Benchmarking of Different Approaches for Objects Matching

Dmitry Frolov

Master student, Bauman Moscow State Technical University Moscow, Russia
frolov.dmtrii@gmail.com
This work is supervised by Roman S. Samarev, Associate Professor, Ph.D. Computer Systems and Networks Department, Bauman Moscow State Technical University Moscow, Russia
samarev@acm.org

Abstract. Data matching is widely used in different applications. However, very often we do not know data schema completely of the matching objects, because it might be changed during application use. As a result, objects require different processing methods, depending on their structure. To provide this, a special way of storing data should be used. For building an effective application, we need to choose DBMS suitable for selected storage data structure by measuring time spent on different operations with a high load database. The purpose of this article is to demonstrate a benchmark development which is performing these measurements and estimations.

Keywords: Benchmark, Friend of a friend, object matching.

1 Introduction

Nowadays, due to a huge amount of information, describing real-life objects, searching, comparison and choosing the most suitable object requests are becoming more and more actual. Data matching is widely used in different applications. It might be processing of social data and real-time similarity calculation between persons, recommending systems with computation of scores for different goods for a specific person and many other domain areas. The data matching issue can be easily solved when the data structure is fixed and simple, or data can be stored in memory only, but there are many issues when an object has complex, dynamically changed structure and it becomes impossible. In that case, we have to speak about using some storages and databases. Nevertheless, matching of data is a different task comparing with data fetching. Moreover, very often we do not know data schema completely of the matching objects because it may be changed during application use. As a result, the answer to which data structure should be used for matching of some objects is not so obvious. Besides, those objects require different processing methods for different substructures.

The main purpose of this work is to do assessment of different implementations of data storing for solving the task of data matching, on big data databases with complex objects.

To achieve this, we need to conduct a performance analysis of the number of database systems and provide systematic comparison of them. Among analytical modeling methods for are applicable for DBMS's are:

1. Queuing Models: Queuing models are effective to study the dynamics of a database system when it is modeled as a multi-component system with resource allocation constraints and jobs moving around from one component to another. Examples of such dynamic studies are concurrent transaction control algorithms, data allocation and management in distributed database systems etc;
2. Cost Models: Cost Models are useful in studying the cost in terms of Physical storage and query processing time. The cost model gives some real insight into the actual physical structure and performance of a database system;
3. Simulation Modeling: A simulation Modeling is more effective for obtaining better estimates since it not only analyses the database system in isolation but also can effectively analyze the database system with the application program running on top of it and the database system itself operating within the constrained environment of an operating system on real physical hardware;
4. Benchmarking: Benchmarking is the best method when multiple database systems need to be evaluated against each other but suffer from the inherent setback that it assumes all systems to be fully installed and operational. Benchmarking relies on the effectiveness of the synthetic workloads. Real workloads are non-repeatable and hence not good for effective benchmarking.

The most suitable is benchmarking method, because we need to measure a similar operation on a set of DBMS.

2 Review of Benchmarks

There are standardized benchmarks called as TPC or "Transaction Processing Performance Council" [1]. Which are widely used as benchmarks for transaction processing and database performance analysis. These benchmarks do not solely evaluate the database component of the system but rather evaluates the whole system of which the Database system is one of the key differentiating factors. The suite contains a mix of benchmarks for evaluating different performance aspects of such systems.

- TPC-C Benchmark – contains a mix of different types of concurrent transactions, a complex database, nine types of tables with different record and population sizes. It simulates the process of a multi-user environment making concurrent queries to a central database. The performance evaluation involves a thorough monitoring of all system state parameters for the correctness of update as well as performance parameters such as service time etc. This benchmark is most suitable for businesses that need a database to support online handling of orders, sell product and manage inventory.
- TPC-E benchmark – designed for evaluating database systems needed to be installed at brokerage firms. It is quite similar to the TPC-C benchmark in terms of setup and components differing only in the design of the transactions that are more relevant in

a brokerage firm environment such as account inquiries, online trading and market research, etc.
- TPC-H benchmark – is fine-tuned for decision support systems. The transactions in such environments are characterized by business intelligence intensive complex data mining queries and concurrent data modifications. The performance metric used to evaluate such systems is generally TPC-H composite query per hour.

