Performance Study of SQL and NoSQL Solutions for Analytical Loads

Conference Paper · September 2014

DOI: 10.13140/2.1.1307.7766

2 authors:

Hristo Kyurkchiev
Sofia University "St. Kliment Ohridski"
7 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Emanuela Mitreva
Sofia University "St. Kliment Ohridski"
9 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Performance Study of SQL and NoSQL Solutions for Analytical Loads

Hristo Kyurkchiev and Emanuela Mitreva

Faculty of Mathematics and Informatics, Sofia University "St. Kliment Ohridski",
5, James Bourchier Blvd., 1164 Sofia, Bulgaria
hkyurkchiev@fmi.uni-sofia.bg, emitreva@gmail.com

Abstract. Data mining has become more popular in recent years and with it the trend of analyzing transactional data straight from the DBMS. This trend is provoking the research into read-optimized database solutions. One branch of such research – column orientation claims significant improvement in read performance in comparison with row oriented DBMS. Another branch is the growing number of NoSQL solutions. Having previously conducted research between the traditional relational DBMS and column-stores we take on studying the performance of NoSQL DBMSs in the face of MongoDB with the goal of comparing the three approaches. The first step in this is examining their data models. We then use our previously developed benchmark to measure each DBMS's performance. Evaluating the results we draw conclusions about each DBMS's suitability and main advantages over the other.

Keywords: database systems, relational databases, data warehouses, NoSQL, column stores, C-store, MongoDB, performance evaluation, analytical queries

1 Introduction

Mining transactional databases has become increasingly popular in recent years. Thus the research into database management systems (DBMS), which can provide fast performance for analytical queries, has great significance to the business world. One such DBMS is Abadi's Vertica [7], the commercialization of C-Store [2, 3, 13]. It is a column store (a DBMS, which stores data in columns, rather than rows), which provides both the standard SQL language for querying databases, and the performance needed for effective data mining.

Since several performance studies [1–3, 5, 13] have been carried out using C-Store and a custom build row store DBMS and some comparing C-Store and Vertica [7] there was little information on how these databases fare against commercial grade RDBMS solutions such as Oracle. Furthermore, most of the results were achieved by using either an implementation of the TCP-H [1, 5, 7, 13] or the Star schema benchmark [3], a benchmark commissioned [9] by Stonebraker, one of Vertica's creators. This posed some reasonable doubts about the validity of the results.

Thus in previous research [6] we aimed at filling this gap by researching how Vertica compares to Oracle – a commercial grade row store. We found out that by using Vertica instead of Oracle one achieves significant performance gains, which however, are not in the same order of magnitude as previously suggested [16].

Stepping on these findings we decided to widen the research scope and as a logical next step to include other alternatives to traditional relational DBMS in this comparison. The natural choice for such solutions is the modern NoSQL databases [12], as little information about how they fair against the relational world as far as online analytical processing (OLAP) is concerned.

2 Choosing a NoSQL Solution for the Comparison

In order to make an educated choice of which NoSQL solution is the most appropriate for a data warehousing solution one has to critically look into their characteristics. And although they share one thing in common – not being relational – they are very different considering almost all other aspects [12]. Thus the first step in choosing would be to split them in homogenous categories and analyze each of them sequentially.

One way to categorize the NoSQL databases is by the way they store data:

- key-value stores – they save the data as key-value pairs (having the ability to search by the key and supports only indices on the key) [10];
- column-oriented databases – they store the data in the form of one extendable column of closely related data [10];
- graph stores – they keep the data in graph structure – by using nodes and relations [8];
- document stores – as their name suggests, they save the data as documents as either using XML or JSON [14].

Key-value stores and column-oriented databases do not offer the necessary flexibility and expressivity needed in a data warehousing solution, thus are not suitable for our experiment. This leaves graph and document stores. And while both are valid options and graph stores are more expressive and flexible [4], we chose MongoDB, a document store, for the comparison because of its simplicity, scalability, community support, provision of easy data access and performance [11].

3 Architectural Overview

3.1 MongoDB Data Model

MongoDB was created with the idea of handling large amounts of data, which is even suggested by its name (huMONGOus) [11].

