Automatic SQL to HQL-NoSQL Querying using PostgreSQL and Integrated Hive-HBase

OUDAY SAADA, JIHAD DABA
Department of Electrical Engineering,
University of Balamand,
LEBANON

Abstract: - The amount of digital data is constantly growing in almost all fields. This data is divided into two categories, structured and unstructured data. Non-structural databases known as NoSQL became one of the main fields of big data. Many companies are still using relational databases like PostgreSQL and MySQL. But with the rapid evolution and diversity of stored data, companies find themselves obliged to use big data tools like HBase or Hive. Big data is characterized by its capacity, speed, and ability to store diverse types of data. Data analysis and high storage capacity are the main reasons for companies to search for new database systems. Data migration to new systems is associated with the modification of the existing data and applications. This process costs a lot to adopt new specialists to handle this transition. Furthermore, due to different sources of data in old systems, e.g., real-time applications that are continuously collecting new data, companies will not be able to leave relational databases. For this reason, we present a system, termed Automatic Query Language, or AQL in short form, for migrating data from PostgreSQL to integrated HBase/Hive databases. In addition, we provide a platform that allows any user to query automatically PostgreSQL, Hive, and HBase databases using SQL query only. Querying the system is related to where each big data tool's performance is better. After the platform was completed, we were able to insert and select data from both relational databases and big data components. Join operation was not a problem because complex queries for analysis were executed using Hive which was integrated with HBase. The tested AQL system proved that HBase can insert data with more efficiency than PostgreSQL and Hive, and that select query in Hive has a better performance than PostgreSQL for big data size, whereas, for small data size, the performance of PostgreSQL is better.

Key-words: - Automatic Query Language, Big data, HBase, HDFS, Hive, PostgreSQL, Relational Database, Sqoop

Received: April 11, 2022. Revised: November 15, 2022. Accepted: December 16, 2022. Published: January 26, 2023.

1 Introduction

With the huge evolution in technology, the amount of data is getting bigger in all enterprises. The explosion of data happens when companies are producing valuable types of data in a non-measurable way and at speed.

Researchers have intensified their research work in this field to improve the ability to benefit from these data. Most companies that are using relational databases did not expect this big evolution of the amount of data, and they found themselves in a situation where relational databases cannot process this vast amount of diverse data anymore.

Big data tools like HBase and Hive have become the alternative solution for dealing with large quantities and diversity of data because of their capabilities of handling volume, value, variety, velocity, and veracity. In addition, big data and RDBs integration became very popular with the understanding that there is no universal solution.

Integration of heterogeneous systems needs experts with good knowledge to benefit from each data store. Also, the original system architecture could be impacted by the database integration, [1], which is also a limitation. As is happening in almost all companies, End-user applications remain connected to RDBMSs. Applications are not supported for transformation to new systems because they only understand SQL language and they need experts with new programming knowledge to handle this work.

The main problem in migrating a relational database into HBase, that is, a NoSQL database, is a JOIN operation between two or more tables. JOIN

operation is of interest to many researchers as it could be faster but not as powerful as relational databases. HBase is integrated with Hive which is a SQL-like interface because of two reasons:

- HBase does not support the JOIN operation.
- Hive is the king of relational data analysis

In this paper, we propose a way of integrating relational databases with integrated HBase/hive. Our system is termed AQL. Data migration from a relational database into an integrated table is handled in addition to a platform where users can insert and retrieve data from both relational databases and integrated Hive/Hbase tables. The main parts are listed below.

(1) Data conversion: It offers a way of data migration from PostgreSQL to HBase integrated with Hive. This migration is done using Sqoop with JDBC driver which connects both databases. This process allows load reduction on the existing system (PostgreSQL) which leads to better performance.

(2) Automatic SQL Interface for PostgreSQL HBase/Hive: A new platform is designed to provide users with the ability to execute any SQL command in both PostgreSQL and integrated HBase/Hive. This platform allows any user with SQL knowledge to insert one row or retrieve data from both PostgreSQL and integrated HBase/hive tables at the same time.

This work is needed because it facilitates the process of analyzing data more efficiently in companies that are not able to completely leave existing systems. This work also helps to prevent the existing applications from losing new data. In addition, integrating Hive and HBase improves data testing, which is done using high server speed.

There are other snip-offs to this work. Big data applications in satellite imaging systems, [2], [3], [4], [5], [6], [7], [8], [9], and more generally in coherent imaging systems, [10], [11], have dramatically increased lately. Satellite companies provide commercial high-resolution muti-spectral images, and computer companies deliver powerful cloud facilities to process them. Most notable is NASA's Landsat mission with over 2600 operational imaging satellites orbiting Earth and relying on machine learning and artificial intelligence (AI). The number of images acquired is overwhelming and requires big data storage and strong analytics. Since satellite onboard computers are not powerful enough for big data processing or storage, these tasks are handled by ground stations that are equipped with powerful processors, tailoring big data analytics to specific clients' needs.

Big data also find increasingly important applications in next-generation wireless networks. The evolving 5G and the planned 6G networks employ at their core massive multiple-input-multiple-output (m-MIMO) systems driven by orthogonal frequency division multiplexers (OFDM), [12], [13], [14], [15], [16]. Such systems inevitably generate massive data, and to procure useful analytics from m-MIMO resources, a big data-aware 5G/6G mobile communication system needs to be developed. A careful selection of big data analytics allows for the integration of big data with m-MIMO systems to improve spectral efficiency.

The paper is structured as follows: In section 2, the software and hardware specifications are discussed. In section 3, the system work process of AQL is explained. In section 4, the testing and results of our work are presented. Section 5 summarizes the conclusions of this work and lays the path for future work.

