DE GRUYTER OPEN

Scientific Annals of Economics and Business
63 (SI), 2016, 21-50
DOI: 10.1515/saeb-2016-0134

PERFORMANCE ANALYSIS OF TWO BIG DATA TECHNOLOGIES ON A CLOUD DISTRIBUTED ARCHITECTURE. RESULTS FOR NON-AGGREGATE QUERIES ON MEDIUM-SIZED DATA

Marin FOTACHE[*], Ionuț HRUBARU[**]

Abstract

Big Data systems manage and process huge volumes of data constantly generated by various technologies in a myriad of formats. Big Data advocates (and preachers) have claimed that, relative to classical, relational/SQL Data Base Management Systems, Big Data technologies such as NoSQL, Hadoop and in-memory data stores perform better. This paper compares data processing performance of two systems belonging to SQL (PostgreSQL/Postgres XL) and Big Data (Hadoop/Hive) camps on a distributed five-node cluster deployed in cloud. Unlike benchmarks in use (YCSB, TPC), a series of R modules were devised for generating random non-aggregate queries on different subschema (with increasing data size) of TPC-H database. Overall performance of the two systems was compared. Subsequently a number of models were developed for relating performance on the system and also on various query parameters such as the number of attributes in SELECT and WHERE clause, number of joins, number of processing rows etc.

Keywords: Big Data, cloud computing, performance benchmarks, Hadoop, Hive, PostgreSQL, Postgres XL, R

JEL classification: M15

1. INTRODUCTION

Big Data refers to the storage, processing and analysis of big volumes of data, generated with big velocity in a big variety of sources and formats (Buhl *et al.*, 2013; Kowalczyk and Buxmann, 2014; Stonebraker, 2012b; Ylijoki and Porras, 2016). As the volume, speed and heterogeneity of data have accelerated with the web and mobile technologies, centralizing all the data in a monolith database and/or data warehouse system became less and less feasible. In terms of data storage and processing, essential for Big Data solutions is the distributed computing with systems deployed in-house or in cloud (Jacobs, 2009).

[*] Faculty of Economics and Business Administration, Alexandru Ioan Cuza University of Iași, Romania; e-mail: *fotache@uaic.ro* (corresponding author).
[**] Faculty of Economics and Business Administration, Alexandru Ioan Cuza University of Iași, Romania; e-mail: *ionut.hrubaru@gmail.com*.

Traditional relational/SQL databases and data warehouses could be part of any viable Big Data system. But in terms of data storage and processing Big Data has brought mainly three group of technologies: NoSQL data stores (Cattell, 2010; Cogean *et al.*, 2013; Fotache *et al.*, 2014; Lungu and Tudorica, 2013), Hadoop ecosystems (Fotache and Hrubaru, 2016; Hrubaru and Fotache, 2015; Li *et al.*, 2014; Stonebraker, 2012a), and NewSQL/in-memory systems (Pavlo and Aslett, 2016; Stonebraker, 2012a; Trancoso, 2015). A recurrent theme in Big Data literature is the improved performance of data processing in new Big Data technologies compared to the classical relational/SQL Data Base Management Systems (RDMS's).

This paper inquires the performance related to data processing operations of two systems, one representing the relational/SQL camp - PostgreSQL/Postgres XL and the other the Big Data camp - Hadoop/Hive. Both are free, open-source technologies, largely popular and affordable for almost any type of organizations. Tests were devised on a distributed five-node cluster deployed in cloud (Amazon).

Unlike some acknowledged benchmarks such as YCSB (Cooper *et al.*, 2010) and TPC (Transaction Processing Performance Council - TPC, 2014), the focus here was on executing (and collecting the results, i.e. the duration of query completion) *randomly* generated non-aggregate queries on TPC database subschema loaded with different volumes of data and. Results were related to various parameters of executed queries. In this paper the generated queries (by modules written in R language) were non-aggregate, i.e. lacking GROUP BY and HAVING clauses.

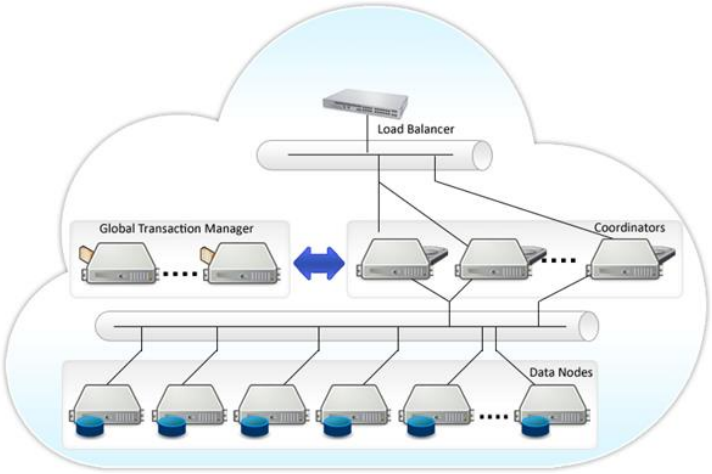
Overall performance of the two systems was compared. Subsequently a number of models were developed for relating performance on the data store and also on various query parameters such as the number of attributes in SELECT and WHERE clause, number of joins, number of processing rows etc.

2. POSTGRES AND HADOOP/HIVE AS BIG DATA TECHNOLOGIES

Relational/SQL DBMS's and Hadoop ecosystem are popular choices for data persistence systems today. Relational databases were proposed in 1970s and become operational in 1980s, whereas Hadoop technologies developed at a spectacular pace after 2003. As all other RDBMS's, PostgreSQL was based mainly on a centralized architecture. In recent years, as a response to Big Data pressure, PostgreSQL has been gradually incorporating principles of distributed computing but it is not a fully distributed DBMS yet. As consequence the tests were performed on Postgres-XL which is a forked version of PostgreSQL designed to be distributed and has some other features like MPP (PostgresXL, 2016).

2.1 PostgreSQL/Postgres XL

Postgres XL is a shared nothing distributed database management system (Figure no. 1) acting as a cluster that can split OLTP and OLAP workloads among its nodes and can scale horizontally (scale-out). It is based on Postgres and fully supports SQL.



Source: PostgresXL (2016)

Figure no. 1 – Postgres XL architecture

Postgres XL is based on three major components:

- Global Transaction Manager (GTM) node handles distributed transactions and provides ACID capabilities. GTM is vital for Postgres XL architecture so it is usually deployed on a separate and (more) powerful machine. For dealing with the risk of a single point of failure for the cluster, a standby GTM node can be configured for transparently taking over when master GTM fails.
- Coordinator node handles client application requests, determines which data nodes need to be queried and builds and sends execution plans to the data nodes. It also gathers the results and sends them to the client applications. Coordinators hold only metadata (in catalog views). If one coordinator fails, then the application will transparently connect to another.
- Data node holds the data and does the processing. It is recommended to have as many coordinators as the data nodes in a cluster in order to manage the workload balance and to reduce the network calls.

A functional cluster must have all the three types of nodes. Optionally GTM-Proxy nodes can be installed for grouping multiple messages coming from Coordinator and Data nodes on the same server. Each node is in fact a Postgres instance, and resource allocations such as port numbers need configuration. Postgres XL is fully ACID compliant and it is an open source project targeted to merge back its functionalities to Postgres source code. Since it is based on Postgres, it supports all its functionality.

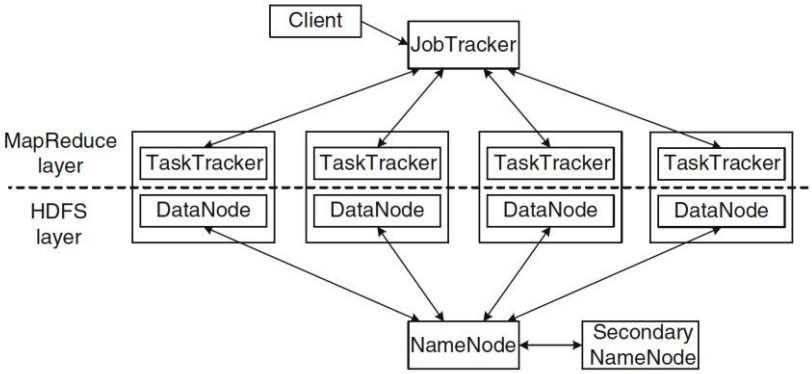
Data is either distributed through a shard key or replicated at each data node. Data which is small in volume and relatively static can be replicated, while big data should be *sharded*. Data collocation is important for sharded tables, since joins should operate on a single node ideally, and not send data through network from one node to another because this usually hampers performance. When a query is issued to a Coordinator, it identifies the nodes holding the data, produces an execution plan and sends it to the data nodes. Each data node processes the query in parallel and sends back the results to the coordinator. Scalability can be achieved by adding nodes in the cluster when workloads require it, and the system will transparently self-rebalance in terms of data. High availability is achieved by adding

slave nodes for each data nodes that take over the request when a data node fails. Postgres XL works on both Windows and Linux systems and maps very well to elastic clouds such as Amazon AWS.

2.2 Hadoop/Map Reduce/Hive

Hadoop is essentially a framework for storing data in large clusters of commodity data. Fitted for storing and processing big volumes of heterogeneous data Hadoop is not appropriate for dealing with big velocity requirement since it was designed to process static data.

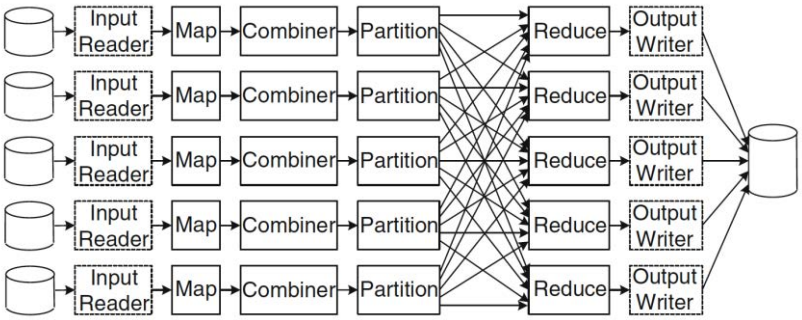
Hadoop architecture (Figure no. 2) has two main components: the storage which is a distributed file system (HDFS) and the processing component which is most cases relies on a MapReduce implementation. MapReduce is a framework/algorithm designed specifically for parallel processing which maps very well to the HDFS philosophy of storing data across multiple data nodes in a distributed environment. Recent Hadoop releases provides additional data processing using a new resource engine (YARN) and a new data processing framework, Apache Tez (Lublinsky *et al.*, 2013, pp. 435-455).