Logical and Physical Database Structure. In MongoDB the data is stored as either XML or binary JSON (BJSON) [14]. The databases in it are schemaless – they do not require defining the structure of the data prior to its importing, but are created the moment some data is stored in them. Instead of traditional relations or tables, they have the so-called collections, which are not stored in rows and column, but rather in JSON dictionaries (Fig. 1). MongoDB allows for great flexibility – the number and type of the attributes or fields, as they are referred to, can vary significantly between the records, which provides for easy extensions to the database, but at the same time makes it easy for mistakes to get inserted into the structure. Another feature is documents' (records) nesting, which is unlimited in level, but can prove to introduce difficulties for DML queries.

```
{
      "_id" : 0,
      "name" : "John Doe",
      "scores" : [
              {
                    "score" : 1.24345355,
                    "type" : "exam"
              },
              {
                    "score" : 12.4983573,
                    "type" : "quiz"
              },
              {
                    "score" : 34.2434535,
                    "type" : "homework"
              }
       ]
}
```

Fig. 1. Sample JSON document

Read Optimization. MongoDB provides a set of features to optimize read performance – query optimizer, indices, parallelization, etc. It supports various types of indices – single field indices, compound indices, multikey indices, geospatial indices, text indices, hashed indices [15].

3.2 Compared to Vertica and Oracle

There are both similarities and differences between MongoDB, Oracle, and Vertica. A comparison of the main ones is presented in Table 1.

As seen from the table above, there are several significant differences between the three systems, starting with the physical data model, which has significant impact over the other features of the DBMSs, such as replication, sharding, etc.

Table 1. Basic comparison between MongoDB, Vertica and Oracle

Characteristic	MongoDB	Vertica	Oracle
Data storage	XML or (B)JSON	Columns	Tables, rows
CAP	Consistency and Partition tolerance	Consistency and Availability	Consistency and Availability
ACID rule	No	Yes	Yes
Transactions	No	Yes	Yes
JOINs	No	Yes	Yes
Indices	Supports single and compound indices on every level of the JSON	Does not support indices at all, uses projections for optimization	Supports single and compound indices on all columns
Replication	Yes	Yes	Yes
Sharding	Yes	Yes	Yes

Another significant difference, especially when data warehousing is concerned, is that MongoDB does not support joins [15]. The work around for this is to store all the data in one single collection by pre-joining the tables and using nesting if necessary. This can be considered, as a denormalization in the eyes of the relational world and in most cases is prone to leaving the data in an inconsistent state at some moment. Of course, data can be stored in several separate collections and joined by the source code of the application that is using it. Validating whether this limitation results a performance overhead on the part of MongoDB is something we would be discussing in the later sections of the study.

On the similarities front, all systems follow the CAP theorem [10], which states that only two of the three (Consistency, Availability and Persistence) can be supported. For MongoDB these are consistency and partitioning, while for Vertica and Oracle they are consistency and availability [14]. These characteristics result in Oracle and Vertica's support for the ACID properties and transactions and MongoDB's lack thereof.

Based on this analysis, our hypothesis is that MongoDB would be on par with Oracle and Vertica, when making DML queries (insert, update, delete), but would be less efficient when it comes to OLAP processing due to its lack of join support and possible code overhead.

4 Experiment Setup

In this section the main aspects of the experiment such as the hardware and software, the database schema, the benchmark, and the measuring tools are discussed. Only a part of the original database schema is presented, as it is proprietary information.

4.1 Setup Description

In order for the comparison to be valid the same setup as in the original performance study [6] is used. For completeness, we have included a brief outline bellow.

Hardware Setup. The DBMSs were run as virtual machines on VMWare ESXi, each equipped with two CPU cores (2.4 GHz each), 6 GB of RAM and 16 GB of storage.

Database Setup. No specific tweaks have been performed on each database to improve performance, except the ones, which are implicit or completely trivial:

– For Oracle, Oracle 11g SE One was used with implicit indices on all of the surrogate keys, as well as explicit indices on all of the foreign keys.
– HP Vertica Community Edition was used as the Vertica instance with the compulsory super projection on each table.
– The latest version of MongoDB (2.6.1) was installed on server with indices on the same columns as the ones in Oracle.