2 Software and Hardware Specification

In this work, we use open-source software mounted on specific hardware as shown in the sub-sections below.

2.1 Software Specification

All services are installed using Ubuntu 18.04 machines. Python is the main prerequisite for Hadoop tools. In our work, we employ Python 3.7. Ambary server (2.7.4.0) installation is used to install Hortonworks (HDP 3.1.4.0). One of the biggest contributors to the open-source big data analytics community is Hortonworks.

The services that are installed are:

- HDFS (Hadoop Distributed File System) version 3.1.1
- Yarn version (3.1.1)
- MapReduce2 version (3.1.1)
- Tez version (0.9.1)
- Hive version (3.1.0)
- HBase version (2.0.2)
- Sqoop version (1.4.7)
- Zookeeper (3.4.6)

PostgreSQL v.12 and pgAdmin4 are already installed on the existing system. PuTTY is used to access each machine from any connected workstation.

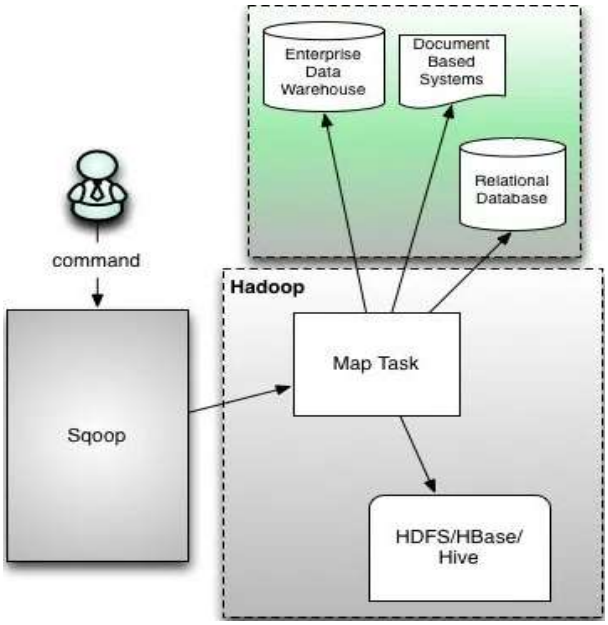
After installing all software, we can start data migration.

Finally, to connect PostgreSQL with the Hortonworks data platform, PostgreSQL JDBC 4.2 driver is installed.

2.2 Hardware Specifications

Hive JOIN performance improves using a higher number of CPUs, [17], and RAM capacity by 21% to 25%. In our study, we use a 3-nodes cluster where one node is the master and the other two are slaves. Each node is mounted on a single server with 60GB RAM, 500GB SSD drive, and 16 CPU core Intel® Xeon Gold 6132@2.60GHz.

Fig. 1: Apache Scoop architecture [18]



3 AQL System Work Process

Once the environment is ready, we can start data migration into integrated HBase and Hive tables. PostgreSQL data presented in our research institution that took years to be collected are used for testing. HBase and Hive integrated tables are created first, then data migration from PostgreSQL into integrated HBase/Hive tables is started. After data is migrated, we can start testing insert and select queries time using different sizes of data.

3.1 Data Migration

Apache Sqoop is a big data tool that allows data migration between RDBs and big data components like Hadoop, Hive and HBase, as illustrated in Fig. 1.

Sqoop uses a map program only when launching many mappers and depends on user requirements to transform imported data in a way that HDFS can understand (since most of the big data components are based on HDFS). Every mapper creates its own connection to the target database using the JDBC connector to import its assigned data.

A method for loading data in-path from a file into integrated HBase/Hive is provided in [19]. The data is stored in a single HBase table with a unique row key, where querying is compared with the same data on different Hive tables. However, there are three ways of migrating data from PostgreSQL into HBase:

(1) Migrate all PostgreSQL-related tables into a single table in HBase. Each migrated table is assigned to an independent column family in the same table.
(2) Create for each PostgreSQL table a distinct relative table in HBase where column fields are divided into different column families depending on fields name relations.
(3) Create for each table a relative table in HBase where all table columns are under the same column family.

Many researchers worked on JOIN operations using an HBase table where all related tables are stored in the same HBase table using different column families. This is a limitation since it needs more refactoring and costs. In our work, we migrate data from PostgreSQL into HBase using the third scheme where each PostgreSQL table is considered as a table in HBase, and data is stored in a single column family name having the same HBase table name.

Let us consider that we have a geometry table named “building” for all buildings in a country under a schema named “GIS” under a database called “Lebanon”. To migrate this table, we need first to create a table in HBase. A column family named as table name is assigned to the created table:

```
create '<country==>table name>','<country==>column family>'
```

After creating the table and the assigned column family, we can start data migration using the Sqoop command where the JDBC connection is established between the databases as stated below.

Sqoop Import command:

```
connect jdbc:
PostgreSQL://localhost or ip
/database_name
username root
Password
table table_name (postgresql table)
map-column-java 'geom'=string
(because HBase does not support
geometry type)
HBase-table (postgres_schema_name)
column-family table_name (postgres
table name)
columns "id, geom, fid, objected,
textstring, point_x,
point_y.معلومات
schema Lebanon
HBase-row-key id -m 1
```

Because we are working here on a defined number of local machines, it is preferred to use only one mapper. After the data is migrated into HBase, the next step is to integrate it with Hive. We have to create an external table in Hive considering that the table has eight fields, including one Arabic field. We need to declare a process on how to transfer tables having Arabic field names into Hive that does not accept Arabic fields. In addition, this integration will not affect the old system data since Hive is a relational-based table, and as mentioned in [20], traditional databases migrate to a combination of SQL and NoSQL databases, adding flexibility, mobility, and efficiency to the characteristic of the system. The creation of a Hive table is stated below.