Source: Doulkeridis and Norvag (2014)
Figure no. 2 – HDFS architecture

The main characteristics of HDFS are sequential writing, lack of support for random reads and the fact it cannot update the content of a file (White, 2015, pp. 73-99). In HDFS a file is split between multiple blocks and distributed across multiple commodity machines in a Hadoop cluster. These nodes are called data nodes, and the information about blocks (name, size, path, permissions) – the metadata, are stored on a name node. Data nodes keep data in blocks. Each data node runs a background process which keeps tracks of the blocks and talks to the name node about status and health. The metadata on the name node keeps track of the block stored on each slave node.

A Map Reduce job is a piece of code (written in Java, Python etc.) that runs on each data node in a cluster containing two main functions/interfaces: (1) Map which processes the input in parallel, produces key value pairs which become input to a (2) Reduce function which produces the final result – see Figure no. 3.



Source: Doulkeridis and Norvag (2014)
Figure no. 3 – Map-Reduce dataflow

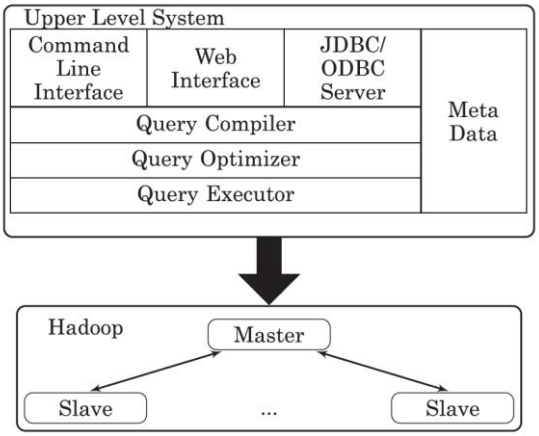
The first step in the flow is a logical input split based on the number of data nodes in the cluster and the number of data blocks. After setting the number of mappers, the processing starts in memory (for performance reasons) until the buffer is filled, and from this point on, the buffer will be spilled to the disk. The result of each mapper is then shuffled –this is when the data from each task tracker is sent through the network, and this can be a performance penalty, depending on the network latency. The result of the shuffle is the actual input to the reducer which applies the code logic to produce the final result and persist it in HDFS.

As Li *et al.* (2014) pointed out, MapReduce adopted a loosely coupled design, where the processing engine is independent of the underlying storage system, allowing the system to scale its processing and storage both up and down as needed. Another advantage is the fact that the distribute execution of the program, the resource allocation, block identification and monitoring of the execution is done by the framework itself. Hence there are some common patterns in which Map Reduce works best: summarization, filtering, grouping and joining.

The main problem with Map-Reduce is a consequence of its tightness to the input data. Programmer must know in detail the data structure and has to design and implement all the logic of both Map and Reduce phase. Changes in the input data are not only about changes in data the structure (new keys, new attributes, etc.) but also changes in data volume scale which can affect the logic used by a client application to process the data. This kind of logic and decisions are automatically taken by an optimizer in the case of a RDBMS, but for Map Reduce, it is the programmer's responsibility to implement it. In contrast to relational/SQL DBMSs, Map-Reduce lost the logical data independence (Hrubaru and Fotache, 2015).

It is difficult for inexperienced users (non-programmers) to write Map-reduce jobs to solve even the simplest of queries, which were a trivial task in high-level languages such as SQL (Stonebraker, 2015). Consequently, a large number of Big Data projects focused on building frameworks that implement high-level query languages (SQL-like) hiding to the users all the details of Map-Reduce tasks. Hive has been created by the Facebook Data Infrastructure Team as an open-source data warehousing solution built on top of the Hadoop environment (Sakr *et al.*, 2013; Thusoo *et al.*, 2009).

Hive projects structure into the data stored in HDFS by storing the metadata in system catalog file called metastore – see Figure no. 4. The query lifecycle in Hive is similar to that in a relational database server (Hrubaru and Fotache, 2015) - parsing (syntactic and semantic), execution plan generation (using an optimizer which can be a rule based one or in the latest versions a cost based one), generating the steps (Map Reduce jobs) and executing them.



Source: Li et al. (2014)

Figure no. 4 – Hive basic architecture

Hive works with well-known concepts from relational databases such as tables, columns, rows and data types. Data types, there are primitive (string, integer, timestamp, date, etc.) and collections data types (struct, array, and map) but custom types can be created, all implemented in Java and inheriting the underlying behaviour. Besides tables, from a logical perspective there are partitions also. Tables in Hive can be partitioned, just like in a classical RDBMS, using different columns as criteria to horizontally split the data.

Hive Query Language (HiveQL) is a SQL like declarative language which is used for both DDL and DML operations in Hive (Hrubaru and Fotache, 2015; Sakr *et al.*, 2013). Since Hive is in fact another layer over Map Reduce, all the HiveQL commands will be translated in Map Reduce jobs executed against HDFS. This has the advantage of hiding the Map Reduce implementation from the end user and tells hive what the user wants not how to process it.

Apparently Hive provides what Map-Reduce could not, i.e. data logical independence. But logical data independence in Hive is incomplete. HiveQL provides features for directly interacting with the physical layer. There are clauses for serialization and de-serialization, file formats etc. Hive operates based on *schema on read* principle. The logical data model is available only for projecting structure over a physical file and data block.

3. TPC-H BENCHMARK

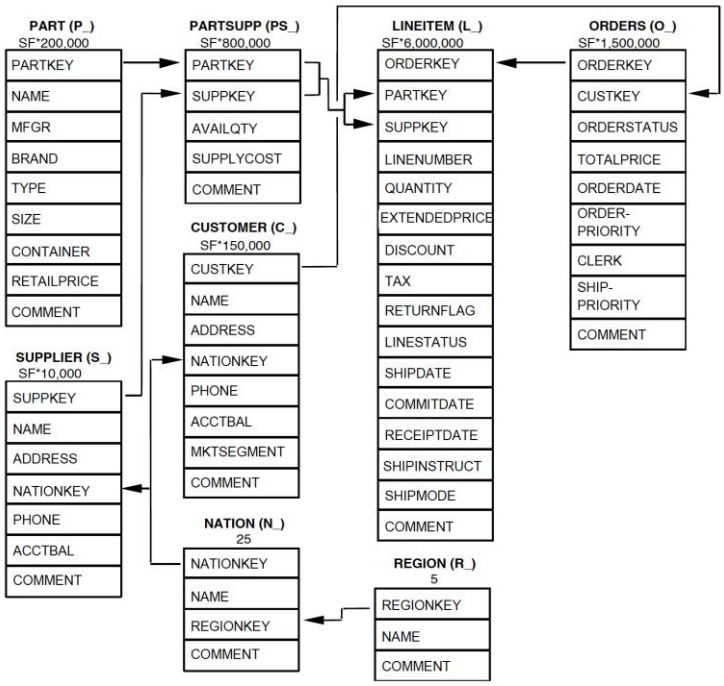
A large number of benchmarks have been proposed for assessing data processing performance for data stored/servers deployed in centralized or distributes architectures implemented in house or in cloud. Currently *Yahoo! Cloud Serving Benchmark* (YCSB) and *Transaction Processing Performance Council* (TPC) are the most prominent. As the interest was in relating query execution duration to various query parameters, the latter was preferred.

TPC is a non-profit corporation established for defining transaction processing and database benchmarks and to disseminate objective, verifiable benchmarks for data performance in one of the following four categories:

- OLTP: TPC-C and TPC-E;
- Decision support: TPC-H, TPC-DS and TPC-DI;

- Virtualization: TPC-WMS;
- Big Data: TPCx-HS.

Considering the batch processing nature of Hadoop and the objective of comparing it with other systems, specifically a classical RDBMS, TPC-H was chosen as the benchmark for this paper. TPC Benchmark H (Transaction Processing Performance Council - TPC, 2014) is a decision support benchmark that comes with both a logical database design (a schema structured in a set of predefined tables) – see Figure no. 5 – and a set of predefined queries (templates) that are executed against the schema.



Source: Transaction Processing Performance Council - TPC (2014)

Figure no. 5 – TPC-H database schema

The standard implementation provides two tools, *DBGen* for generating the schema and populate the database, and *QGen* that generates queries based on values for specific input parameters in the templates (Kejser, 2014). The volume of data generated is set through a scale factor parameter (SF) for which the minimum is the estimated business data of 10000 suppliers (almost 10 million rows) with approximately 1 GB of storage data.

There are two performance metrics reported by TPC-H:

- Query – per – Hour Performance metric (QphH@Size) which takes into account the database size (Scale Factor), the query processing power for a single stream and query throughput for multiple concurrent users;
- Price/Performance metric which takes into account Price/QphH@Size

The components of the TPC-H database consist of eight individual tables (the Base Tables) as seen in Figure no. 5. Inside the parenthesis there is the prefix of each table. The

record count is either fixed (tables NATION and REGION) or dynamic based on the scale factor (SF). The arrows point to one-to-many relationship between parent and child table (primary and foreign keys).

TPC-H proposed twenty-two decision support queries and two database refresh functions which must be executed as part of the TPC-H benchmark. Queries generated by QGen module are guaranteed to be compliant with the standard. However, adjustments and changes can be made to the queries from a syntactical point of view to guarantee executions on specific DBMS. There are two performance metrics suggested by the TPC-H benchmark: (1) Query power which measures the time in seconds taken to execute all 22 queries by one user/session only (one query stream) and (2) Query throughput which accounts for a variable number of users.

4. METHOD AND TOOLS

The experimental design proposed in this paper questioned the generally acknowledged idea that, in distributed architectures, Big Data/Hadoop technologies perform better than traditional SQL/relational technologies in data processing (database query). Gradually increasing the database size, a couple of models were built and tested for predicting system performance based on various query parameters and the database loading.

4.1 Design

A module was devised (in R language) for generating random non-aggregate queries to be executed on various subschema of TPC-H database. Each subschema was created and populated for a specific scale factor (data loading) – see Table no. 1 – and have distinct, randomly generated records.

Table no. 1 – Number of records for each table and scale factor

Table in TPC-H schema	SF = 0.5	SF = 1	SF = 2	SF = 5
REGION	5	5	5	5
NATION	25	25	25	25
CUSTOMER	75,000	150,000	300,000	750,000
SUPPLIER	5,000	10,000	20,000	50,000
PART	100,000	200,000	400,000	1,000,000
PARTSUPP	400,000	800,000	1,600,000	4,000,000
ORDERS	750,000	1,500,000	3,000,000	7,500,000
LINEITEM	2,999,671	6,001,215	11,997,996	29,999,795

Each generated query had a random numbers of attributes in SELECT and WHERE query clauses. Then attributes in both clauses were also selected randomly. Based on the attributes, FROM clause was automatically built so that the query can extract and filter the attributes. Filter predicate (WHERE clause) was also randomly generated using connectors such as AND and OR, and various operators (BETWEEN, IN, >, >=, etc.). Queries were meant to be operational, so every query was targeted for a specific scale factor subschema (100 queries were created and launched for each scale factor subschema).