Database Schema. The same schema as the one used in previous research [6] is employed here as well – three dimension tables and one fact table. Not changing the schema proved to be the cause of some issues that we will discuss later on, when we present the results, due to the fact that MongoDB does not support joins.

The data used is from a travelling agency and concerns autobus. It includes autobus data ($\sim$ 50 records), station data ($\sim$ 5000 records), trips data ($\sim$ 100 000 records), and stops data ($\sim$ 1 750 000 records), which has been transferred to MongoDB without any changes.

4.2 Benchmark

For the results to be comparable to the ones previously measured for Oracle and Vertica the same benchmark suggested in previous research was used [6]. Of course, since the queries were originally written in SQL and MongoDB does not support SQL for querying data, they had to be rewritten in JavaScript – the native query language of MongoDB. Since the elementary queries are straightforward to translate and it would take too much space to include all of the others, only a model approach is presented here. Essentially the rewriting of each join is done by first reducing the *Trips* table based on its ID (no actual reducing done) so that the appropriate computation can be done (MongoDB does not support computations inside an aggregate function). Then performing the necessary aggregations on this newly made collection. And finally "*joining*" it with the other tables as required by using a couple of MapReduce calls. The resulting collection was then used in a simple query to get the end result. Later on in the study, it will become obvious that it is not the actual "*joining*" that is slowing the process down, but rather the preparation of the first and largest collection – *Trips*. Another trade-off with this approach for translating the SQL queries into MongoDB ones is that in order to make the needed result set, on which the actual query will be executed, additional temporary collections are created – a copy of the *Trips* collection, prepared to be joined by the other collection. This not only takes time to prepare and implement, but it also takes a great deal of resources, which after the query is performed, should be cleared.

To make things worse, appropriate mapping and reduce functions should be prepared and applied in a specific order to get the desired result. This makes the usage of MongoDB more than error prone and hard on the developer. We should note, however, that the idea behind not including joins in MongoDB is based on the idea of using nested document structures instead, which we are not using in order to remain true to the original schema and experiment setup.

In addition to this disadvantage we also were unable to rewrite the fourth flight of queries using this or any other approach due to MongoDB's query language being limited and due to the queries requiring joining of relations with different kinds of relationships between them (e.g. one-to-many and many-to-many). Thus this flight was not implemented and run on top of MongoDB.

5 Performance Comparison

Aqua Fold's Aqua Data Studio was used to measure the response time and Fig. 2 and 3 were generated using Shield UI. All queries have been run multiple times and the results were averaged so that any differences are smoothened. It should be noted that for response times of less than 1 s the measured results varied significantly on MongoDB.

5.1 General Queries Performance Analysis

The result patterns seen in Fig. 2 for the DML statements are on par with Oracle and Vertica with some peculiarities, e.g. the result for the insert in stations and the delete of an autobus, which however, can most likely be attributed to measurement error. As expected all DML statements take longer on the MongoDB DMBS than on Oracle and Vertica and while we see some uniformity in the results of the other two DBMSs as far as the same kind of statement is performed on a different relation, with MongoDB this is not the case – the results are very unpredictable.

The second part of this flight consists of general select statements. In contrast to the DML statements, here there is a very distinct pattern, which matches the one we saw in Vertica in the previous study [6]. Again, the results show that MongoDB is significantly slower than the other two systems, with the difference reaching 11 fold. It is interesting to note that there is one occasion when MongoDB outperforms, although with a narrow gap, both Oracle and Vertica and this is when all stations are selected. This difference is most likely due to the size of the relation.