Create Hive table process:

```
CREATE EXTERNAL TABLE (id int, geom
string, fid string, objectid
string, textstring string, point_x
string, point_y string, maalumat
string) STORED BY
'org.apache.Hadoop.Hive.HBase.HBase
StorageHandler' WITH
SERDEPROPERTIES
("HBase.columns.mapping" =
":key,buildings:geom,
buildings:fid, buildings: objecid,
buildings:textstring,
buildings:point_x, buildings:
point_y,buildings: معلومات")
TBLPROPERTIES ("HBase.table.name" =
"country"); //to assign which HBase
table.
```

Each field created in the Hive table should have its map field in hbase.columns.mapping. Hive does not accept the Arabic language that's why we created a field named "maalumat" (Arabic for "information") or in any language that is supported by Hive.

In this way, we have the same data on both HBase and Hive tables which allows us to benefit from the features of both databases. In addition, for testing purposes, we use the same data on both PostgreSQL and integrated HBase/Hive. Later on, we can delete the existing data on PostgreSQL to improve its performance. In addition, data that are used from a persistent user's application for example should not be deleted to avoid application failure.

3.2 AQL Insert Process

Data migration into new systems and querying are not easy as they may appear. Experts find themselves obliged either to work with NoSQL and old system relational databases separately or to combine them to provide more accuracy in the system. The second choice is more complex since it needs experts with high knowledge about both systems.

In our work, we employ the second scenario. Data presented in relational databases are deleted to improve relational databases and existing applications' performance. Users with ordinary SQL knowledge can deal with the transferred data in our platform termed AQL, allowing any user to execute insert and select queries. The user query targets the one with the best performance and results. For example, insert statement targets HBase table knowing that HBase is better for real-time applications. This is where the role of integrating HBase with Hive becomes important, where every insert statement in HBase is automatically updated in hive tables where it can be retrieved later.

SQL insert statement is divided into three parts where the user specifies the table name, fields and the values to be inserted. Insert statement is a limitation for Hive because of the high delay in query execution proved later in the last section of this paper. The SQL insert process is stated below.

SQL insert query:

```
INSERT INTO table_name (column1,
column2, column3, ...)
VALUES (value1, value2, value3,
...);
```

While HBase is less complex than PostgreSQL referring to the tables' design, it is more complex for

querying data. HBase inserts the query depending on many variables as stated below.

```
Input
'<HBase_table_name>','row_key','<co
l_family:column_name>','<value>'
```

The insert statement in our platform, as shown in Fig. 2, allows users to insert one row at a time in the existing database.

The AQL insert form is divided into four parts. The user should insert:

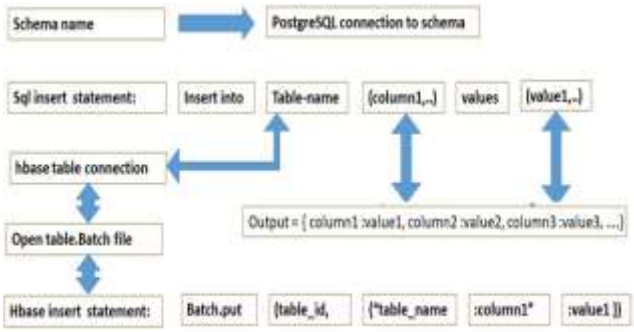


Fig. 2: AQL insert form

- The name of the schema needed to connect the database and redirect to the form of Fig. 3.
- The table name that should exist in the schema.
- The table fields with the existing ID column as the primary key at the beginning of the table.
- The values that need to be saved in the field.

The user is automatically connected to the schema it enters. After the connection is established, the table name should exist inside the schema. Fields entered by the user should have the same number as Postgres table fields. From the data entered by the user, a NoSQL query is automatically generated to insert this row in HBase, depending on user demand.

When a user chooses to insert a row in HBase, the table name will be the key to establishing the connection to the HBase table that has the same name. In addition, the table name will open a batch file having the same table name concatenated with (.batch) as (table-name.Batch). Each insert query is saved before it is migrated to HBase. The architecture of the AQL HBase query work is shown in Fig. 3.

Fig. 3: AQL HBase query work architecture

The insert query in HBase is divided into *n* numbers of queries which depend on the number of fields needed to be inserted. For example, if a table has 10 fields to be inserted, 10 queries will be saved in the batch file before it is executed and imported into HBase.

Insert query in HBase depends on storing SQL query as (key: value). The key is the column name and the value is the value referred to the same column name.

Query of data insertion is conduced as:

```
Batch.put(
id.{"table_name:column_name":
value}
```

Each insert query contains the same ID of inserted rows in PostgreSQL.

We can extract the size of the batch file as much as we want before it is executed in HBase. For example, we can resize the batch file to handle 1000 queries for the same table before it is imported into HBase.

When working on data migration and querying using multiple data stores, the insert statement should be targeted into one data store to avoid data overlapping.

3.3 AQL Select Process

Since the select query is used for data analysis, it is one of the main reasons behind using integrated HBase with Hive instead of using only NoSQL. Many researchers work on retrieving data from relational tables stored in NoSQL databases using different methods. For example, the researchers in

[21] stored the whole relational databases using the same HBase table under different column families. Also in [22], the author provided a solution for HBase to support JOIN operations to save data from different relational tables stored in the same HBase table, but no matter how much the system is strong, query efficiency will not be as effective as Hive, especially for complicated queries using JOIN operation.

Hive has many limitations in using insert, update, and delete, but when it comes to selecting, Hive is the engine of data analysis.

