JHive: Tool for Integrating Hive for Performance Analysis

Dipti Shikha Singh[1], Garima Singh[2]

[1] *Student, M.Tech, Computer Science Department, Babu Banarasi Das University, Lucknow, U.P, India*
[2] *Assistant Professor, M.Tech, Computer Science Department, Babu Banarasi Das University, Lucknow, U.P, India*

---***---

Abstract - *Social, personal or professional use of internet gives rise to Big Data with an incredible speed. The Big data analysis has emerged as an important activity, there is still a debate about the tools and management frameworks that work on top of MapReduce. This document sheds light on many of these documents that help us with the idea of translators that help SQL-to-MapReduce translations for managing Big Data. Also, we discuss the right approach to get valuable information from large data stack using Hive. Although HiveQL provides similar features as SQL, complex SQL queries are difficult to map as HiveQL while they often results in longer execution time. A tool JHive is designed to solve this problem using query rewrite based MapReduce that, while the correction is saved, improves performance with respect to execution time.*

Key Words: **Big Data, Hive, SQL-to-MapReduce translators, TPC-H, Query rewriting, etc.**

1. INTRODUCTION

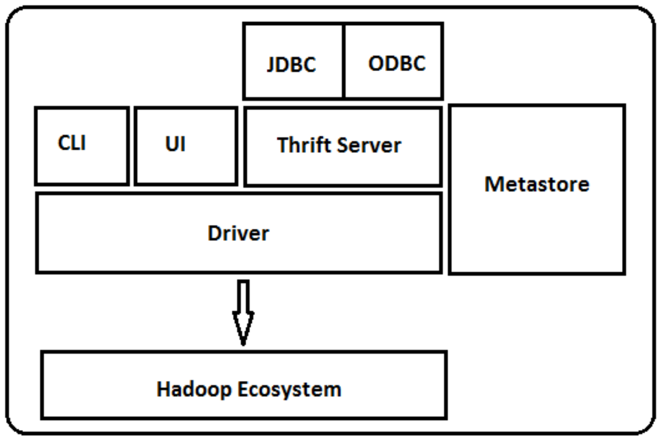
The enormous amount of data from various sources, including companies, health systems, social websites, etc., cannot be processed by traditional databases. This large amount of raw data is known as Big Data. This amount of data includes high-volume, high-speed and range data exponentially increasing and are measured in exabytes (10^{18}) and zettabytes (10^{21}). Therefore, Apache Software Foundation led a framework called Hadoop for Big Data Management and processing challenges to solve.

These features help manage and understand data centers and use this data to extract valuable information. Hadoop is an open source framework that focuses on the processing potential of Big Data in a distributed environment. It contains two modules one of them is MapReduce that is a model of parallel programming which is to process large amounts of structured, semi-structured and unstructured data in large groups of standard hardware, while another is Hadoop Distributed File System (HDFS), which forms part of the framework used in Hadoop to store and to generate sets of processed data. Ecosystem contains various Hadoop sub projects as Scope, Pig and Hive to help.

Hive is a tool for data warehousing, infrastructure building, data querying and analysis. Hive provides an SQL-like interface to query data stored in multiple databases and file systems that integrate with Hadoop. The traditional SQL queries must be implemented in the Java MapReduce API to perform SQL queries over distributed data and applications. Hive provides the abstraction necessary to integrate SQL (HiveQL) queries to the underlying Java API without the need to implement queries in the Java low-level API. Since most of the work application uses data warehousing query language based on SQL, Hive supports simple portability from SQL to Hadoop-based application. Although originally developed by Facebook, now Apache Hive is used and further developed by other companies such as Netflix, Financial Industry Regulatory Authority and Amazon Web Services.