4.2 Variables

The variable of most interest was the duration of the query (the interval required for each system for completing a query) codified as *duration*. Average number of processed rows within a query was preferred to the scale factor variable and coded *avg_n_of_rows*. Nominal variable *dbserver* qualify the system (Hive or Postgres) where each query was executed and query duration was recorded. Other variables refer to various query parameters:

- *n_of_attribs_select* – number of attributes in SELECT clause;
- *n_of_attribs_where* – number of attributes in WHERE clause;
- *length_attrib_string__select* – sum of string (CHAR and VARCHAR) attributes length in SELECT clause;
- *length_attrib_string__where* – sum of string attributes length in WHERE clause;
- *n_of_and__where* – number of AND connectors in WHERE clause;
- *n_of_between__where* – number of BETWEEN operators in WHERE clause;
- *n_of_oper_eq_where* – number of equality (=) operators in WHERE clause;
- *n_of_oper_other_than_eq_where* – number of comparison operators, other that the equality operator (>, >=, <, <=, <>) in WHERE clause;
- *n_of_in__where* – number of IN operators in WHERE clause;
- *n_of_in_values__where* – total number of values included as arguments for IN operators in WHERE clause;
- *n_of_joins* – number of table joins;
- *n_of_pku_int__select* – number of integer and primary key attributes in SELECT;
- *n_of_pku_int__where* – number of integer and primary key attributes in WHERE.

Some of the parameters, such as *n_of_and__where*, *n_of_between__where*, *n_of_in__where*, *n_of_joins*, are basic, "raw" operators whereas parameters like *n_of_attribs_select*, *length_attrib_string__where*, *n_of_oper_other_than_eq_where*, are pre-aggregated (e.g *n_of_attribs_select* is the sum of integer, number/real, char, varchar and date attributes included in SELECT clause). Additional details about variables are provided in section 5.1.

4.3 Platforms

The system under testing was a five-node cluster running Oracle Linux Server version 7.2 and deployed into AWS cloud using c4.2xlarge instances which have 8 vcpu, 15GB of RAM and 1000 mbps dedicated EBS bandwidth. Expected throughput is 128 MB/s and 8000 IOPS 16 KB size. Disks were standard EBS volumes each node having 70GB which at some point was increased by 100GB because Hive map reduce jobs were exhausting all the space when running with 5 GB scale factor. Special private IPs, hostnames, firewall rules and opened ports were applied to solve connectivity issues so that nodes could communicate between themselves.

All nodes in the cluster were configured using public key authentication for password less SSH login. Putty was used to SSH on the machines. On each node Hadoop was installed using Cloudera Distribution CDH 5.7.1. This distribution contains Hadoop 2.6 and Hive 1.1.0. Java 1.7 was used to run Hadoop. One node was configured as the name node while the remaining four were configured as data nodes. For the Hadoop cluster setup, administration and monitoring, Cloudera Manager was used. Apache Hue was used for Hive

to create databases and load data into the TPCH tables together with the hive command line interface (CLI).

Data has been loaded from the Linux file system into HDFS using LOAD command which made data accessible for Hive. Tables have been created in HIVE with CREATE TABLE command and the data was automatically distributed on each node's HDFS.

Jobs have been monitored for resource utilization: memory, disk space and CPU.

After tests were run on Hive, the Hadoop cluster was stopped and uninstalled and the data has been cleared. Then Postgres XL version 9.5 was installed in the cluster using pgxc_ctl utility. Using the tarball from their official site did not work and the problem was solved by cloning the latest Git repository from 9.5 STABLE branch, manual building the installer with make and then deploying it in the cluster using one GTM node, 4 coordinators and 4 data nodes with pgxc_ctl utility. WAL (write ahead log) had to be moved to a mount disk with more free space (/var). Pgxc_ctl utility was used to administer (start/stop) and monitor the cluster.

Tables can be created either as distributed tables on each node or replicated ones. It makes sense to distribute large tables and to replicate small ones. One important thing in a distributed system is data locality – when joins are involved they should happen locally by collocating data with same keys.

In Postgres XL tables LINEITEM and ORDERS were distributed by HASH using ORDERKEY and tables PART and PARTSUPP were distributed by HASH using PARTKEY as the column. Also CREATE TABLE AS command used keyword ROUNDROBIN for distributing the created table on all the nodes, otherwise the table would have been replicated and performance would suffer especially from large result sets.

In Hive, the distribution of the tables is done automatically by HDFS and not accessible to the user. The Hive clause CLUSTERED BY/DISTRIBUTE BY distributes the table in buckets inside each data node, but the data blocks location is controlled by HDFS, and not Hive. HDFS evenly distributes data across data nodes.

Results were gathered with JMeter. Each query uses a CREATE TABLE AS statement to eliminate the network traffic and possible JMeter crashes because of memory exhaustion due to the result set size. After each query the table would be dropped to clear the disk space.

Data was generated on each scale factor using DBGen (Kejser, 2014). The query generator was devised in R. The data analysis platform was open-source, R/RStudio with various packages (libraries) presented in the subsequent sections. Other R modules for data processing and analysis were created as well. All of the figures in sections 5.1 and 5.2 were created with *ggplot2* package (Wickham, 2016).

4.4 Methods

First an Exploratory Data Analysis was conducted for describing the distribution of value for all the variables and also for comparing the overall performance of the two data stores. A number of questions were considered in performance comparison, as they could generate testable hypotheses.

- Is overall Hive/Hadoop performance better than PostgreSQL?
- Which factor loading (database size) seems more appropriate for relational/SQL technologies (Postgres XL) and which are more appropriate for Big Data (Hadoop/Hive) technologies?

- Which is the database size threshold when Big Data technology have to replace SQL data stores?

Exploratory data analysis for overall and each scale factor results were accompanied by a set of non-parametric tests for assessing the statistical significance of the results. As shown in next section (5), non-parameter tests were necessary, as normality and variance homoscedasticity requirements were not fulfilled. In order to confirm the statistical significance of differences previously identified using graphics and descriptive statistics, the following hypotheses were tested:

H1: Overall non-aggregate query performance is better for Postgres than Hive

H2: Non-aggregate query performance is better for Postgres than Hive for scale factor 0.5

H3: Non-aggregate query performance is better for Postgres than Hive for scale factor 1

H4: Non-aggregate query performance is better for Postgres than Hive for scale factor 2

H5: Non-aggregate query performance is better for Postgres than Hive for scale factor 5.

Performance drivers were analysed building regression models. For each query the result (duration) of just one of the tested systems was used in the regression models (randomized/blocked design). Regression models were expected to provide reliable answers to questions such as:

- are the performance differences between Postgres XL and Hive constant or varies on different levels of some of the predictors?
- which are the most important predictors in explaining variation of the outcome (duration)?
- is predictors interaction significant?

As the requirements for OLS (Ordinary Least Square) regression were not met, the OLS models were subsequently tested using techniques such as Generalized Least Squares, Robust Regression (Huber M-Estimation) and Non-parametric, rank-based regression.

5. RESULTS

This section is organized as follows. First, the main parameters of generated non-aggregate queries are presented. Second, results (duration of query execution) are compared for the tested systems, overall and on each scale factor. Third, performance drivers are identified and assessed through a series of regression models.

5.1 Main parameters of generated non-aggregate queries

In section 4.2 main variables of interest were explained. Distribution for variable duration will be described in the next section. This section will detail the values of the main query parameters and also variables associated to the database size.

On average, the SELECT clause of the query contained about 19 attributes and the WHERE clause contained 14 attributes (see Figure no. 6). Median number of attributes was 19 for SELECT and 13 for WHERE.

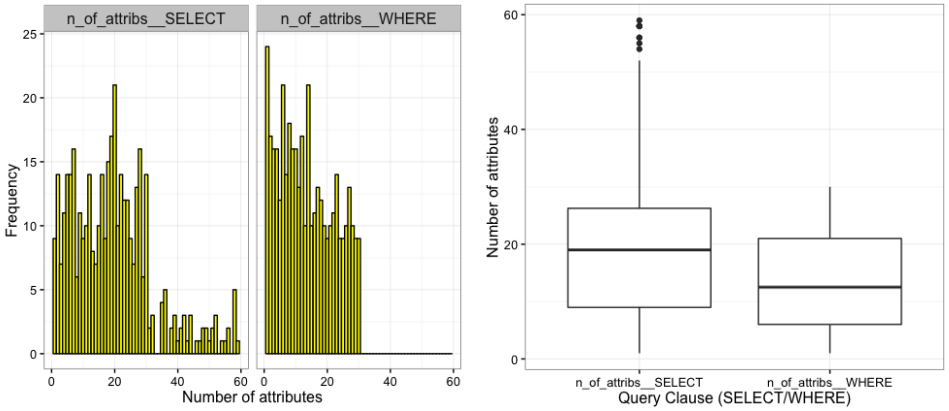


Figure no. 6 – Distribution of the values for number of attributes in SELECT and WHERE clauses

Charts in Figure no. 7 compare the frequency of attributes in the SELECT clause by data type. As query attributes were randomly included in the queries, TPC-H database structure was pivotal in the results. The most frequent attributes in the SELECT clause are of type INTEGER (with both mean and median around 6), as the primary keys (surrogate keys) for most tables are of this type. Then come CHAR (mean=5.38, median=5) and VARCHAR (mean=4.57, median=4) followed by the NUMERIC - decimal or real (mean=3.5, median=3). As expected the least frequent is the DATE data type (mean=1.84, median=2). All distributions are skewed to the right.

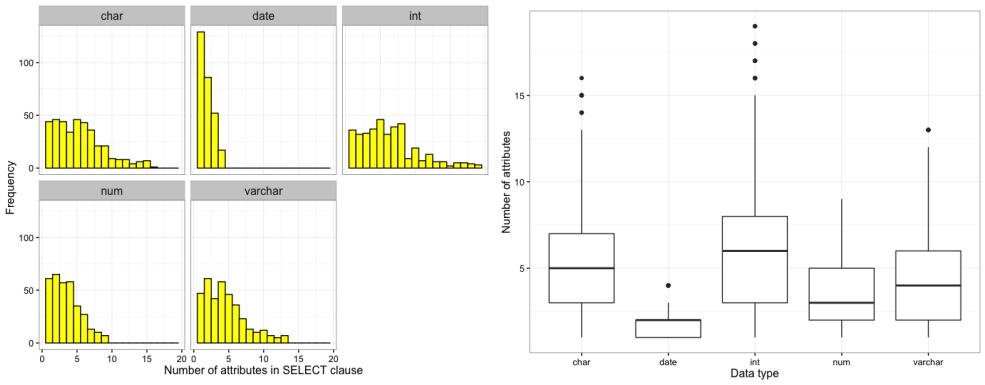


Figure no. 7 – Frequency the attributes data type in the SELECT clause

Frequency of data type attributes in the WHERE clause is displayed in Figure no. 8. As in the case to the SELECT clause, the most frequent attributes in the SELECT clause were of type INTEGER (mean=4.71, median=4), followed by CHAR (mean and median of 4) and VARCHAR (mean=4.57, median=4).