The second reason for using integrated Hive with HBase is that when it comes to real work, JOIN operation using HBase performance will not be as high as that of Hive. It is well known that HBase is not designed for accepting RDBMS tables. In addition, the researchers in [23] mentioned the profoundness to deal with NoSQL data stores because of the lack of a single API and query language. HBase RAM caching is detailed below.

- Hfile: Contains data and index of table and metadata of that data.
- Memstore: One of two memory cache where the region server is in-memory storage, data is accumulated until it is full, then it writes data to a new HFile on a named disk (Flushes).
- Blockcache: One of two memory cache and it is responsible for writing the retrieved data of HDFS.
- Block: Each HFile is composed of a series of small blocks.
- Write Ahed Log(WAL): A log file that saves every change in the data until it is written to the disk and prevents data loss before Memstore is written to the disk.

JOIN operation is the main problem for NoSQL database querying systems and the most used operation in database systems, [24]. Many researchers are working to solve this problem using different techniques. The work in [19] has no issue in JOIN operation using HBase because data is integrated into a single table in HBase. It is stated in the HBase reference guide, [25], that when the number of integrated tables in HBase gets larger than three column fields, it causes performance degradation and it is one of many limitations in HBase. To be more exact, any small PHP application will generate more than 4 related tables depending on the design that can generate different column numbers and sizes. Any small application today can generate a moderate number of columns with different sizes.

When a client creates a column family in the HBase table, a new Memstore in the region server is created. Data is accumulated in MemStore until it is full and then it writes ("Flushes") where a new Hfile is created on disk. More flushes with more Hfiles will be presented on the disk.

Generating too many column families causes too many Flush operations. This increase in the HFile number leads to the blocking of the region server.

In addition, more columns in the same table leads to more unnecessary I/O operations. For example, the presence of many column families in the same table generates many queries, where each one targets one column family. Cao et al in [26] state that querying HBase with multidimensional queries are not efficient. This increase in the query statements for the same column family leads to a time delay.

Flush operations will also increase if the column families number has different data sizes. For example, if the first column family has 10000 rows and the second one has 1000 rows, this will lead to more HFile splits, creating more small files. This operation also leads to system degradation.

It is common knowledge that HBase is faster than Hive in real-time applications. But for analyzing data, Hive is the better choice since it uses the MapReduce program and is compatible with relational tables. It is also common knowledge that HBase is not made for accepting relational databases. The testing done before using HBase to JOIN table was mostly done using 2 or 3 tables, which does not show any weakness in HBase performance. In addition, JOIN operation will be an issue in HBase if tables are stored using different VMs, [27]. In addition, if relational tables are stored in the same HBase table using different column families, there will be a limitation since it needs more refactoring, [28].

SQL Select query in our work is taken from users using Python (flask) application. Since the design of both PostgreSQL and Hive database tables are the same, the query first targets the PostgreSQL database and then the Hive tables.

JDBC connector for connecting Hive and PostgreSQL remains the same as in HBase. The select query retrieves PostgreSQL data and saves it to a table, then the query targets the Hive table and retrieves data which is added to the first table results. In this way, data is retrieved from migrated data and

at the same time from new data entered into the old relational system to avoid losing any data. The procedure mentioned above is depicted in Fig. 4.

4 Testing and Results

4.1 Insert Simulation and Results

It is well known that Hive is used for data analysis and not for real-time applications. Insert query in our platform is done using HBase where saved data can be retrieved also using Hive.

Our platform allows users to insert one row at a time in both PostgreSQL or integrated HBase/Hive. Query is executed using AQL in three database systems. The delay time for each query is recorded to compare the three systems. The first query is executed in HBase, the second query is executed in PostgreSQL, and the last one is executed in Hive. The insert query performances for recorded delays are shown in Figs. 5 and 6.

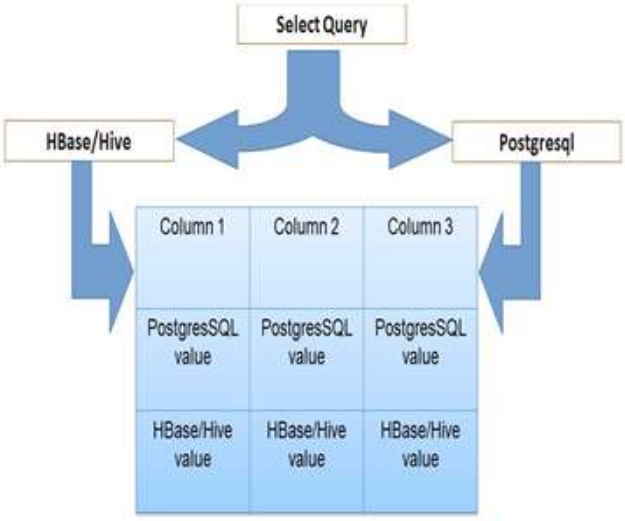


Fig. 4: Select from Hive and PostgreSQL

Figure 5 shows the delay for insert statement execution in Hive, PostgreSQL, and HBase. We notice that HBase has the best performance with a 9 ms delay, which is smaller than PostgreSQL and Hive delays with 0.09 s and 0.88 s delays, respectively. Our code can be used for real-time applications when companies want to transfer the whole system into big data keeping the same SQL insert queries. Considering that the first statement delay is always higher than the other, we exclude it from our study.