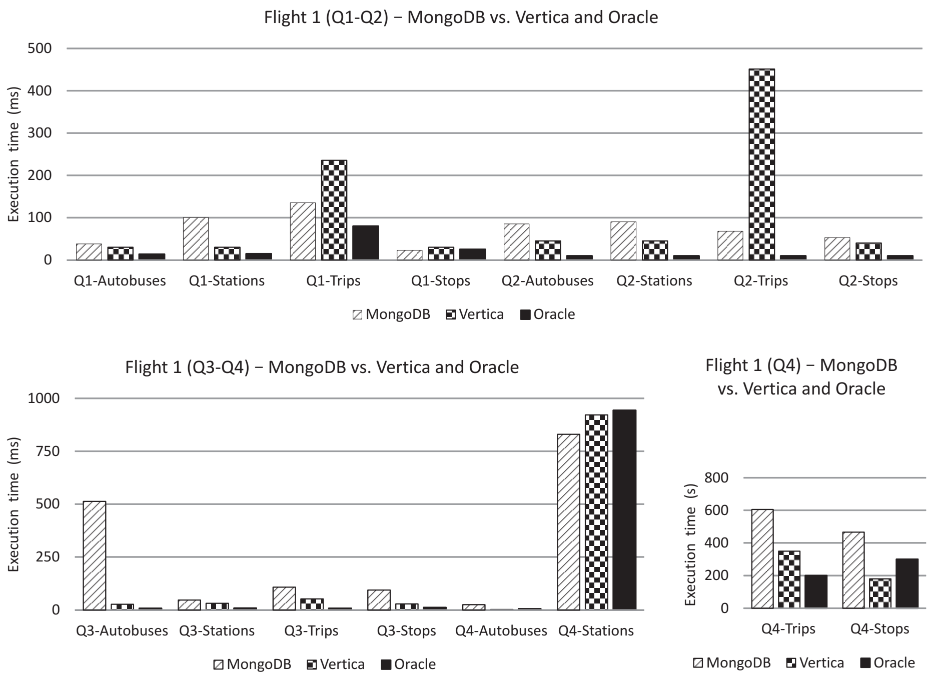

Fig. 2. General queries results

5.2 Analytical Queries Performance Analysis

We ran the analytical queries (flight two and three) several times performing some optimizations with each subsequent run (see Fig. 3).

After the initial tests for the second flight, which showed about 25 times slower responses than Oracle and close to 85 times slower than Vertica, corrections were made on the map functions used for the queries by adding some preliminary filtering on the data, which is inserted in the temporary collection.

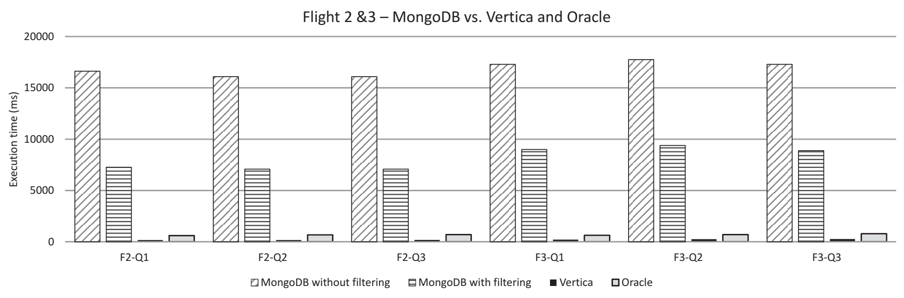

Fig. 3. Flight 2 and 3 results

And since we have already proven that the DML operations are slower than the other two DBMSs, this slowness is propagated to the other queries due to the necessity of building temporary collections (especially if no filtering is done on the data).

The initial test results show that if there is no filtering applied to the data used to build the new collection upon, it took around 16–17 s for the query to be executed. If the data is filtered beforehand (thus limiting the number of inserts to be made during the MapReduce functions), the query execution time is decreased to 7–8 s (the time is cut by 50%).

These results were still more than unsatisfactory – 10 times slower than Oracle and 30 times slower than Vertica for a NoSQL solution, designed to handle large amounts of data such as MongoDB.

The next step in measuring the performance was to make a new denormalized collection by nesting the other three collections (*Autobuses*, *Stops*, and *Stations*) within it. Having just one collection allows the skipping of the MapReduce functions and achieving the result with a simple non-join query. This resulted in execution times of 2–3 s. While it still cannot compare with the results of Oracle and Vertica, but was of 8–9 times better than the initial test results for MongoDB.