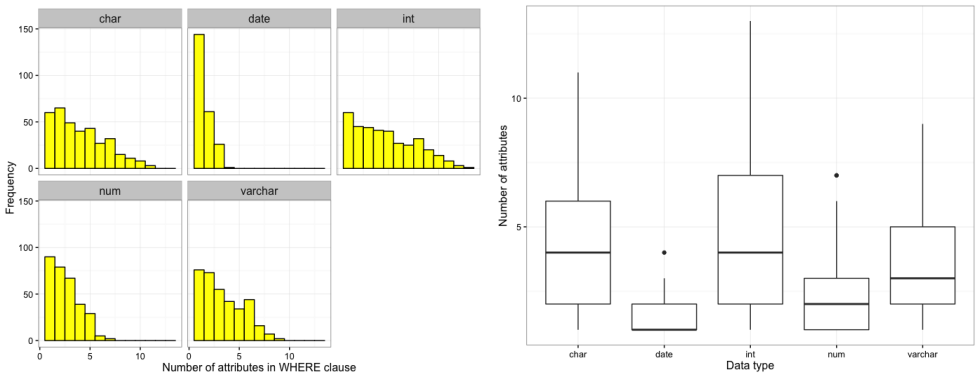


Figure no. 8 – Frequency the attributes data type in the WHERE clause

The left side of the Figure no. 9 depicts the distribution of string attributes length in the SELECT clause, by data type (CHAR vs VARCHAR). The scales are different for the two panels, so the aggregated length of CHAR attributes in SELECT clause varied between 1 and 253 whereas for VARCHAR attributes it varied between 23 and 1052. Distribution of CHAR attributes length is skewed to the right (mean=85, median=77) and for the VARCHAR attributes is bimodal (mean = 366, median= 308), with peaks around values of 175 and 1050. Figures and shapes were not radically different for the WHERE clause.

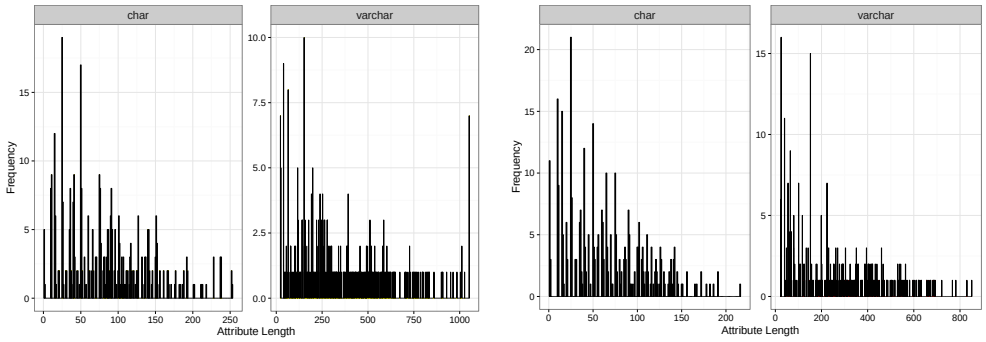


Figure no. 9 – Frequency of attributes length for string attributes in SELECT (*left*) and WHERE (*right*) clauses

It is generally expected that the duration of query execution to be less influenced by the frequency and length of various attributes in the SELECT clause, but predominantly by the structure of WHERE clause (apart from the type and length of attributes presented above).

Figure no. 10 shows the number of the comparison operators in the WHERE clause by type. The most frequent operator is BETWEEN followed by IN. The figure depicts also the number of values in the IN clause showing that there are queries with more than 250 values in the list.

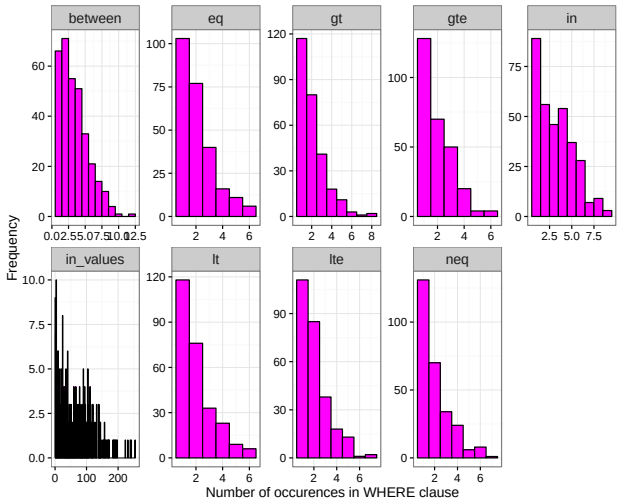


Figure no. 10 – Frequency of operators for comparison

Figure no. 11 on the left shows the distribution of the number of tables and joins in the queries. The two parameters are perfectly correlated so one of them must be removed for avoiding collinearity in further regression models. Perfect correlation indicates that generated queries did not contain self joins (when simple, non-aggregate and not subqueried queries contain self-joins, the number of joins could exceed the number of tables). By far the most frequent number of tables included in queries was 8. Even if the query attributes were included randomly, apparently it was necessary to include most or all of the tables in order to make the queries operational (by building a proper JOIN chain).

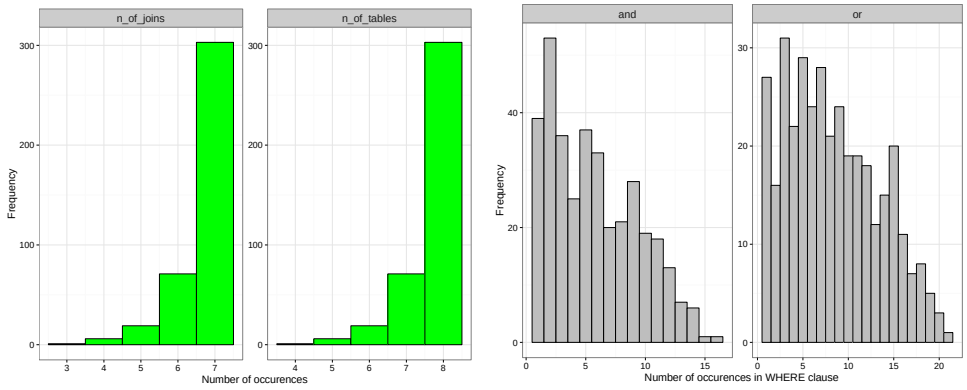


Figure no. 11 – Frequency of the number of tables and the number of joins (*left*) and the frequency of connectors AND and OR in WHERE clause (*right*)

Another important factor that can influence query performance could be the number of AND and OR operators in the WHERE clause. The right side of Figure no. 11 depicts their distribution. Skewness of their distributions is related to the number of attributes (randomly) included the clause WHERE.

Naturally one of the most important divers of query execution duration was the size of the processed data. Instead of the scale factor variable interest was initially on four other parameters related to the data size: the minimum number of processed rows in a query (*min_n_of_rows*), the maximum number of processed rows in a query (*max_n_of_rows*), the average number of processed rows in a query (*avg_n_of_rows*) and the median number of processed rows in a query (*median_n_of_rows*).

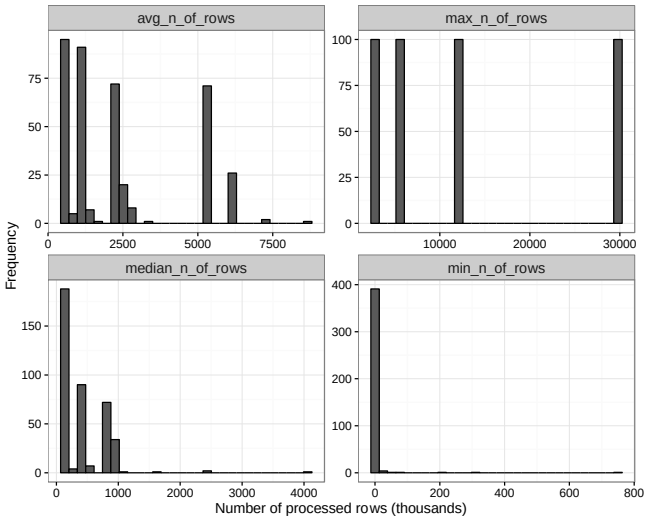


Figure no. 12 – Summaries about the number of processed rows in a query

Figure no. 12 shows that the distribution of all four parameters is not continuous (a consequence of including in the data set the results of only four scale factors). Variable *min_n_of_rows* id highly concentrated on value 1. Also variable *max_n_of_rows* has a small number of values. Finally of four variables only *avg_n_of_rows* was kept in the analysis, as it seems to present the larger variability.

5.2 Query duration on two data servers for the five-node cluster

This section investigates one of the main topic of interest for the thesis, i.e. big data technologies performance compared to traditional SQL/relational counterpart when processing medium-sized data.

Particularly, the main questions were:

- Is overall Big Data (Hive/Hadoop) system performance better than the relational/ SQL (PostgreSQL) performance?
- Which factor loading (database size) seems more appropriate for relational/SQL technology and which are more appropriate for Big Data technology?
- Which is the database size when Hadoop/Hive will clearly be a better option than Postgres XL?

Exploratory data analysis for overall and each scale factor results were accompanied by a set of non-parametric tests for assessing the statistical significance of the results. As

will be seen, non-parameter tests were necessary, as normality and variance homoscedasticity requirements were not fulfilled (so t-tests were not appropriate).

Figure no. 13 presents an overview of overall (all scale factors) distribution of query duration in Hive and PostgreSQL. Both the superimposed density curves (left side of the figure) and the boxplot of duration for the tested systems (right) provide a proper glimpse of overall performance.

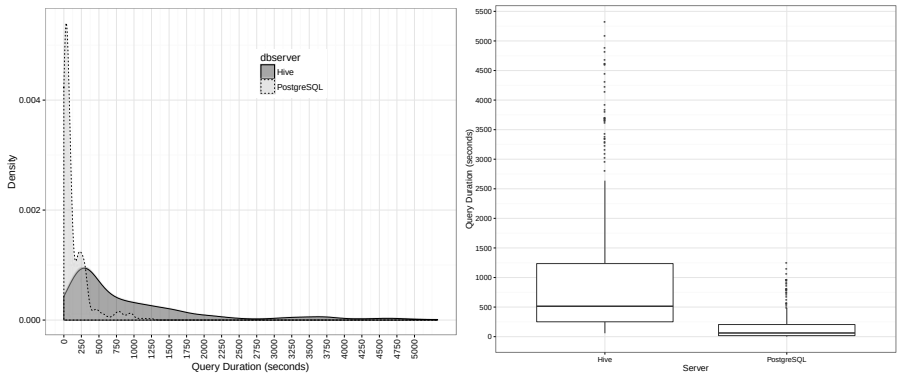


Figure no. 13 – Overlapping density curves (*left*) and boxplot (*right*) for the distribution of duration (all scale factors)

Next investigated question was if there are significant differences in performance between Hive and PostgreSQL among different scale factors (loadings) of the database – see Figure no. 14. Specifically, it was expected that, with the increase of database size, Hive will catch-up and eventually outperform PostgreSQL.