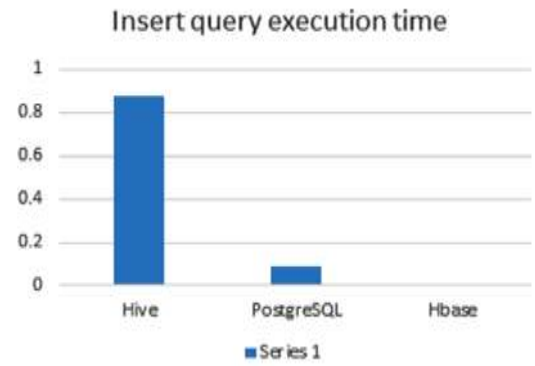


Fig. 5: Insert delay using one statement

We tested the system by repeating the insert query 10 consecutive times. For PostgreSQL, the ID of the insert row is auto-incremented with the DEFAULT value, whereas for Hive, we used the same query repeated 10 times.

In addition, HBase inserts the 10 queries in the BATCH file and executes all queries in one step. These 10 queries generate N number of queries, where N is equal to the sum of all fields in the total number of queries.

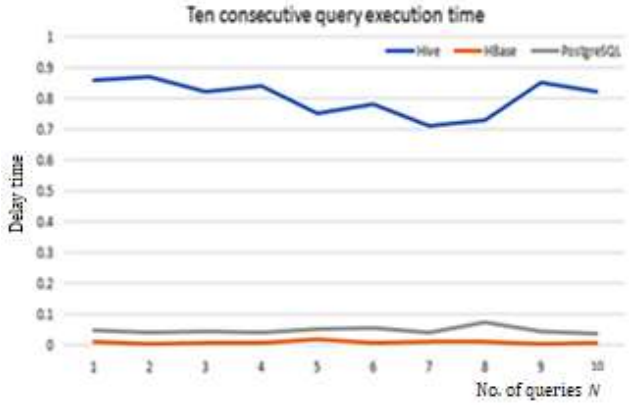


Fig. 6: Insert delay for ten consecutive statements

The average delay time in Fig. 6 is 0.803 s for Hive, and 0.0474 s for PostgreSQL, and the average time taken from HBase is added to the total time of batch file execution. The average time taken by HBase is 8.48 ms to write the queries inside the batch file, requiring 0.47 s to execute the file. The total average HBase time is thus T_{Avg} = 0.047/10 + 0.00848 = 0.05548 s.

The batch file execution process runs in the background of the system, which is why we can exclude it and the user will not experience this delay. But for the total time of HBase, insert delay

performance remains approximately the same as that of PostgreSQL.

4.2 Select Simulations and Results

In this section, we test select queries using two different queries on databases with different sizes. The database used in this simulation is taken from our institution which took years to collect.

Five tables are created before starting the simulation. The tables created for the simulation are shown in Table 1.

In our experiment, two queries are used. The first query retrieves the count of persons who need food, medicants, and the count of needed money for married persons that are unemployed, for rent or school payments, for persons who are older than 45 years, and for persons who should be living in a rented apartment in a specific cadaster. This is coded as follows:

Table 1. Created tables for simulation

Tables	Buildings	Section	Person	Status
Fields	id	id	id	person_id
	building_id	building_nb	section_id	cars
	plot_number	section_nb	name	type
	building_name	floor_nb	last_name	model
	street_name	owner_tenant	mother_name	lands
	geom	type_section	date_of_birth	others
	zone	size_of_section	phone_number	need_food
	cadaster		status_marriage	need_medicant
			date_of_entry	type_medicant
				need_money
				money_for

```
SELECT cadaster, count(need_food),
count(need_medicant), count
(need_money), count (money_for)
FROM properties p
JOIN person pn on (p.person_id =
pn.ID)
JOIN section s on (pn.section_id =
s.id)
JOIN building b on (s.building_nb =
b.building_id)
where
p.status_work = 'no'  and
      pn.status_mariage = 'yes'
and (p.money_for = 'rent' or
p.money_for = 'school')     and
      pn.date_of_birth <1975
and s.type_section='residental' and
b.cadaster='tripoly'
GROUP BY cadaster;
```

The second query retrieves the name, last name, phone number, building number, section number, cadaster, street name, building name, and date of birth from both Hive and PostgreSQL for persons who need food and money, who are older than 51 years, who are married, tenants and unemployed, in all cadasters. This is coded as follows:

```
SELECT pn.name, pn.last_name,
pn.phone_number, pn.date_of_birth,
s.building_nb, s.section_number,
b.cadaster, b.steet_name,
b.building_name
FROM person pn
JOIN properties p on (pn. ID =
p.person_id)
JOIN section s on (pn.section_id =
s.id)
JOIN building b on (s.building_nb =
b.building_id)
where
p.need_food = 'yes'
and p.need_money = 'yes'
and pn.date_of_birth <1970
and p.status_work = 'no'
and s.owner_tenant = 'tenant'
and pn.status_marriage = yes
GROUP BY pn.name, b.cadaster
ORDER BY pn.date_of_birth;
```

The simulation is done using three different sizes of databases. Both PostgreSQL and integrated HBase/Hive tables have the same size of data, which helps us understand the performance of each one after the execution of queries. As stated in [29], when data is less than 1 GB, the performance of relational databases is better, whereas when data gets bigger than 1 GB, Hive performance becomes better at almost all types of queries. Thus, three simulations were conducted using approximately the total data sizes shown below:

(1) Five tables of 0.25 GB total
(2) Five tables of 30 GB total
(3) Five tables of 400 GB total

The test returned six results, where we applied two queries for each of the three database sizes.

4.2.1 The First Simulation

In this section, we show the test of both queries on the first set of databases which has 0.25 GB as the total size.

Each query was executed 10 consecutive times, and the delay was recorded when each query was finished. Queries one and two for Hive and PostgreSQL are displayed in Figs. 7 and 8, respectively.

We notice from Fig. 7 that the needed time in Hive is more than in PostgreSQL. The average time needed is 1.691 s for PostgreSQL and 3.59 s for Hive. The average delay difference between them is 1.899 s.