Hive supports the analysis of large amounts of data stored in Hadoop HDFS and supported file systems such as Amazon S3 file system. SQL language represents a type HiveQL with read schema and queries translucently converting MapReduce Jobs, Apache Tez and sparks. To speed up the queries, it provides indexes, including bitmap indexes. The properties are:
• Stops the schema in a database and processes the data in HDFS.
• It is designed for OLAP.
• Provides SQL-like language for HiveQL or HQL query.
• It is known to be fast, scalable and extensible.
• Different types of storage memories, such as plaintext, rcfile, HBase, and others.
• Save metadata in an RDBMS to significantly reduce the time required during query execution to perform semantic audits.
• Operation with compressed data stored in the Hadoop ecosystem.
• Manipulate user defined (UDF) functions for data, strings, and other data mining tools.

Major components of the Hive architecture are:



Metastore: Stores metadata for each of the tables, such as the control and the location. It also includes metadata partition, which will help drivers to follow the progress in multiple sets of data distributed across the cluster. The data is stored in a traditional RDBMS format. Metadata helps the driver maintain a data track and is very crucial. Therefore, a server backup regularly replicates data that can be lost in case of data loss.
Driver: It acts as a controller that receives HiveQL records. Initiated conducting the assessment meetings and creating lifecycle monitoring and implementation progress. Stores the required metadata generated during the execution of a set HiveQL. The controller also acts as a collection point or to obtain a data query result after the operation.
Compiler: Performs query compilation HiveQL, which converts the query into an execution plan. This plan includes the tasks and steps required by which Hadoop MapReduce are performed to get the output as translated by the query. The compiler converts the query to abstract syntax tree (AST). The compatibility and the compiler errors after checking, converts AST to a directed acyclic graph (DAG). DAG share operator levels and MapReduce tasks based on input query and data.
Optimizer: Make several changes to the implementation plan for optimized DAG. Various changes can be added together as the conversion of a pipeline of joins to a single-junction, for better performance. You can also share tasks as a transformation to data before a reduction operation application to provide better performance and scalability. However, the transformation logic used for optimization may be used, modified or channeled using another optimizer.
Executor: After compiling and optimizing, Executor performs the tasks according to the DAG. Interact with the Tracker of Hadoop to schedule tasks to run. It is responsible for the tasks of channeling to ensure a dependent task is executed only when all prerequisites are executed.
CLI, UI and Thrift Server: Command line interface and UI (User Interface) allow an external user to interact with Hive by submitting requests, instructions, and status monitoring process. The Thrift server allows customers to interact with external Hive in the same way that the JDBC / ODBC server.

These languages and translators have significantly improved the productivity of writing MapReduce programs. However, in practice, it is observed that self-generated MapReduce programs for many queries are often extremely inefficient compared to hand-optimized programs for experienced programmers. These SQL-to-MapReduce inefficient translations bring two critical problems. First, the MapReduce Jobs run unacceptably long in the production environment. Secondly, for a cluster of large production programs generated by SQL-to-MapReduce translations would create a lot of unnecessary work that is a serious waste of resources in the cluster. This motivates us to investigate bottlenecks in translators as hive and to develop highly-optimized MapReduce programs for complex SQL queries to produce a more efficient SQL translator.

2. LITERATURE SURVEY

Exploring my topic of managing big data with Hive various steps for processing large data and problems centered in the administration. Articles talk about how to achieve an impressive amount of data developed using technology to get new insights. These documents deal with the challenges created by Big Data and explore the features of the database. Hive Hadoop is a distributed system based on open source SQL-like problems to deal with, by providing a similar SQL on top of Hadoop framework abstraction.

Tansel Dokeroglu, Serkan Ozal, Murat Ali Bayir, Muhammet Serkan Cinar and Ahmet Cosar in Improving the performance of Hadoop Hive by sharing scan and computation tasks [4] discusses the optimization framework for multiple queries, SharedHive to improve the overall performance of Hadoop Hive, using MapReduce. In order to improve the performance of Hadoop Hive environments it is suggested that SharedHive queries as batch processes and improves the overall run time before it merges the optimized Hive queries. SharedHive converts a set of correlated HiveQL conversions to a new set of insert queries within a shorter implementation time. To benefit from common scan/ join tasks of input queries and reduce the number (i.e, the total amount) of redundant tasks SharedHive melds queries into a new set of insert queries and generates each query as a stand-alone HDFS file. This approach has shown experimentally that you can achieve significant performance improvements by reducing the number of MapReduce tasks and total file read / write data.