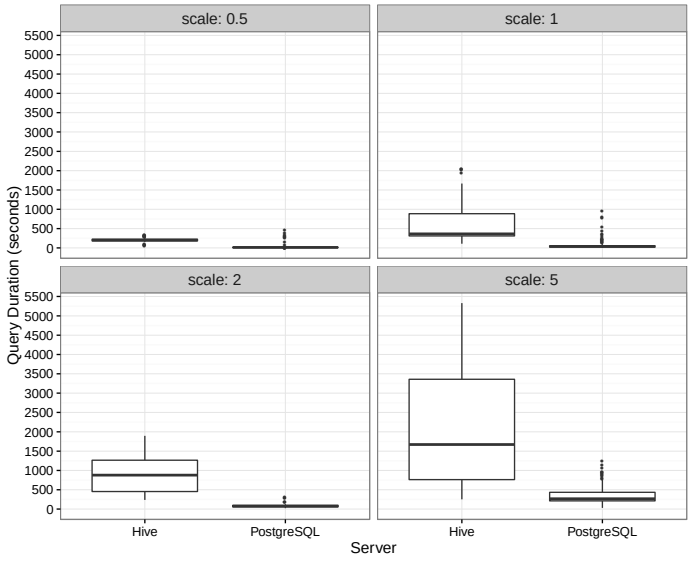


Figure no. 14 – Comparison of duration median, quartiles and outliers, by each scale factor

Figure no. 14 shows that, contrary to what was expected, with the increase of the database size, the performance gap between PostgreSQL and Hadoop/Hive did not shrink al all, but on the contrary. Table no. 2 provides the values of medians, differences of medians and ratio of median for the duration of queries in each of the two tested systems, overall and for each scale factor.

PostgreSQL performed better than Hive overall and for each scale factor. The highest gap was recorded for scale factor of 2. With the increase of database size from scale factor 2 to 5 the gap ration narrows spectacularly, and one can speculate about the gap in the case of higher scale factors (10, 50, and 100).

Table no. 2 – Basic figures about PostgreSQL-Hive performance gap

Scale factor	Median hive	Median pg	Diff median hive-pg	Ratio median hive/pg
0.5	203.891	14.338	189.553	14.22
1	363.749	41.798	321.951	8.703
2	879.909	77.059	802.85	11.419
5	1671.336	267.746	1403.59	6.242
Overall	***513.832***	***61.657***	***452.175***	***8.334***

In order to confirm the statistical significance of differences previously identified using graphics and descriptive statistics, the following hypothesis will be tested:

H1: Overall non-aggregate query performance is better for PostgreSQL than Hive

H2: Non-aggregate query performance is better for PostgreSQL than Hive for scale factor 0.5

H3: Non-aggregate query performance is better for PostgreSQL than Hive for scale factor 1

H4: Non-aggregate query performance is better for PostgreSQL than Hive for scale factor 2

H5: Non-aggregate query performance is better for PostgreSQL than Hive for scale factor 0.5

Classical parametric test for comparing performances of two data servers would be the t-test for paired data. This test requires that data to be normally distributed and to have equal variance. In R there is a plethora of functions for normality, such as *shapiro.test* (base R) for Shapiro-Wilk normality test, *ks.gof* in package *pgirmess* (Giraudoux, 2016) for Kolmogorof-Smirnov goodness of fit test to normal distribution and *ad.test* in package *nortest* (Gross and Ligges, 2015) for Anderson-Darling test for normality. In each case the null hypothesis was H0: *the distribution of duration is normal* and the alternative hypothesis was Ha: *the duration does NOT follow a normal distribution*. Non-normality was reported by all the functions (W = 0.624144359, p-value = 1.76E-38).

As the normality assumption was not met for the technical score, a non-parametric "paired" Wilcoxon Signed-Rank test (Kloke and McKean, 2015, pp. 16-22) was performed using function *wilcox.test* implemented in base R and *wilcox.exact* implemented in package *exactRankTests* (Hothorn and Hornik, 2015), overall and for each scale factor.

The tested null and alternative hypotheses for the Wilcoxon Signed-Rank test were stated as follows (see column alternative which shows the current setting of the alternative hypothesis):

- H0: The median difference between paired samples is 0 (median of query duration in Hive – median of query duration in PostgreSQL); alternative hypothesis Ha: The median difference between paired samples is different from 0 – this is the case when alternative is set on "two.sided" value.
- H0: The median difference between paired samples is equal or less than zero; alternative hypothesis Ha: The median difference between paired samples is larger than 0 – this is the case when alternative is set on "greater" value.
- H0: The median difference between paired samples is equal or greater than zero; alternative hypothesis Ha: The median difference between paired samples less than 0 – this is the case when alternative is set on "less" value.

The groups correspond to query execution duration in Hive and PostgreSQL. The medians of the two groups were 513.832 (Hive) and 61.657 (PostgreSQL), respectively.

The paired Wilcoxon Signed-rank test shows in Table no. 3, that there is a significant effect of group (V = 79283, p-value < 2.493E-64). Reported difference between median of query duration in Hive and that of PostgreSQL was 624.76, 95% CI [530.92, 704.87].

Table no. 3 – Results of paired Wilcoxon Signed-rank tests

scale	function	V-statistic	p-value	alternative	method	95% CI	median (Hive-Pg) estimate
overall	wilcox.test	79283	2.49* E-64	two.sided	WSRTCC	[530.9, 704.9]	624.76
overall	wilcox.test	79283	1.25* E-64	greater	WSRTCC	[543.7, Inf]	624.76
overall	wilcox.test	79283	1	less	WSRTCC	[-Inf, 692.7]	624.76
overall	wilcox.exact	79283	0	greater	AWSRT	[543.7, Inf]	624.76
0.5	wilcox.test	4894	3.83* E-16	two.sided	WSRTCC	[175.4, 191.4]	183.76
0.5	wilcox.test	4894	1.92* E-16	greater	WSRTCC	[176.9, Inf]	183.75
0.5	wilcox.test	4894	1	less	WSRTCC	[-Inf, 190.1]	183.75
0.5	wilcox.exact	4894	2.22* E-16	greater	AWSRT	[176.9, Inf]	183.75
1	wilcox.test	4920	1.82* E-16	two.sided	WSRTCC	[383.2, 595.0]	466.48
1	wilcox.test	4920	9.12* E-17	greater	WSRTCC	[400.0, Inf]	466.48
1	wilcox.test	4920	1	less	WSRTCC	[-Inf, 577.7]	466.48
1	wilcox.exact	4920	1.11* E-16	greater	AWSRT	[400.0, Inf]	466.17
2	wilcox.test	5050	3.96* E-18	two.sided	WSRTCC	[730.3, 922.2]	823.89

scale	function	V-statistic	p-value	alternative	method	95% CI	median (Hive-Pg) estimate
2	wilcox.test	5050	1.98* E-18	greater	WSRTCC	[741.9, Inf]	823.89
2	wilcox.test	5050	1	less	WSRTCC	[-Inf, 909.2]	823.89
2	wilcox.exact	5050	0	greater	AWSRT	[741.9, Inf]	823.93
5	wilcox.test	5041	5.19* E-18	two.sided	WSRTCC	[1415.2, 1958.5]	1703.28
5	wilcox.test	5041	2.60* E-18	greater	WSRTCC	[1484.9, Inf]	1703.28
5	wilcox.test	5041	1	less	WSRTCC	[-Inf, 1919.8]	1703.28
5	wilcox.exact	5041	0	greater	AWSRT	[1484.9, Inf]	1703.32

WSRTCC - Wilcoxon signed rank test with continuity correction
AWSRT - Asymptotic Wilcoxon signed rank test

As the database size increased so was the case of the median of Hive – PostgreSQL difference.

5.3 Main drivers of non-aggregate query performance for the five-node cluster hostes in (Amazon) cloud

Previous section suggested that in the case of non-aggregate queries executed on a five-node distributed cloud architecture, Postgres performs better than Hadoop/Hive, overall and on each scale factor. This section investigates the relationship between the outcome (query duration) and various query parameters as predictors in a series of regression models.

An R module was devised for selecting the appropriate regression model incrementally. Starting linear regression models included the outcome (duration) and all the predictors described in section 4.2. As shown in section 5.2, the distribution of variable duration was skewed. Consequently, the regression model outcome was transformed without compromising the model clarity, from duration to the cube root of *duration* ($\sqrt[3]{duration}$).

Table no. 4 – The most non-significant attribute of each iteration model

Iter.	Removed predictor	Estimate	Std. Err.	t-value	p-value
1	n_of_attribs_select	-0.0036	0.0159	-0.2239	0.8229
2	n_of_pku_int__select	0.0224	0.0389	0.5759	0.5650
3	length_attrib_string__where	0.0004	0.0005	0.8040	0.4219
4	n_of_pku_int__where	-0.0873	0.0579	-1.5065	0.1327
5	n_of_in_values__where	0.0040	0.0024	1.6616	0.0974
6	n_of_in__where	-0.7971	0.4841	-1.6467	0.1004
7	n_of_oper_eq_where	-0.0570	0.0531	-1.0740	0.2835
8	n_of_oper_other_than_eq_where	-0.0445	0.0411	-1.0836	0.2792
9	n_of_between__where	-0.0384	0.0415	-0.9248	0.3556
10	n_of_and__where	-0.0554	0.0386	-1.4352	0.1520

Using a step-wise procedure, in each iteration the most non-significant attribute (the attribute with the largest/non-significant p-value) was removed – see Table no. 4. The process stopped when current linear model contained no attributes whose p-value was less than 0.05. This was the reference model, labelled as *model B*.

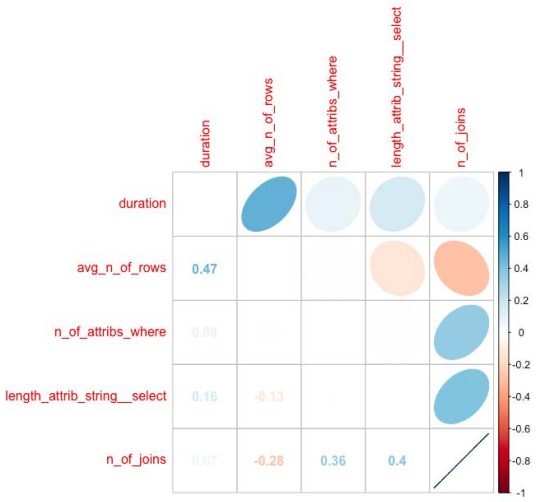


Figure no. 15 – Correlation plot for variables in model B

Next model variables correlations were checked for detecting collinearity. Spearman correlation coefficients were preferred as the distribution of predictor was not normal. Figure no. 15 presents the correlation plot produced by R function *corrplot* of package *corrplot* (Wei and Simko, 2016). Correlation coefficients did not exceed 0.64 which was adequate for model validity.

