International Journal of Computer Applications (0975 - 8887)
Volume 153 - No.8, November 2016

A Preliminary Study of OLAP Queries under different Database Models

Cesar de Carvalho
CEFET-RJ
Rio de Janeiro, Brazil

Eduardo Ogasawara
CEFET-RJ
Rio de Janeiro, Brazil

Ana Beatriz Cruz Silva
CEFET-RJ
Rio de Janeiro, Brazil

ABSTRACT

From the continuous growth of data that arises in this new era of Big Data, the old assumption of one size fits all solutions is no longer valid. There is a huge effort in development alternatives for relational model. Generally, the study of these databases models targets in providing solutions that increase performance of different applications. For example, in nowadays applications, such as Big Table analysis, analytic queries typically encompass aggregations of huge datasets. To allow for data analysis to occur in a feasible time, it is necessary for database systems to offer good performance in ETL (extract, transform, and load) operations. This paper briefly presents the performance of some representative database models in addressing a set of analytical queries.

Keywords

OLAP queries, Big Data, Benchmark, Relational databases, Column-oriented databases, Document-oriented databases

1. INTRODUCTION

The amount of data produced worldwide has grown in an expressive way characterizing the data flood phenomenon. In 2010, the zettabyte barrier was broken and, in 2011, the volume of existing data on internet was 1.8ZB [1]. The advent of technologies capable of collecting large amount of data in different formats and precisions promote the new era of Big Data [2].

Facing the challenge of storing the collected data, the alternatives that could increase the scalability and decrease the computational cost of the database management systems became objects of study. Observing the advent of exploring different database models, characterizing the new NoSQL era [3]. These new database models, commonly gives up one of the properties covered by the CAP theorem (*consistency, availability, partition tolerance*) [4] in order to achieve the desirable performance [5].

A representative example of such Big Data scenario [6] includes the observation of logistics/transport systems, such urban mobility. It generates a big volume and variability of data [7] monitored. In order to enable the analysis and studies of urban mobility data, it is necessary to support analytic processing over collected data set [8]. Online analytical processing (OLAP) commonly involves queries with aggregations and joins [9] in order to enable data analytics. Due to the different natures of Big Data, adopting a particular database model, therefore, can directly influence in the execution time of the analytic queries, affecting, consequently, its performance [10].

This article is a preliminary study that aims to evaluate the support for OLAP queries in different database models. In additional to relational database models and column-oriented, also evaluated by [10], in this article was also analyzed the document-oriented database model. The exploration of these models was made through the execution of typical analytic queries patterns initially instantiated. From this preliminary study, it is aimed to apply this general querying processing results to support relevant new issues such as urban mobility and spatial-time series.

Besides this introduction, this article contains four more sections. Section 2 presents a brief explanation about the main features of the three database models discussed and OLAP queries. Section 3 describes the *benchmark* used, the adopted queries with their respective justifications. Section 4 describes the way in which different database models were evaluated, and results obtained from the experiments. At last, Section 5 presents the conclusion of the work.

2. DATABASE MODELS AND ANALYTIC QUERIES

Each database model covered in this article has specific features, that are best suited to a given scenario. In this section, such features are described, highlighting the contexts in which it is expected that they stand out. Moreover, are presented the types of operations common to OLAP queries, regardless of the domain.

Relational Databases (RDB) are composed of a collection of tables. Each table has one or many attributes, represented by columns. Each row of the table, in turn, represents a relationship between the values of the table attributes [11]. This database is traditionally transactional. In this way, they are optimized to support queries that affects a small portion of the database [12].

Column-oriented databases (CDB) are also composed of a collection of tables. However, unlike the relational database, its attributes are represented in rows, and its values are stored in columns [13]. Due to this structure, the CDB is optimized for queries that scan a limited set of attributes [9, 10].

In document-oriented databases (DDB), the unit of data is a document and the set of documents is a *collection* [14]. Unlike RDB and CDB, documents do not have default attributes nor fixed size. They are composed of pairs of *key-value* [15] objects. Moreover, nested documents and arrays are allowed. This distinct structure allows them to represent more complex data relationships, such as hierarchies [16].