TPC is the largest and most popular benchmarking authority, but, still there are some other benchmarks among them [2]:
- Bristlecone [3] – is a Java-based database performance testing benchmarking utility. It provides knobs to vary the system parameters for a single scenario across different sets of rules or environments. The standard run parameters for synthetic replication include a number of users, number of tables, number of rows per table, number of rows returned by queries or size and complexity of queries, etc.
- CIS benchmark [4] – is a set of security benchmark for the MS SQL Server. These benchmarks provide a testing tool to evaluate these database systems against common security vulnerabilities. Generally, while installing databases most administrators focus on key operating performance issues such as scalability, load balancing, failovers, availability, etc. and let security settings to be default factory settings.
- Yahoo! Cloud Serving Benchmark [5] – is a program suite that is used to compare the relative performance of NoSQL database management systems. The main goal of benchmark is to facilitate performance comparison of transaction-processing workloads which differed from ones measured by benchmarks designed for more traditional DBMSs. YCSB was contrasted with the TPC-H as YCSB is used for big data benchmark while TPC-H is a decision support system benchmark.

Nevertheless, previously mentioned benchmarks are not suitable for data matching measurement. The main problem is that we need to measure big data and decision operations on both SQL and NoSQL DBMS's it one program. Another problem is that we do not know data schemas of the matching objects completely because it may change during application using. In addition, those objects require different processing methods for different substructures. These facts make the development of your own benchmark relevant. The definition of data storage structures is needed for this purpose.

3 Benchmark Building

While implementing a benchmark, we consider Friend of a friend (FOAF) models as a base for our complex object representation. FOAF describes the world using simple ideas inspired by the Web [6]. In FOAF descriptions, there are only various kinds of things and links, which we call *properties*. The types of things we talk about in FOAF are called *classes*. FOAF is therefore defined as a dictionary of terms, each of which is either a *class* or a *property*. Other projects alongside FOAF provide other sets of classes and properties, many of which are linked with those defined in FOAF.

Definition of operations with complex objects

Complex object operations and requests are ones that can be made to specific objects with high probability. For the benchmark, we define the following list of operations based on FOAF object.

- Find object by a set of FOAF properties;
- Find objects by FOAF property with restrictions;
- Find an object by special property and value;
- Find similar objects.

Three groups can generalize these use cases:

1. Finding an object by certain property or group of properties;
2. Finding an object by word or phrase occurrence in property;
3. Finding a similar objects, according to all properties.

For the first group it is advisable to use simple property search with property value selection by conditions, united by logical "and", which is integrated into DBSM toolkit.

For the second group fuzzy search is the most suitable. This search type intended to preserve mistakes that are made because of the user's misprint. The search is proceeded by the entered request first, and then by several similarly written requests.

For the third group comparison on calculating generalized estimates is the most suitable. Working principle of this method is in calculating general estimate for the whole object based on local properties' estimates. Then the most appropriate objects are selected by comparison of objects' general estimates.

To make a proper program that is worked with complex objects it is needed to analyze the requests execution speed on different database sizes and models. Benchmark, measuring time spent on a single operation should be programmed to get this information.

Data models definition

As we have different DBMSs' with different logical and physical data models we need to determine most general data models but with their specifics. As FOAF models are the base for our complex objects, it is suitable to use data models containing full information about storing them. The developed data model should describe the set of object fields for each object in the database.

Three main data models have to be implemented in each test in order to compare DBMS's performance in similar situations. Nevertheless, there can be additional structures to show DBMSs' unique features, according to their specific.

1. Field data model – each object property has its own database object, containing the property value and root object identifier (see Fig. 1).