In query two, we notice from Fig. 8 that the average time needed is 3.95 s and 4.5 s for PostgreSQL and Hive, respectively. The average delay delay time decreases to 0.55 seconds. This is due to the existence of groups by and order by clauses in the statement. Hive uses a MapReduce job, whereas in PostgreSQL data is run by order from top to bottom. Data with small sizes should not be analyzed using Hive because Hive needs more time to assign Map Reduce jobs before starting query execution. Because of this, data that is not big enough or that is expected to develop slowly should not be transferred into big data platforms.

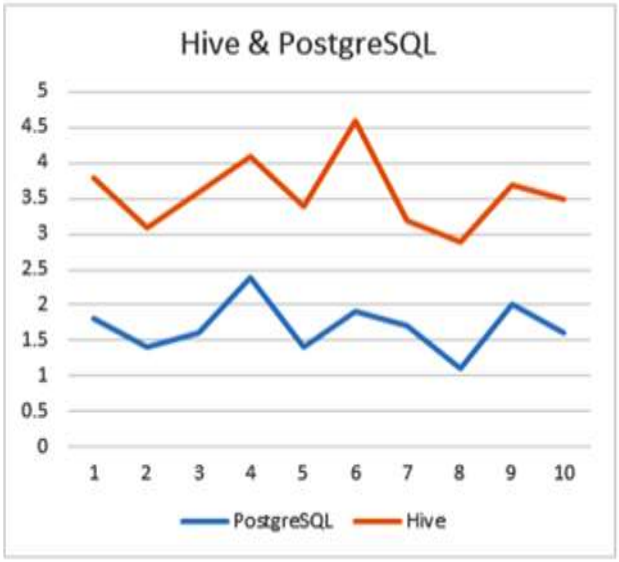


Fig. 7: Query one for Hive and PostgreSQL

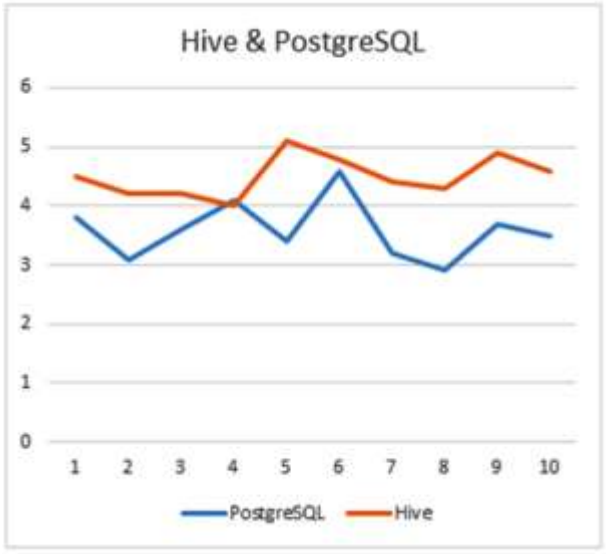


Fig. 8: Query two for Hive and PostgreSQL

In the study, we excluded the first query statement execution time because the delay was larger than the rest of the queries. This is because Hive takes time to establish the connection.

4.2.2 The Second Simulation

In this simulation, Hive and PostgreSQL have the same data size, each measuring 30 GB approximately. Both queries are executed in both systems and the results are depicted in Figs. 9 and 10.

As shown in Figs. 9 and 10, when the data gets larger, the delay difference between Hive and PostgreSQL increases. The average time needed for query one is 76.4 s and 137.1 s for Hive and PostgreSQL, respectively. When data gets larger up to 30 GB, the performance of Hive becomes better than PostgreSQL with an average difference time of 60.7 s.

In query two, the time to finish the query is on average 123.7 s and 237 s for Hive and PostgreSQL, respectively. The increase in query execution time is approximately 100%, but the average difference in delay decreased to 113.3 s.

4.2.3 The Third Simulation

In this simulation, Hive and PostgreSQL have the same data size, with each measuring approximately 400 GB. Both queries are executed in both systems and the results are displayed in Figs. 11 and 12.

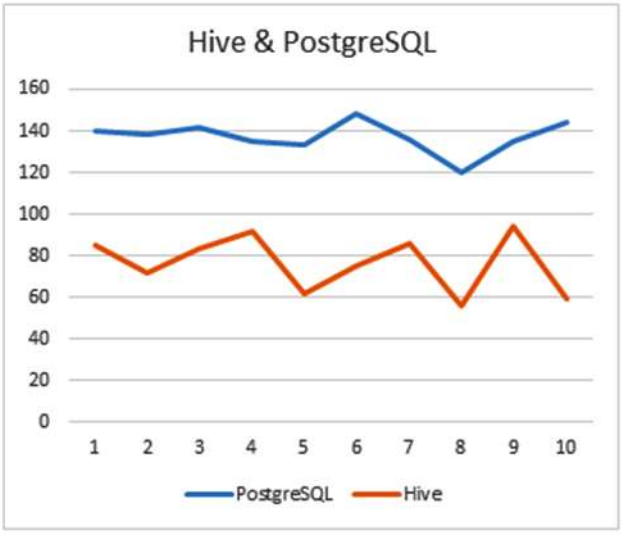


Fig. 9: Query one for PostgreSQL and Hive

As depicted in Figs. 11 and 12, when the data has a size of 400 GB, the delay difference between Hive and PostgreSQL gets larger than in the previous two experiments. The average time needed for query one for Hive and PostgreSQL is 885 s and 2098.7 s, respectively. When the data gets larger up to approximately 400 GB, the performance of Hive becomes better than that of PostgreSQL with an average difference time of 1213.7 s.