Despite the need for performance in the process of data storage in many of nowadays applications, the demand for support to analytic queries is generally necessary. Besides the aggregations and several joins, OLAP queries include *rollup*, *drill-down* and *pivot* operations. These operations are based in different levels of aggregations, selections, projections, and transpositions [17] over a large data set. Such operations are often required regardless of the explored domain.

3. METHODOLOGY

In order to enable a preliminary performance analysis of the database models presented in section 2, it was necessary to adopt a framework containing data and representative queries, *i.e.*, that simulates a real problem. This simulation was done through the instantiation of a subset of TPC-H *benchmark* mapped to different database models. Three databases were adopted, each one representing a model discussed in the article.

3.1 TPC-H

The TPC is a non-profit organization that provides benchmarks for testing the databases and transaction processing performance. Transactions are operations that include reading and/or writing to disk, calls to operating systems or data transfer between systems [18]. The TCP Benchmark H (TPC-H) is designed to decision support systems that examine a big volume of data and run complex analytic queries [18]. The database structure is shown in (Figure 1). The benchmark provides data for populating tables and analytic queries, and are made available by the organization.

3.2 Queries

The TPC-H defines twenty-two decision support queries and instructs ways to execute and vary them. From the twenty-two queries defined by the TPC-H, four were selected to be run on the three selected databases. Four queries were selected as they present a set of query features (Table 1) to be explored in different database models.
Query *Q1* (Figure 2.a) corresponds to the prices report. It presents a report of items billed, delivered, and returned on a given period. *Q1* access just one table and uses aggregation (average and counting). Applying the period indicated by the benchmark as a query filter, it is expected that 95% to 97% of the table content to be scanned. This type of query, without join and with access to a big data volume, is very common in many of nowadays applications (*Big Table* modeling) [20].
Besides the parameters specified by TPC-H (*Q1.1*), some variations of this query were executed. The first one (*Q1.2*) uses just one delivery day as selection criteria, while the second one (*Q1.3*) considers a specific day and hour for the selection. While query (*Q1.1*) does selection by period, in queries (*Q1.2*) and (*Q1.3*), it is intended to verify the performance of the databases when running range queries. The third variation (*Q1.4*) selects data by key. Although these types of queries are not described by TPC-H documentation, they were included to better explore database models.
Query *Q4* (Figure 2.b) verifies the order priority. It returns the number of orders in the period of a quarter, where at least one item has been received by the customer after it is confirmation. The results set is grouped by date and ordered by priority. This query makes possible to define if the priority order system is working in a satisfactory way. With this query, it is intended to mainly explore the behavior of existential queries in databases.
Query *Q13* brings the distribution of customers (Figure 2.c). It has the goal of searching a relation between customers and their orders. Special categories of orders are filtered by the query. The special categories of orders are identified by the content of the attribute relative to the order comment (o_comment). This query is particularly interesting for exploring joins (*left join*) and aggregations.
Finally, query *Q14* analysis the effect of promotions (Figure 2.d) in sales. This query monitors the market response to promotions and special campaigns, computing the revenue (l_extendedprice * (1-l_discount)) of a given month when influenced by promotions. This query explores advanced conditional evaluation during aggregation. In this context, Table 1 presents a summary of features explored in each selected query.

4. EXPERIMENTAL EVALUATION

4.1 Databases

The TPC-H *benchmark* was applied to the databases models described in section 2. PostgreSQL, MonetDB, and MongoDB databases were also selected to respectively represent RDB, CDB, and DDB. In this way, it is possible to explore the performance of different databases in the context of OLAP queries.
PostgreSQL is a traditional relational Database Management System (DBMS), free and *open-source* [21]. Like other relational databases, PostgreSQL supports the traditional transactions mechanisms including the ACID properties. The query language used by this DBMS is SQL.
MonetDB is also free and *open-source* column-store pioneer Database Management System [22]. Although its structure is different from the structure of RDB, MonetDB has some similarities to this database, such as support for SQL language and ACID properties. MonetDB also supports operations and resources common to RDB, such as foreign keys and joins. In addition, the system has query execution plan that intensively use RAM memory and dynamic optimization techniques.
MongoDB is a DDB [23]. MongoDB documents are serialized as JSON (*Javascript Object Notation*), and stored using the JSON BSON binary encoding [24]. Unlike the previous databases, MongoDB is not compatible with SQL. It has a syntax that provides navigations in document hierarchy, being closer to the JSON syntax. In addition, MongoDB gives up on resources common to RDB, like *join* [16], in favor of complex documents storage.