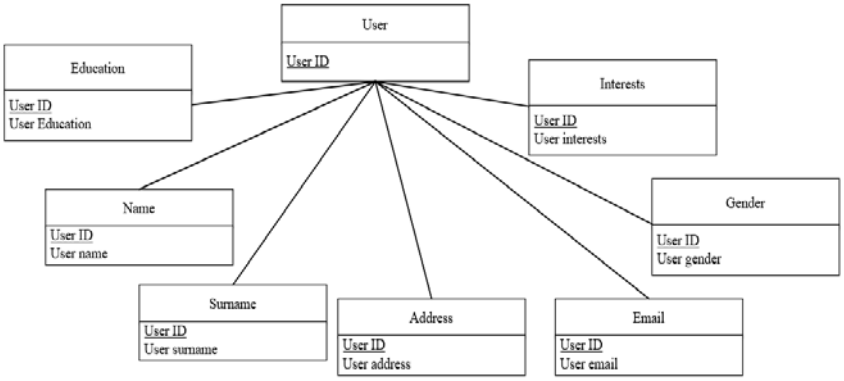

Fig. 1. Field data storage structure diagram

2. Key-value data model – each record in database is key-value pair, where the key is the name of object property and value is the value of this property for a particular object. Also, the identifier of the root object is added to this record to provide connection of the key-value pair with the root object. The example of this structure can be seen in Fig. 2.

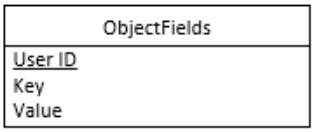

Fig. 2. Key-value data storage structure diagram

3. Tuple data model – each object has its tuple, contains key-value pair for each object property. This tuple also stored with a root object identifier (see Fig. 3).

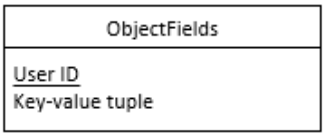

Fig. 3. Tuple data storage structure diagram

Performance measurement has to be done for three different database storage model they are relation, graph and document-oriented. The most common DBMS in their segments are PostgreSQL [7], Neo4j [8] and MongoDB [9] were chosen for analysis.

Two more data storage structures were added to the test according to selected DBMS specific.

4. Json data model – PostgreSQL has a special jsonb type for work with json objects. It stores it in the parsed binary format that helps to speed processing up.

Resulting json in the database consists of key-value pairs for each FOAF property. Fig. 4 is an example of how data stored in json


```
{
   "Name":"Faustino",
   "Email":"colton.roob@yahoo.com",
   "Gender":"male",
   "Address":"Schneiderfurt 07864 Enedina Via",
   "Surname":"Leannon",
   "Education":"Northern Washington Institute",
   "Interests":"1.33, 2.0, 2.67, 3.67, 7.0, 5.33"
}
```


Fig. 4. Data stored in json

5. Connected graph data model – connection to root project is provided by edges with property key-value pairs, making connected tree to particular property value. How data is stored in a graph is shown on Fig. 5.