Fawzy Ramadan Sayed, Mohamed Khafagy in QRMapper: Optimized Tool for Advanced SQL Mapper on Hive [6] discusses advanced system to introduce MapReduce SQL translator named QRMapper. The QRMapper system has five main stages; SQL Query parser; fetch Sub Query, Sub Query Optimizer, execute Sub Query, and Final Query transformation. The system was implemented with Sub Query Optimizer, Sub Query transformation by Query Rewriting. Then the final query transform applied to the input query result after the secondary query returns. The QRMapper system can perform complex queries with UNION, INTERSECT, MINUS, sub-query in HAVING, sub-query clause in the WHERE clause. This document verifies the correctness of the proposed system by using several experiments to perform various TPC-H queries.

Alireza Khoshkbarforoushha, Rajiv Ranjan in Resource and Performance Distribution Prediction for Large Scale Analytics Queries [7] describes the use of Mixture density networks (MDN) for CPU and prediction run-time distribution Large-scale analytic queries with Hive queries. Recent studies have investigated the efficacy estimates based on income distribution of workload versus the prediction of single point for a set of management problems workload as scheduling consultations, access control, etc.; Where one simply assume that the probability distribution function (pdf) of the target value is now available. This article aims to address this problem for an inseparable part of the large workload data analysis, Hive queries. In this work, they combine knowledge of execution of hive queries and MDN to predict the spectrum resource and performance as probability density techniques using TPC-H, which show that not only accurate predictions of pdf are produced but only in half of the experiments exceeds the prior state of art single point technique.

Wu-Chun Chung, Hung-Pin Lin Shih-Chang Chen, Mon-Fong Jiang, Yeh-Ching Chung in "JackHare: a framework for SQL to NoSQL translation using MapReduce" [2] discuss a translator of consultations with the Hadoop Distributed File System. However, it may be difficult to update the data frequently in this file system. Therefore, there is a need for a flexible data storage such as HBase not only to place the data on a scalable memory, but also to manipulate the variable data transparently. However, the HBase interface is not friendly enough for most users. A GUI made by the client application and SQL database connection to HBase facilitates a learning curve. In this article, there is a framework JackHare with SQL query compiler, JDBC driver, and a systematic method to use the MapReduce framework for processing unstructured data in HBase. After importing the JDBC driver on a GUI SQL client, we can use HBase as the underlying data store to run the ANSI SQL queries. Experimental results show that this approach can work well with scalability and efficiency.

Junbo Zhang, Dong Xiang, Tianrui Li and Yi Pan in "M2M: A Simple Matlab-to-MapReduce Translator for Cloud Computing" [3] is a translator is a possible solution to traditional programmers to easily implement an application in the cloud systems by translating sequential codes to MapReduce code. Recently, some translators of SQL-to-MapReduce dive into SQL-like queries that translate codes for MapReduce and have good performance in cloud systems. MATLAB is a high-level and interactive environment for numerical calculation, visualization and programming, which is very popular in technology. Proposed and developed a simple translator Matlab-to-MapReduce for cloud computing, called M2M, for basic numerical calculations. M2M can translate a Matlab code in seconds using up to 100 commands to the MapReduce code. In addition, M2M can also detect the dependency between complex commands, which is always confusing for manual coding. They implemented evaluation M2M with Matlab in a cluster commands. Several common commands are used in this experiment. The results show that M2M is comparable in performance to manually encoded programs.

Fawzya Ramadan Sayed and Mohamed Helmy Khafagy in this paper "SQL TO Flink Translator" [5], a SQL Flink translator is proposed as a new system to define and add SQL Query language to Flink, this translation improves SQL without modification within the framework and offers the possibility to executing SQL Query on Flink by generating Flink algorithm. SQL TO Flink translator has the ability to run SQL query when other systems support underperforming queries and also has the best performance when tested on TPC benchmark.