4.2 Instantiation of the TPC-H

Initially it is necessary to create the structure defined by TPC-H (Figure 1) for the three databases. The CDB and RDB representations makes use of SQL, what causes the *scripts* for creating the tables, used by the two databases, be very similar. The DDB representative, in turn, uses a differentiated language, what causes the *script* for creating the *collections* different from the *scripts* used by other databases.
The data for populating the tables are provided by the TPC. The format of the provided files is *TBL*, making it possible to import the data to RDB and CDB. To import data into DDB, however, it is necessary for the input file to be in *JSON* format. Thus, data was converted from *TBL* to *JSON*. The file obtained through this process was then imported to the DDB following the architecture proposed by the TPC-H. The scale factor adopted was 1.

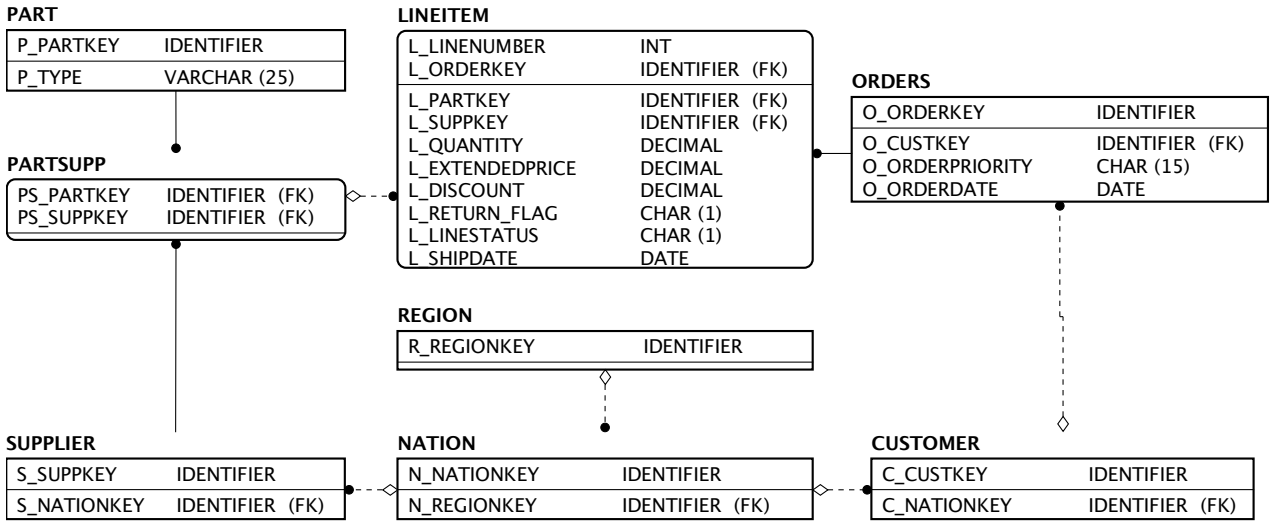

Fig. 1. Entity Relationship Diagram of TPC-H (Adapted from TPC Benchmark H specification document [19])

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price, …,
  count(*) as count_order
from lineitem
where l_shipdate < = date '1998-12-01' -interval '[DELTA]' day (3)
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

(a)

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price, …,
  count(*) as count_order
from lineitem
where l_shipdate < = date '1998-12-01' -interval '[DELTA]' day (3)
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

(b)

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price, …,
  count(*) as count_order
from lineitem
where l_shipdate < = date '1998-12-01' -interval '[DELTA]' day (3)
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

(c)

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price, …,
  count(*) as count_order
from lineitem
where l_shipdate < = date '1998-12-01' -interval '[DELTA]' day (3)
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