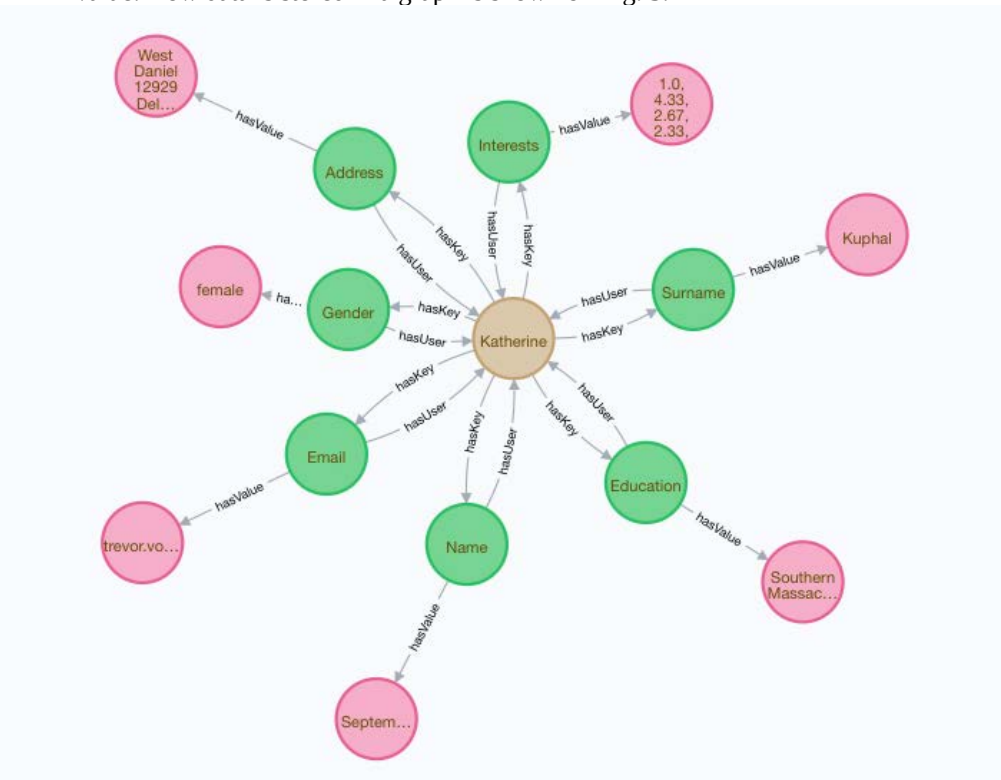

Fig. 5. Data stored in a graph

The table below shows data models distribution across the selected DBMSs.

Table 2. Data **models distribution.**

Data models	PostgreSQL	Neo4j	MongoDB
Field data model	+	+	+
Key-value data model	+	+	+
Tuple data model	+	+	+
Json data model	+	-	-
Connected graph data model	-	+	-

Benchmark architecture

Three modules “Test PostgreSQL”, “Test Neo4j”, “Test MongoDB” were implemented for performance measurement of DBMS. According to program specific, each test should implement three main object operations. The special interface should be created for this purpose, to make operation implementation in each test’s class necessary. Class subject area diagram is shown on Fig. 6.

The diagram represents following classes:

- Test – interface, containing main operation for performance measurement description;
- App – test launcher class;
- TestPostgreSQL – performance test implementation for PostgreSQL DBMS;
- TestMongoDB – performance test implementation for MongoDB DBMS;
- TestNeo4j – performance test implementation for Neo4j DBMS.

All test classes contain three general methods that were declared in Test. These methods are: *seed* – is used for generating random information and save it to selected database; *search* – is used for calling all search-methods in class; *clean* – is used to delete all test data from the database after a test run.

Each class also has specific methods used for searching data. Search queries are depended on data model presented in testing database. For example, search methods are: *testSearchByTuple, testSearchByKeyAndValueInRelations, testSearchSingleRepo.*

4 Benchmark Results

Performance test starts after launching the benchmark’s code on all described DBMS and includes tests with 1000, 10000, 100000 records in the database. During each iteration, a random data, generated by Java Faker library, added to the database. Each added object has seven main fields of FOAF representation. They are: *name, surname, address, email, gender, interests, education*. The example of the database object is on Fig. 7.