In the paper [2], JackHare proposed to process a comprehensive solution including compiler SQL query, the JDBC driver and a systematic method MapReduce using unstructured database NoSQL data. JackHare was developed based on Hadoop and HBase to store data that originally resided in the relational database and developed the corresponding MapReduce methods based on the logic of the SQL queries. In the paper [3], a simple method to translate Matlab code to MapReduce code and develop a simple translator called Matlab-to-MapReduce M2M. Experiments show that M2M is comparable to the performance of manually encoded programs. M2M provides not only data, but also task parallelism. In addition, M2M programmers can help to easily implement Matlab system applications in the cloud without programming with MapReduce and Hadoop programmers help to significantly reduce the programming time. The above document [4] presents the architecture of SharedHive, which is using a new MQO (Multiple Query Optimization), which is in the top of the driver component Hive a modified version of Hadoop Hive. Attempts have shown that you can achieve significant performance improvements by reducing the number of MapReduce tasks and the overall size of the read / write files. The document [6] addresses the rewriting of queries and optimization QRMapper instead of QMapper. It also shows comparisons among other applications translators working in large amounts of data and in any case there is a marked increase in efficiency in all experimental results. In [7] a set of black-box models designed to predict the distribution of CPU and runtime workloads Hive query. The models are based on a set of specific functions of SQL and MapReduce and statistics trained in the execution plan with HiveQL as data input. The approach is evaluated at the reference point support decision of TPC-H technique, which indicates that accurate PDF prediction approximates predictive distribution using appropriate error metrics. In the paper [5] SQL To flink translator SQL is a query language, which is based on the analysis of large-scale data sets a tool built on Apache Flink is to support SQL queries. Users send a SQL query to Flink translator to provide the appropriate code to these queries, which can run on Flink. This translator provides a high degree of flexibility to work with no intervention in Flink algorithm that improves the performance of SQL Query Language in large data.

3. PROBLEM STATEMENT

As working on Big Data in administration, the Hive approach is used as an interface to analyze and manage tables stored in HDFS. The focus is on finding Hive commands, syntax, and semantics in managing Big Data that is usually large. In addition, many of these Big Data solutions include products that are relatively new and are still developing rapidly. These products have not matured to a point where they are used in a variety of applications and are far from being fully tested. Therefore, hive which I'm trying to explore experimentally shows that you can achieve the reduction of the number of tasks, MapReduce, and overall files read / written in terms of the problems achieving significant performance improvements and trying to provide a solution to the challenges offered during my research.

One disadvantage of the MapReduce model is that users have to convert their work into refined maps and reduce code. Apache Hive solves this problem by using it as SQL-to-Hadoop MapReduce for translators. There are some published documents to analyze different techniques and frameworks addressing data and to improve the performance of Hadoop Hive. But my task requires optimization for the transformation of the Hive queries. To improve the performance of Hadoop Hive spent in the issued query, I suggest JHive, which will process the HiveQL queries as a batch and improve the overall run time of correlated join queries before the optimizer passes Hive queries. The developed model is presented as a new component architecture for Hadoop Hive. Similar work has been done in [4] by incorporating a number of MapReduce tasks and overall sizes of read / write records of several optimization issues (MQO) for performance enhancements.

In recent years, a significant amount of research and commercial activity has focused on the integration of MapReduce technologies and structured databases. Mainly there are two approaches, either MapReduce adding functions to a database or adding parallel database technology to MapReduce. [1], [2], [3], [4], [5] and [6] work in the second option, while in [7] the first option works.

My research has similar approach [1] [2] [3] [4] [5] [6] to integrate the framework Hive with using JH optimizer and work results HiveQL queries somehow improve the performance and offers expected results by SQL Surveys and comparison with conventional products, and likewise.