The situation with the third flight of queries is similar to the second one. The difference here is that we have one more additional collection in the join. This proves to not have such a great effect on the query execution times as they are increased with just 1 s to 17–18 s without filtering and a little over 8 s with filtering, which makes it again more than 10 times slower that Oracle and 30 times slower than Vertica. Apparently, as noted previously the biggest performance hit is due to creation of the temporary collection in which the records from *Trips* are inserted after the first MapReduce functions are executed.

6 Conclusion

During the work with MongoDB we have discovered that it is very easy to work with as far as data loading is concerned. This, however, was not the case for querying data. It provides a limited language, which expressive properties are far behind the ones of the well-known and widely used SQL language. The main omission – the lack of joins and their substitution with denormalized relations, although appropriate for the data warehousing solutions and providing some performance benefits, make for a bigger database and are not a subject of our study as we aim at looking into transactional databases, used for analytical purposes. Comparing the performance results of MongoDB with the previously recorded times of Oracle and Vertica, show that it cannot match them in any of the tested scenarios. What is more, it lags significantly behind. Thus we can conclude that for analytical queries Vertica still remains the best choice among the three. Another contender can be the graph databases, which we intend on studying in future research.

Acknowledgments. This paper is supported by the European Social Fund through the Human Resource Development Operational Programme under contract BG051PO001-3.3.06-0052 (2012/2014) and by the Sofia University "St. Kliment Ohridski" SRF under Contract 43/2013.

References

1. Abadi, D. et al.: Integrating Compression and Execution in Column-Oriented Database Systems. In: Proc. 2006 ACM SIGMOD Int. Conf. Manag. data – SIGMOD'06. pp. 671 (2006)
2. Abadi, D.J. et al.: Column-Oriented Database Systems. In: Proc. VLDB '09. pp. 1664–1665 (2009)
3. Abadi, D.J., Madden, S.R.: Column-Stores vs. Row-Stores?: How Different Are They Really? In: SIGMOD'08. pp. 967–980 (2008)
4. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. 40(1), 1–39 (2008)
5. Harizopoulos, S. et al.: Performance Tradeoffs in Read-Optimized Databases. In: Proc. VLDB Endowment. pp. 487–498 (2006)
6. Kyurkchiev, H., Kaloyanova, K.: Performance Study of Analytical Queries of Oracle and Vertica. In: Proc. 7th ISGT Int. Conf. pp. 127 – 139 2013)
7. Lamb, A. et al.: The Vertica Analytic Database?: C-Store 7 Years Later. In: Proc. VLDB Endowment. pp. 1790–1801 (2012)
8. Mitreva, E., Kaloyanova, K.: NoSQL Solutions to Handle Big Data. In: Proc. Doctoral Conference in MIE. pp. 77–86 (2013).
9. Neil, B.O. et al.: The Star Schema Benchmark and Augmented Fact Table Indexing Outline of Talk. In: Nambiar, R., Poess, M. (eds.) Performance Evaluation and Benchmarking. LNCS, vol. 5895, pp. 237–252. Springer, Heidelberg (2009)
10. Padhy, R.P. et al.: RDBMS to NoSQL: Reviewing some next-generation non-relational database's. Int. J. Adv. Eng. Sci. Technol. 11, 15–30 (2011)
11. Redmond, E., Wilson, J.R.: Seven Databases in Seven Weeks. The Pragmatic Bookshelf (2012)
12. Scholz, J.: Coping with Dynamic, Unstructured Data Sets – NoSQL: a Buzzword or a Savior? In: Schrenk, M., Popovich, V.V., Zeile, P. Proc. REAL CORP 2011 May, 1–9 (2011)
13. Stonebraker, M. et al.: C-store: a Column-Oriented DBMS. In: Proc. 31st VLDB Conference. pp. 553–564 (2005)
14. Zaki, A.K.: NoSQL Databases?: New millenium database for big data, big users, cloud computing and its security challenges. 403–409 (2014)
15. MongoDB Manual. `http://docs.mongodb.org/manual/`
16. The Vertica® Analytic Database Technical Overview White Paper (2010)