From model B, model A (which serves as a null model) was derived by removing the categorical factor (*dbserver*). Model C resulted by adding interactions in the model B (initially, all interactions were tested but gradually the least significant interactions were removed until the model contained only significant interactions). Table no. 5 contains key information about the competing models A, B and C: formula, R^2 (r-squared), adjusted R^2, p-value describing the overall significance of the model and AIC (Akaike's Information Criterion) with the degrees of freedom.

As the models are nested, an ANOVA test was performed to check for model improvement. Tested null hypotheses were H0a: *model B does not improve the prediction of the outcome relative to model A* and H0b: *model C does not improve the prediction of the outcome relative to model B*. In both cases, function *anova* (base R) suggested the test results were significant. For H0a F (df=1) = 885.89, p-value < 2.2E-16, whereas for H0b F (df=5) = 90.69, p-value < 6.8E-12.

Table no. 5 – Summaries of main three regression models for randomized data

Model	Formula	R^2	Adj.R^2	p-value	AIC (df)
A	duration ^ .3 ~ 1.0983 + 0.6839 * avg_n_of_rows + 0.0143 * n_of_attribs_where + 0.0017 * length_attrib_string__select + 0.2517 * n_of_joins	0.30	0.29	1.27e-29	1758 (6)
B	duration ^ .3 ~ 1.2489 - 3.5143 * dbserverPostgreSQL + 0.6953 * avg_n_of_rows + 0.0284 * n_of_attribs_where + 0.0016 * length_attrib_string__select + 0.4680 * n_of_joins	0.76	0.76	1.89e-120	1327 (7)
C	duration ^ .3 ~ 2.26981 - 2.3951 * dbserverPostgreSQL + 0.3394 * avg_n_of_rows - 0.0386 * n_of_attribs_where + 0.0008 * length_attrib_string__select + 0.4198 * n_of_joins - 0.1826 * dbserverPostgreSQL:avg_n_of_rows - 0.0016 * dbserverPostgreSQL: length_attrib_string__select + 0.0194 * avg_n_of_rows:n_of_attribs_where + 0.0005 * avg_n_of_rows: length_attrib_string__select + 0.0001 * n_of_attribs_where: length_attrib_string__select	0.80	0.79	7.12e-128	1274 (12)

Figure no. 16 contains the two plots of the statistically significant interaction effects of predictor *dbserver* with predictors *avg_n_of_rows* (left) anf *length_string__select* (right). The plots were generated with package *effects* (Fox, 2003). In both cases the slopes differ between the panels corresponding to the server levels, suggesting that interaction between predictors is important in explaining the variation of the outcome.

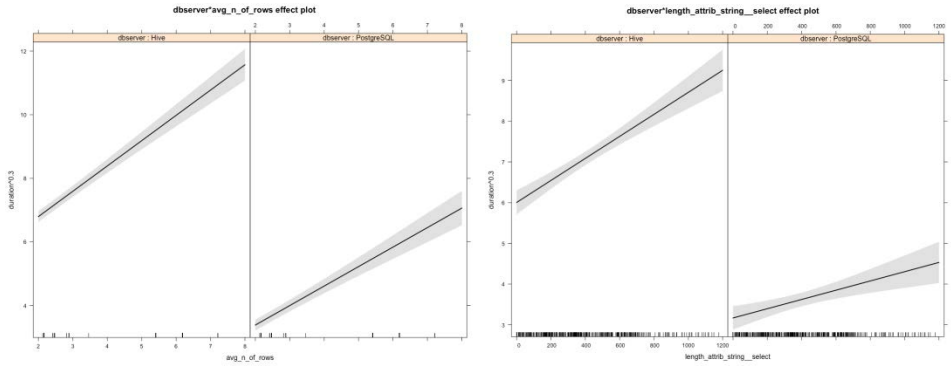


Figure no 16 – Visualize interaction effects with package effects

Another way to visualize the interaction between predictors is provided by package *interplot* (Solt *et al.*, 2016). It represents the conditional coefficients ("marginal effects") of variables included in multiplicative interaction terms. Function *interplot* plots the changes in the coefficient of one variable in a two-way interaction term conditional on the value of the other included variable, including a simulated 95% confidential intervals of these coefficients. Notably *interplot* plots the changes in the conditional coefficient of one variable in the interaction, rather than changes in the dependent variable itself. The chart in the left side of Figure no. 17 shows how the database server affected the coefficient for the predictor *avg_n_of_rows* on the outcome (*duration* ^ .3), whereas that in the right side of the figure shows how the database server affected the coefficient for the predictor *length_attrib_string__select* on the outcome (*duration* ^ .3),

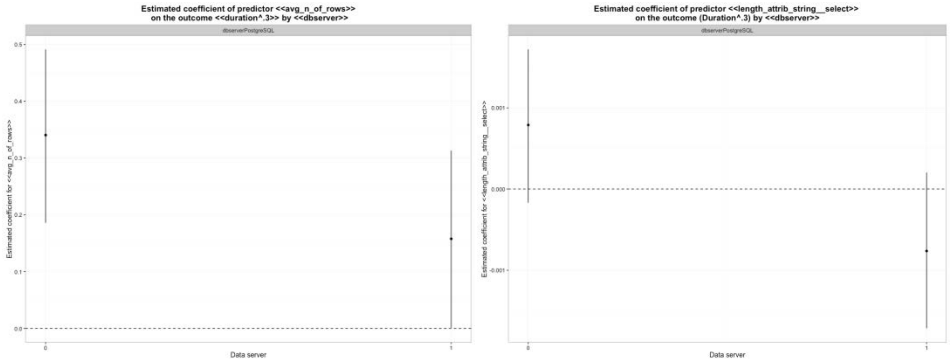


Figure no 17 – Visualize interaction effects with package interplot

Next the models A, B and C were tested for conformity with the Ordinary Least Square assumptions. Error independence of the models was questionable since function *durbinWatsonTest* in package *car* (Fox and Weisberg, 2011) reported autocorrelation (p-value = 0) in all the cases. Function *dwtest* in package *lmtest* (Zeileis and Hothorn, 2002) reported similar results. Also constant error variance (homoscedasticity) was not met by any of the models since the null hypothesis of homoscedasticity was rejected by function *ncvTest* of package *car* with p-values below 0.001. Results was confirmed with the studentized Breusch-Pagan test. Function *bptest* in package *lmtest* reported also p-values below 0.001.

Error normality was not met, both graphically (function *qqPlot* in package *car*) – see Figure no. 18 and with function *shapiro.test* (base R). Nevertheless, error departure from normality is not huge for all three models.

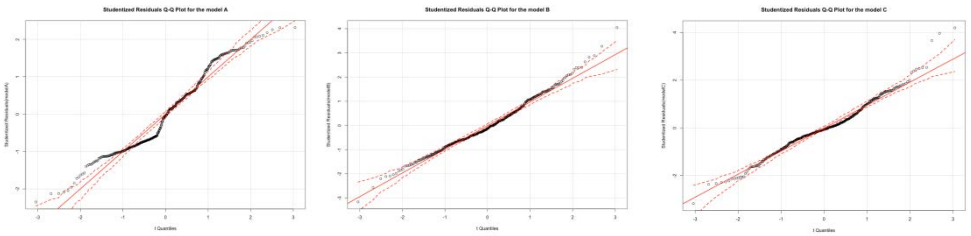


Figure no 18 – Error normality with qq-Plots for models A, B and C

Also models linearity is questionable in all three cases as the component plus residual plots show (see Figure no. 19).

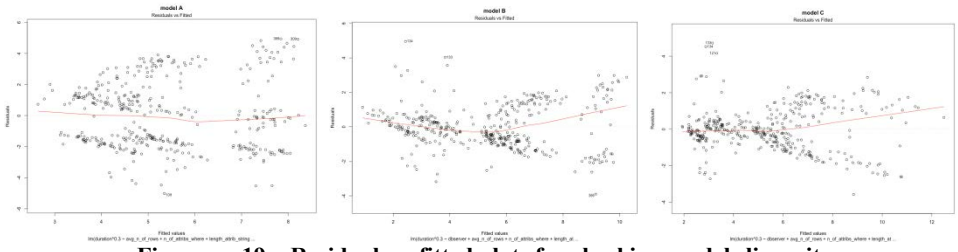


Figure no 19 – Residual vs. fitted plots for checking models linearity

There were some issues related to outliers, high leverage points and influential observations as seen in the graph of Figure no. 20 generated with function *influencePlot* in package *car*.

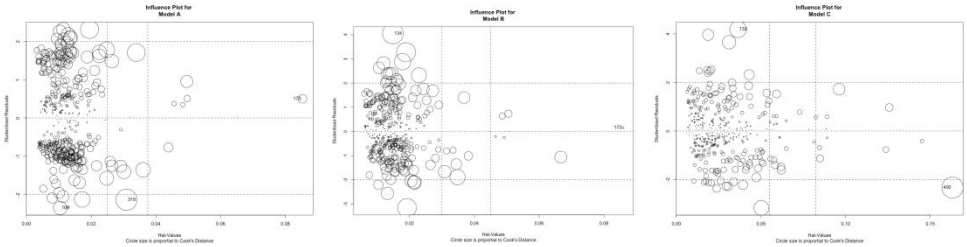


Figure no. 20 – Outliers, high leverage points and influential observations

Variance Inflation Factor (function *vif* in package *car*) did not report collinearity in models A and B. Multicollinearity manifested in model C can be explained by the interaction terms.

As the Ordinary Least Squares assumptions were not met, alternative regression methods were put in use in order to confirm or reject OLS findings. These methods were applied to model C as seemingly the most appropriate of the three models.

Whenever linear regression models incorporate error dependence it was suggested to use Generalized Least Squares GLS (Faraway, 2015, p. 113; Fox, 2016, pp. 474-495). Package *nlme* (Pinheiro *et al.*, 2016) is one option for building GLS models (function *gls*). GLS model C reported identical residual standard error and estimates for predictor coefficients.

As model C contained outliers, high leverage points and unusual observations which determined the non-normality of errors (residuals), robust regression (Fox, 2016, pp. 586-601) was applied for testing original OLS model C for significance. As the extreme erorrs were not numerous (see Figure no. 20), M-Estimation type of robust regression was preferred (Huber method). The mode was built with function *rlm* of package *MASS* (Venables and Ripley, 2002). Table no. 6 presents predictor estimated values, standard errors and t-values for "original" OLS model C and "Huber" model C.

