ms
17 Query 16 Execution took: 1005 ms
18 Query 1 Execution took: 1142 ms
19 Query 2 Execution took: 1505 ms
20 Query 3 Execution took: 1468 ms
21 Query 4 Execution took: 3965 ms
22 Query 5 Execution took: 2864 ms
23 Query 6 Execution took: 6095 ms
24 Query 7 Execution took: 3143 ms
25 Query 8 Execution took: 1773 ms
26 Query 9 Execution took: 1065 ms
27 Query 10 Execution took: 3452 ms
28 Query 11 Execution took: 774 ms
29 Query 12 Execution took: 1037 ms
30 Query 13 Execution took: 3915 ms
31 Query 14 Execution took: 4397 ms
32 Query 15 Execution took: 18426 ms
33 Query 16 Execution took: 1001 ms
34 Query 1 Execution took: 1581 ms
35 Query 2 Execution took: 1168 ms
36 Query 3 Execution took: 1901 ms
37 Query 4 Execution took: 3501 ms
38 Query 5 Execution took: 3506 ms
39 Query 6 Execution took: 3506 ms
```

yorku-itec-lab / itecprj6220

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Branch: master itecprj6220 / reports / neo4j_report.txt Find file Copy path

johnql Add files via upload 3cae34 8 hours ago

1 contributor

```
238 lines (236 sloc) | 28.8 KB
Raw Blame History

1 neo4j testing started...\n
2 neo4j query 1 report...\n
3 neo4j testing started...\n
4 neo4j testing started...\n
5 neo4j testing started...\n
6 neo4j testing started...\n
7 neo4j testing started...\n
8 neo4j testing started...\n
9 neo4j testing started...\n
10 neo4j testing started...\n
11 neo4j query 1 report...\n
12 u1, u2
13 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"})
14 neo4j query 1 report...\n
15 u1, u2
16 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"})
17 neo4j query 2 report...\n
18
19 neo4j query 3 report...\n
20 u1, u2, u3, u4
21 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "5806"}, (:RateUser {RateUserId: "5606"}))
22 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "5806"}, (:RateUser {RateUserId: "5623"}))
23 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "5806"}, (:RateUser {RateUserId: "5602"}))
24 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "5806"}, (:RateUser {RateUserId: "5661"}))
25 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "2835"}))
26 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "62"}))
27 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "5859"}))
28 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "5602"}))
29 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "1529"}))
30 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "3040"}))
31 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "4519"}))
32 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "1512"}))
33 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "4611"}))
34 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "5695"}))
35 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "3744"}))
36 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "5906"}))
37 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "3470"}))
38 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "5482"}))
39 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "3893"}))
40 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "2045"}))
41 (:RateUser {RateUserId: "5929"}, (:RateUser {RateUserId: "5606"}, (:RateUser {RateUserId: "3722"}, (:RateUser {RateUserId: "2524"}))
```


Thank you