(d)

Fig. 2. Queries used for performance evaluation of the different database models

Table 1. Relation between the queries and operations explored by them

Feature	Q1.1	Q1.2	Q1.3	Q1.4	Q4	Q13	Q14
Exists					X		
Nesting						X	
Conditionals						X	X
Selection by key				X			
Selection by date		X	X				
Joins						X	X

4.3 Execution of the Queries

The access to the three databases and the execution of the four queries were automated in *Python*. This automation makes it possible to use the same parameters on the three databases so that the query returns the same result. Each selected query was executed thirty times and, except for the query *Q1.1*, for each time that the queries were executed, their parameters were randomly selected. The average runtimes can be seen in Figure 3.

Query *Q1.1* was executed faster by the CDB, with an average difference of the query runtime greater than 10 seconds when compared to the time required by the other databases. DDB had a small advantage over RDB, when it is considered the average time of the query executions. However, the first execution of the query had a

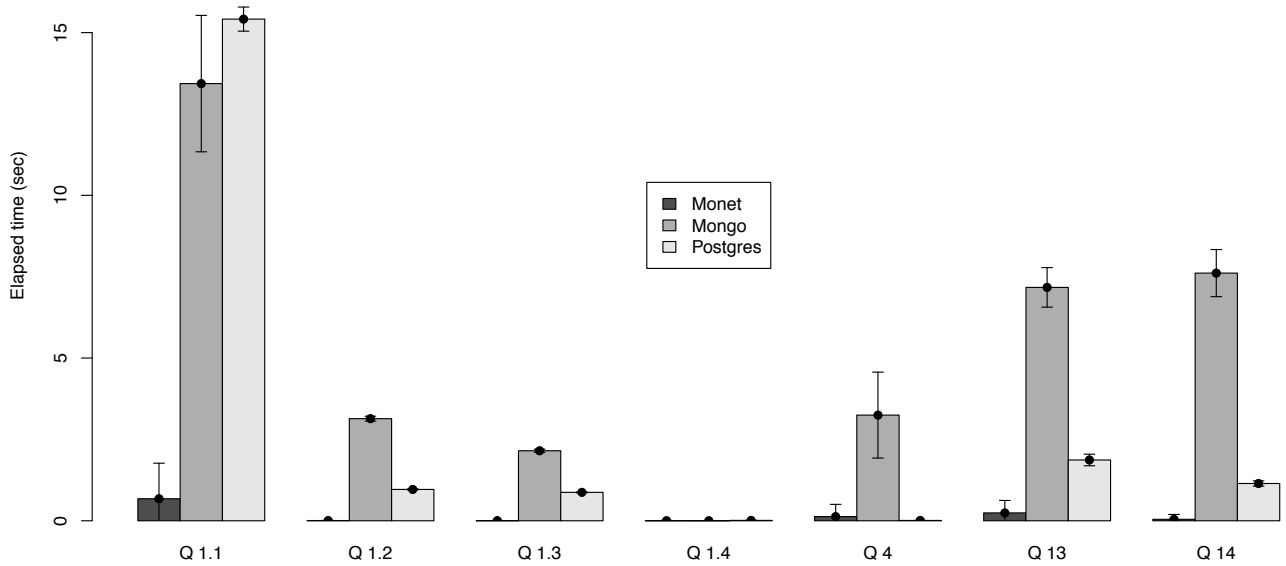

Fig. 3. Average elapsed time for runtime queries

runtime of 24,1 and 16,2 seconds for DDB and RDB, respectively. From the second execution, the runtime of the query on DDB had a significant reduction, while RDB showed a much inferior reduction. This result indicates that for recurring queries, without joins and with access to a big data volume, the DDB can have a better performance than the RDB. In both situations, CDB had a better performance.

Queries *Q1.2* and *Q1.3* also had an inferior runtime when executed on CDB. However, unlike the previous query, in these two queries, RDB had a better performance than DDB. A factor that justifies this result is the high selectivity of the queries. The high selectivity is also a characteristic of the query *Q1.4*, but, in contrast to the result of the queries *Q1.2* and *Q1.3*, DDB had shorter runtime, with a subtle difference when compared to the runtime of the query on CDB. RDB had the worst performance. This result can be justified by the use of index as selection criteria.