Table no. 6 – Model C – Ordinary Least Squares vs. Huber (Robust, M-Estimation) Regression

Predictor	Model C - OLS			Model C - Huber		
	Coeff	Std.Err	t-value	Coeff	Std.Err	t-value
(Intercept)	2.2698	0.6938	3.2714	2.2972	0.6078	3.7797
dbserverPostgreSQL	-2.3951	0.2678	-8.9431	-2.1784	0.2346	-9.2852
avg_n_of_rows	0.3394	0.0773	4.3888	0.4127	0.0678	6.0910
n_of_attribs_where	-0.0386	0.0157	-2.4577	-0.0308	0.0138	-2.2389
length_attrib_string__select	0.0008	0.0005	1.6488	0.0008	0.0004	2.0072
n_of_joins	0.4198	0.1107	3.7930	0.3737	0.0969	3.8543
dbserverPostgreSQL: avg_n_of_rows	-0.1826	0.0602	-3.0311	-0.2566	0.0528	-4.8635
dbserverPostgreSQL: length_attrib_string__select	-0.0016	0.0004	-3.9811	-0.0019	0.0003	-5.6335
avg_n_of_rows: n_of_attribs_where	0.0194	0.0036	5.3714	0.0174	0.0032	5.5189
avg_n_of_rows: length_attrib_string__select	0.0005	0.0001	3.7442	0.0006	0.0001	5.9852
n_of_attribs_where: length_attrib_string__select	0.0001	0.0000	2.3600	0.0000	0.0000	2.2427

Another technique applied when OLS models expose problems is the non-parametric rank-based regression (Kloke and McKean, 2015, pp. 83-116). One of the available packages is *Rfit* (Kloke and McKean, 2012). Function *rfit* applied for model C found that the model is significant (Overall Wald Test = 2584.451, p-value=0) and produced predictor coefficients shown in Table no. 7.

Table no. 7 – Model C predictors when applying non-parametric, rank-based regression

Predictor	Model C - OLS		Model C – Rank-based	
	Coeff	p-value	Coeff	p-value
(Intercept)	2.2698	0.0012	2.5217	0.0000
dbserverPostgreSQL	-2.3951	0.0000	-2.1984	0.0000
avg_n_of_rows	0.3394	0.0000	0.3784	0.0000
n_of_attribs_where	-0.0386	0.0144	-0.0307	0.0153
length_attrib_string__select	0.0008	0.1000	0.0010	0.0101
n_of_joins	0.4198	0.0002	0.3363	0.0002
dbserverPostgreSQL: avg_n_of_rows	-0.1826	0.0026	-0.2223	0.0000
dbserverPostgreSQL: length_attrib_string__select	-0.0016	0.0001	-0.0019	0.0000
avg_n_of_rows: n_of_attribs_where	0.0194	0.0000	0.0184	0.0000
avg_n_of_rows: length_attrib_string__select	0.0005	0.0002	0.0006	0.0000
n_of_attribs_where: length_attrib_string__select	0.0001	0.0188	0.0000	0.0292

Differences in coefficient size for the predictors are minor. All of the models built using alternative techniques (other than OLS) displayed similar sign, scale and statistical

significance for predictor coefficients. Consequently, even if model C did not fit the OLS assumptions, it seems appropriate for the given the data set.

6. DISCUSSION

Big Data umbrella covers a large range of technologies for storing and processing large amounts of data produces at great speed (velocity) in a variety of formats (section 1). This paper compared two of the most popular products of "traditional" RDBMS's and "in vogue" Hadoop systems. Postgres XL (as PostgreSQL fork) was chose instead of standard PostgreSQL distribution, as it is natively a distributed RDBMS and (almost) fully compatible with the SQL dialect implemented in PostgreSQL (section 2.1).

Hive was preferred as Hadoop platform for testing, as it implements, as it provides a decent level of data independence and SQL-like query language – HiveQL (section 2.2).

Both systems were installed and deployed on a five-node Amazon cluster (main features are described in section 4.3) so that the results are fully comparable.

Relative to the current benchmarks for comparing data stores performance, this paper focused on testing queries generated randomly so that statistical tools could be (more) relevant. Currently the TPC-H benchmark provides 22 queries to be executed on different scale factors (section 3). Data about query execution (mainly query completion duration) is recorded for each tested data store and then compared. The 22-query set makes the results prone to be compared as this is the main goal of a benchmark. But is this 22-query set representative for the population of all of queries executed on a transactional (sales) database for a "typical" company/organization? The answer to this questions largely depends on each organizations information landscape (data structure, reporting requirements, data analysis etc.).

As the query set was nominated (supposedly based on authors' expertise) by TPC-H, the lack of queries randomness and their low variability makes the results less appropriate for statistical analysis. Moreover, generally papers on data stores performance have dealt mainly with basic operations (database load, update, or query) relating the results to just basic parameters, such as database size, number of concurrent used, etc.

This paper tested the performance of Postgres XL versus Hadoop/Hive on randomly generated 100 query set for each scale factor (section 4.1) with a total of 400 executes queries. Data was generated for four scale factors of TPC-H database schema using DBGen utility (section 3). As the record sets of each scale factor is independent of the other scale factors), each query targeted a specific scale factor subschema (section 4.1).

The finer-grained analysis of results was devised by recording not only the general information about query execution (duration, scale factor, data store), but also a variety of parameters about each query (section 4.2). As queries were randomly generated, most variables have a positive skewed distribution (section 5.1). Values for variables number of joins and number of tables (these two parameters are highly correlation since generated queries did not contain self-joins) are highly concentrated on value 7 (8), since even for queries with a small number of attributes an entire join chain was necessary to ensure query validity (functionality).

For expressing the data size, four parameters were available: the minimum, maximum, average and median number of processed rows by the query. Of four, only *avg_n_of_rows* was kept in the analysis since it has the largest variability of four (section 5.1).

Overall results of query execution duration contradicted what was initially expected (al least by the authors), as seen in Figure no. 13. Duration range was quite large in Hive, from 0 to about 6000 seconds with an average around 1000 seconds and a median about 500. The distribution was skewed to the right as further normality tests would confirm. By contrast, the distribution of overall query duration was more concentrated in PostgreSQL. The range varied between 0 and 1300, the average was placed around 150 seconds and the median was about 60 seconds. As with Hive, PostgreSQL distribution was skewed to the right. Overlapping density curves suggested differences in the concentration of values for two data servers. That came as a surprise, as it was expected that, when data size increases, Hadoop/Hive will eventually surpass PostgreSQL in a distributed architecture. Statistical tests confirmed the significance of performance gap suggested visually.

Also in section 5.2, Figure no. 14 and Table no. 2 shows that with the increase of the database size, the performance gap between PostgreSQL and Hadoop/Hive did not shrink at all. Ratio of medians (of query duration) between Hive and PostgreSQL, overall and for each scale factor shown in Table no. 2 exposes a "low" for Hive at factor scale of 2 followed by a notable performance improvement for scale factor of 5. Statistical significance of the gap for each scale factor was confirmed by the tests deployed in section 5.2 (see also Table no. 3).

One can only speculate that with further increase of the scale factor, Hive will eventually outperform Postgres XL. But for the available data set and deployed testing platform, it is obvious that Hadoop/Hive performed poorly.

In section 5.3 the relationship between the outcome (query duration) and various query parameters as predictors was analysed with a series of regression models. As the distribution of the outcome variable was positively skewed, it was transformed (in subsequent regression models), from duration into *duration ^ .3* (the cube root of *duration* or $\sqrt[3]{duration}$). That does not alter models interpretability.

Using a step-wise procedure, in each iteration the most non-significant attribute (the attribute with the largest/non-significant p-value) was removed (Table no. 4). The process stopped when current linear model contained no attributes whose p-value was less than 0.05. Resulted model was the baseline (*model B*). Model B predictors was tested for collinearity (Figure no. 15) and results did not display high correlation among predictors.

For assessing the importance of nominal predictor *dbserver* a simpler model (*model A*) was drawn from the baseline model by simply removing the predictor. Predictor *dbserver* proved to be pivotal, since by adding it from model A to model B, model adjusted R^2 increased from 0.30 to 0.76 (Table no. 5). Also Akaike's Information Criterion (AIC) decreased from 1758 to 1327 and ANOVA test for nested models ($F(1) = 885.89$, p-value < 2.2E-16) suggested that model B (incorporating *dbserver* variable) models better the outcome variability as a function of predictors for the given dataset. That confirmed the results in section 5.2.

Interactions were introduced in model C because it was expected that the performance differences could vary in each data system on different levels on certain predictors. Two predictors are said to interact in determining the outcome when the partial effect on one depends on the value of the other. Interaction refers to the manner in which predictors combine to affect the outcome, not the relationship between predictor themselves (Fox, 2016, p. 141). Model C conformed to the principle of marginality that specifies that a model including interactions should normally include the main effects that "compose" the interactions (Fox, 2016, p. 144). This is referred also as the hierarchical principal (James *et*

al., 2014, p. 89). But generally the main effect of predictors that interact are neither tested nor interpreted (Fox, 2016, p. 144).

Five interactions proved to be significant (see Table no. 5):

- Between *dbserver* and *avg_n_of_rows*
- Between *dbserver* and *length_attrib_string__select*
- Between *avg_n_of_rows* and *n_of_attribs_where*
- Between *avg_n_of_rows* and *length_attrib_string__select*
- Between *n_of_attribs_where* and *length_attrib_string__select*

By adding interactions, from model B to model C, model adjusted R^2 increased from 0.76 to 0.79 (Table no. 5) which was quite spectacular, but notable. Model Akaike's Information Criterion (AIC) decreased from 1327 to 1274 and ANOVA test for nested models ($F(5) = 90.69$, p-value < 6.8E-12) also suggested that model C (incorporating above interactions) models better than model B the outcome variability.

As this paper targeted mainly performance comparison between Hive and Postgres XL first two interaction terms were particularly important. On the two plots in Figure no. 16 the slopes differ slightly between the panels corresponding to the server levels, suggesting that interaction between predictors *dbserver* and *avg_n_of_rows* (left) and between predictors *dbserver* and *length_string__select* (right) are relatively important in explaining the variation of the outcome. Also in the interaction plot on the left side of Figure no. 17 shows that with the "increase" of the data server (from 0 for Hive to 1 for Postgres) along the x axis, the magnitude of the coefficient of predictor *avg_n_of_rows* decreases (along the y axis). Plot on the right side of Figure no. 17 suggests that with the "increase" of the data server (from Hive to PostgreSQL) along the x axis, the magnitude of the coefficient of *length_at-trib_string__select* decreases along the y axis.