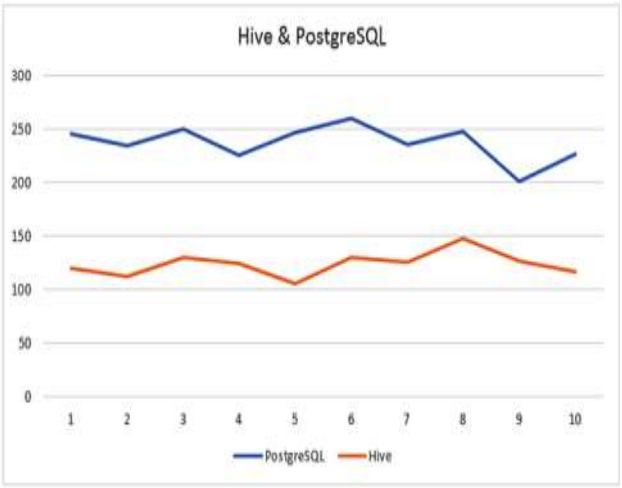


Fig. 10: Query two for PostgreSQL and Hive

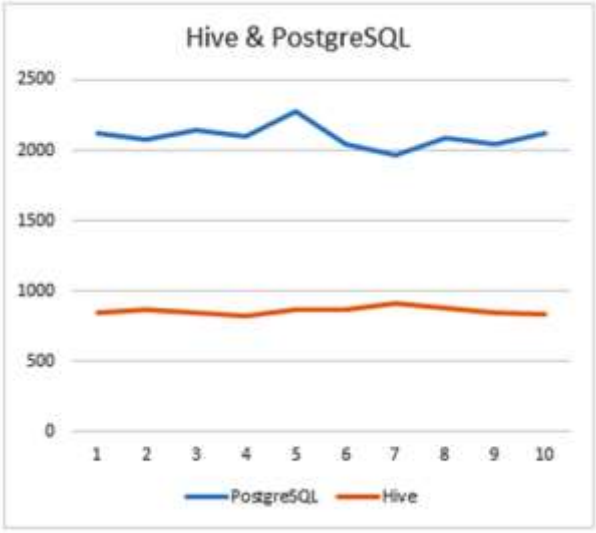


Fig. 11: Query one for Hive and PostgreSQL

Using query two, the time to finish the query has an average of 1112.4 s and 2863.7 s for Hive and PostgreSQL, respectively. The average difference time between both systems increases to reach a value of 1751.3 s.

We conclude that when data gets bigger, Hive performance increases compared to PostgreSQL. After testing the performance of Hive and PostgreSQL, the AQL system deletes the data from PostgreSQL which does not affect the existing applications.

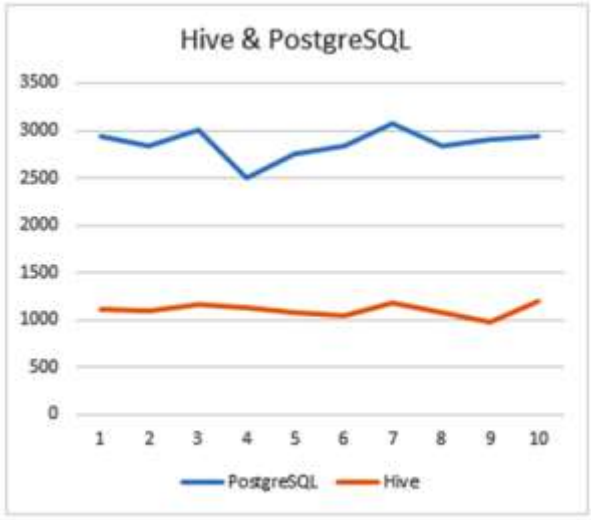


Fig. 12: Query two for Hive and PostgreSQL

5 Conclusion

In this work, we devise a new scheme for data migration and querying for companies that are stuck between old relational and big data systems. Data migration allows these companies to gain better performance in data analysis. Our novel AQL system allows users to insert data in both PostgreSQL and HBase as needed.

In addition, users can conduct Hive analysis which has the best performance for medium and large data sizes. Furthermore, newly inserted data in PostgreSQL is included in the analysis because our AQL system retrieves data from both databases.

We also noted that HBase is not made for relational databases and its performance will degrade with a higher number of tables.

Finally, we tested the system and proved that HBase can insert data with more efficiency than PostgreSQL and Hive. We also showed that the select query in Hive has a better performance than PostgreSQL for big data size, whereas, for small data size, PostgreSQL performance is better. We recommend that data migration into the Big Data system be only conducted if the data is huge enough or it is expected to increase with time.

References:

[1] Y. T. Liao, J. Zhou, C. H. Lu, S. C. Chen, C. H. Hsu, W. Chen, et al., "Data adapter for querying and transformation between SQL and NoSQL database," *Future Generation Computer Systems*, vol. *65*, pp. 111-121, 2016.

[2] J. S. Daba, M. R. Bell, "Object discrimination and orientation determination in speckled images," *Optical Engineering*, vol. 33, no. 4, pp, 1287-1303, 1994.

[3] J. S. Daba, M. R. Bell, "Estimation of the surface reflectivity of SAR images based on a marked Poisson point process model," *IEEE International Symposium on Signals, Systems, and Electronics* (*ISSSE*'95), San Francisco, CA, pp. 183-186, 1995.

[4] J. S. Daba, M. R. Bell, "Segmentation of speckled images using a likelihood random field model," *Optical Engineering*, vol. 47, no. 1, 2008.

[5] J. S. Daba, M. R. Bell, "Statistics of the scattering cross section of a collection of constant amplitude scatterers with random phase," *ECE Technical Reports*, Purdue University, p. 194, 1994.