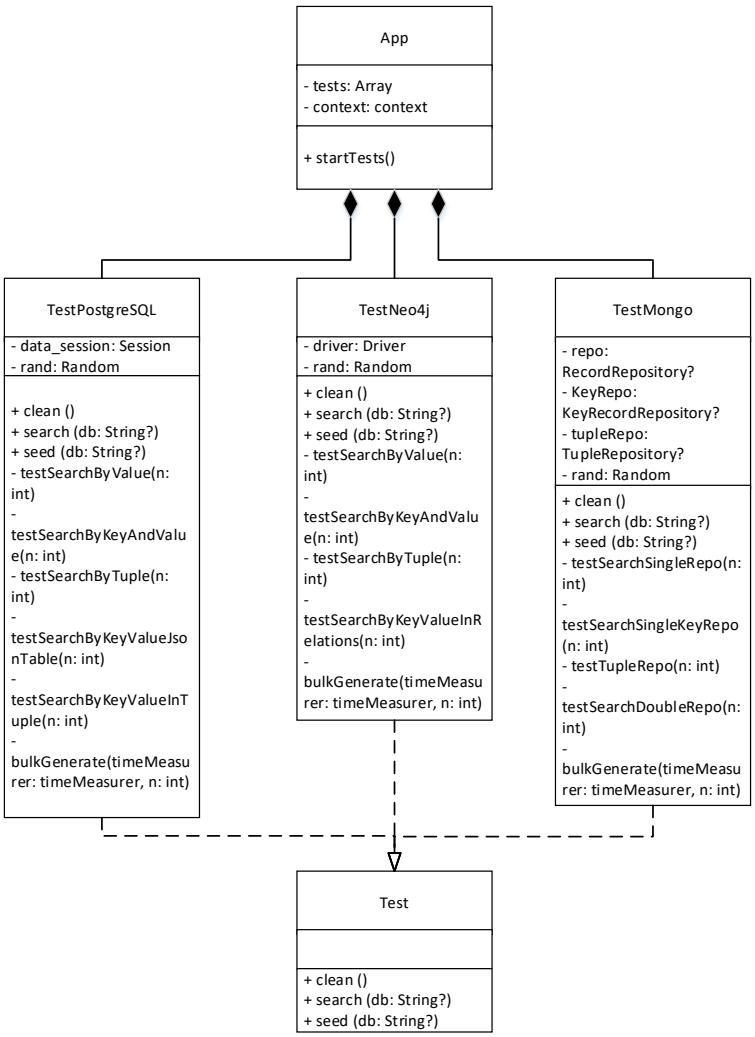

Fig. 6. Class subject area diagram

user_id	user_data
762e2f20-b9eb-42aa-914b-af4e525bbb4b	{{Name, Anthony},{Surname, Paucek},{Address, "East Marlon 32552 Ethel Spurs"},{Email,kim.reynolds@gmail.com},{Gender, male},{Interests, "1.0, 4.33, 2.67, 2.33, 7.0, 4.33"},{Education, "West South Carolina Institute"}}

Fig. 7. Database object example

10000 different searching requests are made during tests. Each search operation relates to described data models. In Table 2 this relation are set.

Table 3. Search operations and data models relations

Search operations	Data models
Search by field	Field data model
Search by key and value	Key-value data model
Search by tuple	Tuple data model
Search by key-value in tuple	Tuple data model
Search by key-value in json	Json data model
Search by key and value in relations	Connected graph data model

The performance testing results are presented in Table 3. Accepted abbreviations in the table: P – PostgreSQL 11.2, N – Neo4j 3.5.3, M – MongoDB 4.0.3.

Table 4. Performance testing results (ms)

	1000 records			10000 records			100000 records		
DBMS	P	N	M	P	N	M	P	N	M
Seed	76	5676	274	70311	40276	2167	668076	426424	21412
Search by field	1	1	1	1	1	1	1	1	1
Search by key and value	1	2	1	1	10	1	1	83	1
Search by tuple	1	1	1	1	5	1	1	47	1
Search by key-value in tuple	1	-	2	7	-	21	72	-	243
Search by key-value in json	1	-	-	8	-	-	81	-	-
Search by key and value in relations	-	8	-	-	64	-	-	176	-
Single field search in repo with dynamic key	-	-	1	-	-	6	-	-	55
Clean	9	21	190	11	6578	2756	49	2745	20252

As it is seen from the table, PostgreSQL requires more time to seed information in the database, but it has the best results in searching scenarios on every tested database size. The PostgreSQL loss on seeding information is not critical, as in real application it is impossible to get such load in a small period of time, the database size will grow constantly.

According to this research PostgreSQL and tuple data storage structure are the most suitable for developing complex object matching system.

Conclusion

This work is in progress. In the future, we will extend a number of operations with data and will change model structures. In addition, some additional matching specifics operations will be added. One of the issues, which is not considered in this paper, is benchmarking of big volumes of data in a computational cluster. In that case, huge difference in performance results might be found. This part we also will consider in further work.

References

1. TCP benchmark description, http://www.tpc.org/information/benchmarks.asp, last accessed 2019/05/27
2. Benchmarks description, https://www.cse.wustl.edu/~jain/cse567-08/ftp/db/index.html, last access 2019/05/30
3. Bristlecone, https://github.com/shutterstock/bristlecone/tree/master/src/com/continuent/bristlecone/benchmark, last access 2019/05/30
4. CIS benchmark, https://www.cisecurity.org/cis-benchmarks/, last access 2019/05/30
5. Yahoo! Cloud Serving Benchmark, https://github.com/brianfrankcooper/YCSB, last access 2019/05/30
6. FOAF specification, http://xmlns.com/foaf/spec/, last accessed 2019/05/27
7. PostgreSQL documentation, https://postgrespro.ru/docs/postgresql, last accessed 2019/03/19
8. Cypher documentation, https://neo4j.com/docs/cypher-manual/current/, last accessed 2019/03/19
9. MongoDB documentation, https://docs.mongodb.com/manual/, last accessed 2019/03/19.