Model C seemed the best (among the three) in explaining the duration main drivers. It makes sense in terms of sign and size of predictor coefficients. Following the hierarchical/marginality principle, individual predictors occurring in interaction terms are not interpreted. It is the case of *dbserver*, *avg_n_of_rows*, *length_attrib_string__select*, and *n_of_attribs_where*. Regression coefficient of predictor *n_of_joins* is considerable. Every additional join in a query seemed to increase the cube root of *duration* ($\sqrt[3]{duration}$) with .4. The size of coefficient for this predictor is explained by its narrow range of values (from 0 to 7). Coefficients of interaction terms *dbserverPostgreSQL:avg_n_of_rows* and *dbserverPostgreSQL:length_attrib_string__select* were negative. As already suggested by the interaction plots, performance in Hadoop/Hive worsened (compared to Postgres XL) when database size increased and also when the query contains longer attributes of type string (CHAR and VARCHAR).

Since OLS assumption on error independence, error normality (Figure no. 18), model linearity (Figure no. 19) and outliers, high leverage points and influential observations (Figure no. 20) were not checked, three alternative methods were put in use for confirming or rejecting OLS results for model C:

- Generalized Least Squares GLS
- M-Estimation (Huber method) robust regression (Table no. 6)
- Non-parametric rank-based regression (Table no. 7)

Results of applying all these three methods were in line with OLS model C (in terms of model significance, parameter size, sign and significance).

7. CONCLUSIONS AND FURTHER RESEARCH

This paper results suggest that on a five-node distributed architecture deployed in cloud, for an average size of the database, Hadoop/Hive (as one of flagship Big Data technologies) failed to outperform the RDBMS contender (Postgres XL). Results validity is supported by the randomness of the tested queries, the variability of the database size and the non-parametric tests deployed accompanied by the regression models built and tested for explaining the main drivers of query duration (the key measure of data processing performance).

The results must be reported strictly to the volume of processed data (0.5-5 GB which is quite typical for many companies), to the nature of tested queries (basically filtering, non-aggregate) and also the transactional nature of the database (TPC-H databases is actually a sales database). Tests must be further deployed on database schemas with larger scale factors, on more powerful architectures (more numerous nodes), and using also aggregate queries which are the main ingredient of data processing in business information systems.

As the default settings were used in testing the systems, another direction could refer to system tuning (i.e. indexes which would eliminate the costly full table scans) for both Postgres XL and Hadoop/Hive.

Also data analysis for explaining and predicting system performance can be improved by using mixed regression levels. Since for larger scale factors, Postgres XL – Hive performance gap is expected to narrow and eventually to be reversed, classical OLS and robust, non-parametric regression used in this paper could prove inadequate. Non-parametric regression models dealing better with non-linearity and correlation among predictors, such as MARS (Multivariate Adaptive Regression Splines), could be preferred instead.

References

Buhl, H. U., Röglinger, M., Moser, F., and Heidemann, J., 2013. Big Data. *Business & Information Systems Engineering, 5*(2), 65-69. doi: http://dx.doi.org/10.1007/s12599-013-0249-5

Cattell, R., 2010. Scalable SQL and NoSQL Data Stores. *SIGMOD Record, 39*(4), 12-27. doi: http://dx.doi.org/10.1145/1978915.1978919

Cogean, D. I., Fotache, M., and Greavu-Serban, V., 2013. NoSQL in Higher Education. A Case Study. In C. Boja, L. Batagan, M. Doinea, C. Ciurea, P. Pocatilu, A. Ion, R. Magos, L. Cotfas, A. Velicanu, C. Amancei, M. Andreica and A. Zamfiroiu (Eds.), *International Conference on Informatics in Economy* (pp. 352-360). Bucharest: Bucharest Univ Economic Studies-Ase.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R., 2010. *Benchmarking cloud serving systems with YCSB*. Paper presented at the 1st ACM symposium on Cloud computing (published in the Proceedings), Indianapolis, Indiana, USA. doi: http://dx.doi.org/10.1145/1807128.1807152

Doulkeridis, C., and Norvag, K., 2014. A survey of large-scale analytical query processing in MapReduce. *The VLDB Journal, 23*(3), 355-380. doi: http://dx.doi.org/10.1007/s00778-013-0319-9

Faraway, J., 2015. *Linear Models with R* (2nd ed. ed.). Boca Raton, FL: CRC Press.

Fotache, M., and Hrubaru, I., 2016. Big Data Technology on Medium-Sized Data. Preliminary Results for Non-Aggregate Queries. In C. Boja, M. Doinea, C. Ciurea, P. Pocatilu, L. Batagan, A. Velicanu, M. E. Popescu, I. Manafi, A. Zamfiroiu and M. Zurini (Eds.), *International Conference on Informatics in Economy, Ie 2016: Education, Research & Business Technologies* (pp. 273-278). Bucharest: Bucharest Univ Economic Studies-Ase.

Fotache, M., Strimbei, C., Hrubaru, I., and Cogean, D. I., 2014. *Scratching Big Data Surface: Comparing Simple Queries in PostgreSQL and MongoDB.* Paper presented at the 13th International Conference on Informatics in Economy - IE 2014 (published in the Proceedings), Bucharest, Romania.

Fox, J., 2003. Effect Displays in R for Generalised Linear Models. *Journal of Statistical Software, 8*(15), 1-27. doi: http://dx.doi.org/10.18637/jss.v008.i15

Fox, J., 2016. *Applied Regression Analysis and Generalized Linear Models* (3rd ed. ed.). Thousand Oaks, CA: Sage.

Fox, J., and Weisberg, S., 2011. *An R Companion to Applied Regression* (2nd ed. ed.). Thousand Oaks, CA: Sage.

Giraudoux, P., 2016. pgirmess: Data Analysis in Ecology. *R package version 1.6.5*. Retrieved from https://CRAN.R-project.org/package=pgirmess

Gross, J., and Ligges, U., 2015. nortest: Tests for Normality. *R package version 1.0-4*. Retrieved from https://CRAN.R-project.org/package=nortest

Hothorn, T., and Hornik, K., 2015. exactRankTests: Exact Distributions for Rank and Permutation Tests. *R package version 0.8-28*. Retrieved from https://cran.r-project.org/package=exactRankTests

Hrubaru, I., and Fotache, M., 2015. On a Hadoop Cliche: Physical and Logical Models Separation. In C. Boja, M. Doinea, C. Ciurea, P. Pocatilu, L. Batagan, A. Ion, V. Diaconita, M. Andreica, C. Delcea, A. Zamfiroiu, M. Zurini and O. Popescu (Eds.), *Proceedings of the 14th International Conference on Informatics in Economy* (pp. 357-363). Bucharest: Bucharest Univ Economic Studies-Ase.

Jacobs, A., 2009. The pathologies of big data. *Communications of the ACM, 52*(8), 36-44. doi: http://dx.doi.org/10.1145/1536616.1536632

James, G., Witten, D., Hastie, T., and Tibshirani, R., 2014. *An Introduction to Statistical Learning With Applications in R*. New York, NY: Springer.

Kejser, T., 2014. TPC-H: Data And Query Generation. from http://kejser.org/tpc-h-data-and-query-generation/

Kloke, J., and McKean, J. W., 2012. Rfit: Rank-based estimation for linear models. *The R Journal, 4*(2), 57-64.

Kloke, J., and McKean, J. W., 2015. *Nonparametric Statistical Methods Using R*. Boca Raton, FL: CRC Press.

Kowalczyk, M., and Buxmann, P., 2014. Big Data and Information Processing in Organizational Decision Processes. *Business & Information Systems Engineering, 6*(5), 267-278. doi: http://dx.doi.org/10.1007/s12599-014-0341-5

Li, F., Ooi, B. C., Ozsu, M. T., and Wu, S., 2014. Distributed data management using MapReduce. *ACM Computing Surveys, 46*(3), 1-42. doi: http://dx.doi.org/10.1145/2503009

Lublinsky, B., Smith, K., and Yabukovich, A., 2013. *Professional Hadoop Solutions*. Indianapolis, IN: John Wiley & Sons.

Lungu, I., and Tudorica, B. G., 2013. The Development of a Benchmark Tool for NoSQL Databases. *4*(2), 13-20.

Pavlo, A., and Aslett, M., 2016. What's Really New with NewSQL? *SIGMOD Record, 45*(2), 45-55. doi: http://dx.doi.org/10.1145/3003665.3003674

Pinheiro, J., Bates, D., DebRoy, S., Sarkar, D., EISPACK authors, Heisterkamp, S., . . . R-core team, 2016. nlme: Linear and Nonlinear Mixed Effects Models. *R package version 3.1-128*. Retrieved from http://CRAN.R-project.org/package=nlme

PostgresXL, 2016. Postgres XL Overview. Retrieved 10 September 2016, from http://www.postgres-xl.org/overview/

Sakr, S., Liu, A., and Fayoumi, A. G., 2013. The family of mapreduce and large-scale data processing systems. *ACM Computing Surveys, 46*(1), 1-44. doi: http://dx.doi.org/10.1145/2522968.2522979

Solt, F., Hu, Y., and Kenke, B., 2016. interplot: Plot the Effects of Variables in Interaction Terms. *R package version 0.1.5*. Retrieved from http://CRAN.R-project.org/package=interplot

Stonebraker, M., 2012a. New opportunities for New SQL. *Communications of the ACM, 55*(11), 10-11. doi: http://dx.doi.org/10.1145/2366316.2366319

Stonebraker, M., 2012b. What Does 'Big Data' Mean? . *Communications of the ACM (BLOG@CACM).*Retrieved 20 March 2016, from http://cacm.acm.org/blogs/blog-cacm/155468-what-does-big-data-mean/fulltext

Stonebraker, M., 2015. Hadoop at a Crossroads. *Communications of the ACM, 58*(1), 18-19. doi: http://dx.doi.org/10.1145/2686591

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., . . . Murthy, R., 2009. Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow., 2*(2), 1626-1629. doi: http://dx.doi.org/10.14778/1687553.1687609

Trancoso, P., 2015. *Moving to memoryland: in-memory computation for existing applications*. Paper presented at the Proceedings of the 12th ACM International Conference on Computing Frontiers, Ischia, Italy. doi: http://dx.doi.org/10.1145/2742854.2742874

Transaction Processing Performance Council - TPC, 2014. TPC Benchmark H Standard Specification Revision 2.17.1. 1-136. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

Venables, W. N., and Ripley, B. D., 2002. *Modern Applied Statistics with S* (4th ed. ed.). New York: Springer. doi: http://dx.doi.org/10.1007/978-0-387-21706-2

Wei, T., and Simko, V., 2016. corrplot: Visualization of a Correlation Matrix. *R package version 0.77*. Retrieved from http://cran.r-project.org/web/packages/corrplot/index.html

White, T., 2015. *Hadoop - The Definitive Guide* (4th ed.). Sebastopol, CA: O'Reilly Media.

Wickham, H., 2016. *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer. doi:http://dx.doi.org/10.1007/978-3-319-24277-4

Ylijoki, O., and Porras, J., 2016. Perspectives to Definition of Big Data: A Mapping Study and Discussion. *4*(1), 69-91.

Zeileis, A., and Hothorn, T., 2002. Diagnostic Checking in Regression Relationships. *R News, 2*(3), 7-10.