[6] J. Dubois, "Scattering statistics of doppler faded acoustic signals using speckle noise models," *IEEE International Conference on Direct and Inverse Problems of Electromagnetic and Acoustic Wave Theory* (*DIPED*), Lviv, Ukraine, pp. 185-189, 2003.

[7] J. S. Daba, M. R. Bell, A. Abdi, S. Nader-Esfahani, "Comments on statistics of the scattering cross section of a small number of random scatterers," *IEEE Transactions on Antennas and Propagation*, vol. 48, no. 5, pp. 844-845, 2000.

[8] J. P. Dubois, O. M. Abdul-Latif, "SVM-based detection of SAR images in partially developed speckle noise," IEC, Prague, pp. 321-325, 2005.

[9] J. Dubois, "Poisson modulated stochastic model for partially developed multi-look speckle," *Proceedings of the American Conference on Applied Mathematics*, Harvard University, Cambridge, MA, USA, pp. 209-213, 2008.

[10] J.P. Dubois, O. M. Abdul-Latif, "Detection of ultrasonic images in the presence of a random number of scatterers: A statistical learning approach," IEC, Prague, pp. 326-329, 2005.

[11] J. Dubois, "Segmentation of speckled ultrasound images based on a statistical model," *EURASIP Proceedings of the 16th International Biosignal Conference*, Czech Republic, vol. 16, 2002.

[12] J. Dubois, O. M. Abdul-Latif, "Novel diversity combining in OFDM-based MIMO systems," *Proceedings of the American Conference on Applied Mathematics*, Harvard University, Cambridge, MA, USA, pp. 189-194, 2008.

[13] J. Dubois, O. Abdul-Latif, "Improved receiver diversity processing over SIMO fading channels," *Proceedings of the IEEE International Conference on Signal Processing and Communications*, 2007.

[14] O. M. Abdul-Latif, J. Dubois, "Performance of UWB system in a partially developed fading channel with CCI," *Proceedings of the 5th IEEE GCC Communication and Signal Processing Conference*, Kuwait, pp.1-5, 2009.

[15] J. Dubois, "Traffic estimation in wireless networks using filtered doubly stochastic point processes," *Proceedings of IEEE International*

Conference on Electrical, Electronic, and Computer Engineering, Cairo, Egypt, pp. 116-119, 2004.

[16] J. Dubois, J. S. Daba, M. Nader, C. El Ferkh, "GSM position tracking using a Kalman filter," *International Journal of Electrical, Computer, and Communication Engineering*, vol. 6, no. 8, pp. 867-876, 2012.

[17] T. Dokeroglu, M. S. Cınar, S. A. Sert, A. Cosar, "Improving Hadoop hive query response times through efficient virtual resource allocation," *Flexible Query Answering Systems*, Springer, Cham, pp. 215-225, 2015.

[18] S. Vithal, "*Sqoop architecture – mappers with no reducers*," Feb. 2018. https://dwgeek.com/sqoop-architecture.html

[19] S. Zulfiqar, A. Faridoon, M. Imran, "A novel architecture to integrate multi-source data into the distributed environment using big data infrastructure," 15*th International Conference on Emerging Technologies* (*ICET*), IEEE Press, pp. 1-6, Dec 2019.

[20] M. V. Sokolova, F. J. Gómez, L. N. Borisoglebskaya, "Migration from an SQL to a hybrid SQL/NoSQL data model," *Journal of Management Analytics*, vol. *7*, no.1, pp.1-11, 2020.

[21] K. Mershad, "MQL: Mixed Query Language for Querying MySQL and HBase databases," *International Conference on Innovative Trends in Computer Engineering* (*ITCE*), IEEE Press, pp. 124-129, Feb. 2019.

[22] B. Liu, Y. Zhu, C. Wang, Y. Chen, T. Huang, W. Shi, et al, "A versatile event-driven data model in HBase database for multi-source data of power grid," *IEEE International Conference on Smart Cloud* (*SmartCloud*), IEEE Press, pp. 208-213, Nov. 2016.

[23] M. Stonebraker, "Stonebraker on NoSQL and enterprises," *Communications of the ACM*, vol. 54, no. 8, pp. 10-11, 2011.

[24] S. Lynden, Y. Tanimura, I. Kojima, A. Matono, "Dynamic data redistribution for MapReduce joins," *IEEE Third International Conference on Cloud Computing Technology and Science*, IEEE Press, pp. 717-723, Nov. 2011.

[25] Apache HBase Team, *Apache HBase™ Reference Guide*. HBase.apache.org/book.html#number.of.cfs

[26] C. Cao, W. Wang, Y. Zhang, X. Ma, "Leveraging column family to improve multidimensional query performance in HBase," *IEEE 10th International Conference on Cloud Computing* (*CLOUD*), IEEE Press, pp. 106-113, June 2017.

[27] J. C. Hsu, C. H. Hsu, C. S. Chen, Y. C. Chung, "Correlation aware technique for SQL to NoSQL transformation," *7th International Conference on Ubi-Media Computing and Workshops*, IEEE Press, pp. 43-46, July 2014.

[28] R. Ouanouki, A. April, A. Abran, A. Gomez, J. M. Desharnais, "Toward building RDB to HBase conversion rules," *Journal of Big Data*, vol. 4, no. 1, pp. 1-21, 2017.

[29] N. Ahmed, S. Ahamed, J. L. Rafiq, S. Rahim, "Data processing in Hive vs. SQL server: A comparative analysis in the query performance," *IEEE 3rd International Conference on Engineering Technologies and Social Sciences* (*ICETSS*), IEEE Press, pp. 1-5, Aug. 2017.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.