4. PROPOSED ARCHITECTURE

JHive architecture, which is a modified version of Hadoop Hive with a new component, JH Optimizer is used in the top of the driver component Hive (see Fig. 2). The inputs to the controller contains, compilers, optimizers, and executors are pre-processed by the aggregate component JH Optimizer analyzing incoming requests and generating a set of HiveQL merged queries. System catalog and the structure of the relational database (relationships, attributes, partitions, etc.) stored and managed by Metastore. Once a HiveQL command is sent, it is retained by the controller, which controls the execution of tasks to answer the request. Compiler analyzes the query string and converts the parsing tree into a logical plan. The optimizer performs multiple passes over the logical plan and rewrites. The physical plan generator creates a physical plan from the logical plan.

The HiveQL sets are sent through the command line interface (CLI), the user interface, or an interface savings. Typically, the query is directed to the driver component in traditional Hive architecture. In JHive, the JH optimization component (following the client interface) receives incoming requests before the driver component. The amount of incoming requests will be examined, their common intermediate tables and common joins are detected, and come together to get a new set of HiveQL queries to answer all incoming queries.

The new JH Optimizer component passes the new set of insert queries to the compiler component of Hive driver that produces a logical plan using information from the Metastore and optimizes this plan using a single rule based optimizer. The runtime receives a directed acyclic graph (DAG) associated with MapReduce and HDFS tasks, and then runs correspondingly to the dependencies of tasks. The new component JH Optimizer does not require a significant change in the system architecture of Hadoop Hive and can easily be integrated into Hive.

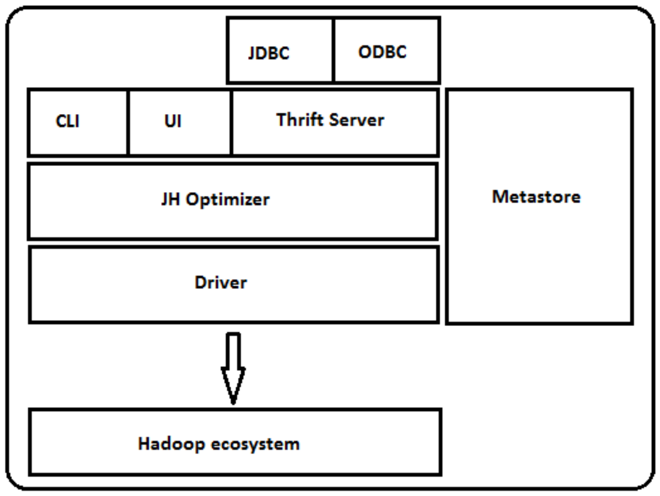


Fig.2 Components of the JHive architecture

4.1 METHODOLOGY

- Phase 1: Query is given by user, rewritten and received from proposed system drivers
- Phase 2: Query is optimized by the optimizer and equivalent query is produced that returns the same result in shorter time
- Phase 3: Query is executed.

The TPC-H queries are described with the SQL language. Hive provides a similar query language called HiveQL. It does not support all features in SQL yet. However, most TPC-H queries can be rewritten in HiveQL without changing the semantics. In particular, some of them require small modifications (e.g. selected from multiple tables are rewritten using joins); some of them need moderate changes (e.g. sub-queries are rewritten to individual queries) and the remaining require relative large changes (e.g. UDFs rewritten with individual queries). In this section, we will explain how TPC-H queries are rewritten in Hive QL. We will go through a TPC-H query as example and describe the rewritten queries in Hive QL.