RDB excelled in query *Q4*, which uses the *exists* operation. CDB took an average of one-tenth of a second longer in the execution time of this query, when compared to RDB. DDB had the worst performance, with a difference of more than 3 seconds when compared to other databases. This big difference of DDB in relation to other databases, was, probably, influenced by the absence of existential clause on DDB. The behavior of this clause was implemented in the application layer.

Queries *Q13* and *Q14* were executed faster by CDB, followed by RDB, and DDB. The two queries explore sorts and joins, which are operations that do not exist in DDB. The join operation was done in the application layer, what may have influenced the big difference in the runtime. The difference from the CDB to RDB is more than 1 second of difference in their average runtime.

5. CONCLUSION

In the scenario of Big Data, it is mandatory that OLAP queries run quickly. OLAP queries commonly have aggregations, join, projections, and transpositions. In order to preliminary evaluate such representative access patterns in different database models, the *benchmark* TPC-H was adopted. The experimental evaluation performed, points out which databases models were more efficient in the execution of representative queries of TPC-H *benchmark*. In the evaluation, it was predominantly considered the runtime of the queries. From the obtained results, it can be seen that the CDB had, in general, better performance in processing of OLAP queries. On the other hand, the results obtained with the DDB pointed a potential deficiency of the document-oriented model when subjected to the scenarios proposed by the TPC-H *benchmark*. For a wider analysis, other database models needs be included in the study and OLAP queries from different applications must be explored. Among the models to be explored, it can be mentioned trajectory and spatial-temporal series, both relevant to support problems of logistics/transport such as urban mobility.

Acknowledgments

The authors thank to CNPq, CAPES, and FAPERJ for partially funding this research.

6. REFERENCES

[1] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[2] HV Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014.

[3] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

[4] Eric Brewer. Cap twelve years later: How the" rules" have changed. *Computer*, 45(2):23–29, 2012.

[5] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. Nodb: efficient query execution on raw data files. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 241–252. ACM, 2012.

[6] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137–144, 2015.

[7] Shashi Shekhar, Viswanath Gunturi, Michael R Evans, and KwangSoo Yang. Spatial big-data challenges intersecting mobility and cloud computing. In *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 1–6. ACM, 2012.

[8] Nikos Pelekis and Yannis Theodoridis. *Mobility data management and exploration*. Springer, 2014.

[9] Hasso Plattner. A common database approach for oltp and olap using an in-memory column database. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 1–2. ACM, 2009.

[10] Daniel J Abadi, Samuel R Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980. ACM, 2008.

[11] Abraham Silberschatz, Henry F Korth, S Sudarshan, et al. *Database system concepts*, volume 4. McGraw-Hill Singapore, 1997.

[12] Ramez Elmasri and Shamkant B Navathe. *Fundamentals of database systems*. Pearson, 2014.

[13] Mike Stonebraker, Daniel J Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, et al. C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.

[14] Eelco Plugge, Tim Hawkins, and Peter Membrey. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*. Apress, Berkely, CA, USA, 1st edition, 2010.

[15] Michael Stonebraker. Sql databases v. nosql databases. *Communications of the ACM*, 53(4):10–11, 2010.

[16] Kristina Chodorow. *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[17] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[18] TPC. The TPC Benchmark H. `http://www.tpc.org/tpch/`, 2016.

[19] TPC. TPC - Current Specifications. `http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp/`, 2016.

[20] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[21] The PostgreSQL Global Development Group. PostgreSQL: The world's most advanced open source database. `https://www.postgresql.org/`, November 2016.

[22] MonetDB B.V. The column-store pioneer — MonetDB. `https://www.monetdb.org/`, 2016.

[23] MongoDB Inc. MongoDB for GIANT Ideas — MongoDB. `http://www.mongodb.org`, 2016.

[24] Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon, and Lavanya Ramakrishnan. Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, Science Cloud '13, pages 13–20, New York, NY, USA, 2013. ACM.