4.1.1 Shipping priority query (Q3):

Hive QL does not support selecting from multiple tables. So in this query, selecting from multiple tables is rewritten to joins and "where" clauses become "on" clauses in the joins. This change is very common across all the queries. The original **TPC-H SQL query is:**

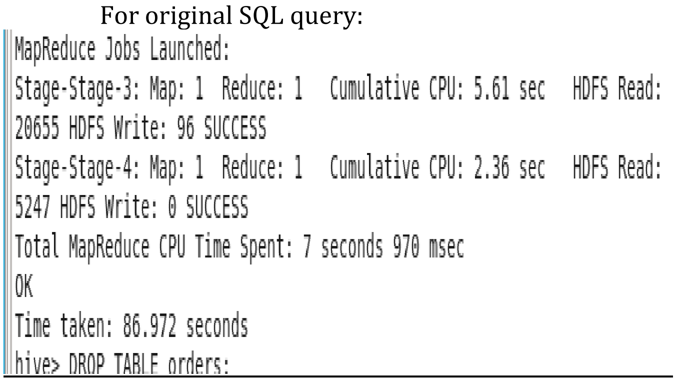
```
select ...
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = 'BUILDING'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < '1995-03-22'
        and l_shipdate > '1995-03-22'
group by
        l_orderkey,o_orderdate,o_shippriority
order by
        revenue desc,o_orderdate
limit 10;
```

The rewritten Hive query looks like:

```
Insert overwrite table q3_shipping_priority
select ...
from
        customer c join orders o
        on c.c_mktsegment = 'BUILDING'
        and c.c_custkey = o.o_custkey
 join
        lineitem l on l.l_orderkey = o.o_orderkey
where
        o_orderdate < '1995-03-15'
        and l_shipdate > '1995-03-15'
group by
        l_orderkey, o_orderdate, o_shippriority
order by
        revenue desc, o_orderdate
limit 10;
```

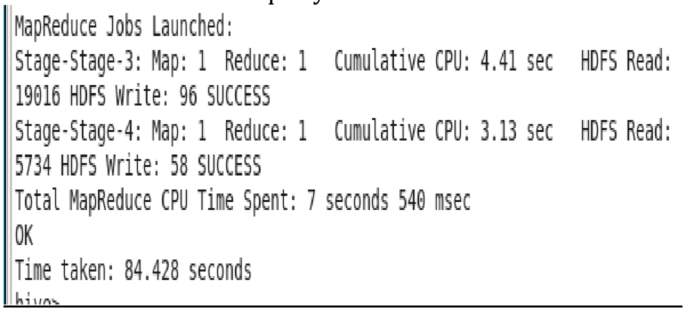
4.1.2 Query Run Snapshot:

For original SQL query:



```
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1  Reduce: 1   Cumulative CPU: 5.61 sec   HDFS Read:
20655 HDFS Write: 96 SUCCESS
Stage-Stage-4: Map: 1  Reduce: 1   Cumulative CPU: 2.36 sec   HDFS Read:
5247 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 970 msec
OK
Time taken: 86.972 seconds
hive> DROP TABLE orders;
```

For rewritten query:



```
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1  Reduce: 1   Cumulative CPU: 4.41 sec   HDFS Read:
19016 HDFS Write: 96 SUCCESS
Stage-Stage-4: Map: 1  Reduce: 1   Cumulative CPU: 3.13 sec   HDFS Read:
5734 HDFS Write: 58 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 540 msec
OK
Time taken: 84.428 seconds
hive>
```

5. COMPARISIONS

In comparison between the features of Hive, JHive and RDBMS few examples of key features that differ from RDBMS.Table 1 shows a comparison of the features of the data management systems with proposed JHive.

FEATURE	RELATIONAL DATABASE/SQL	HIVE/HIVEQL	JHIVE/ HIVEQL
Data loading	takes longer to load data	very fast initial load	very fast initial load
Query execution	query time performance faster	query time performance comparatively slower	query time performance comparatively faster
Multi-table inserts	Not supported	Supported	Supported
Supported paradigms	OLTP	Large scale analysis (Large scale OLAP)	Large scale analysis (Large scale OLAP)
Maximum data size allowed	10's of Terabytes	100's Petabytes very easily	100's Petabytes very easily
Scalability	not that much scalable that too it is very costly scale up	easily scalable at low cost	easily scalable at low cost

Table2. Comparison between Hive, JHive and a relational database

In comparison between the similar TPC-H database feature of SharedHive where they use the same database and benchmark yet differ in the query sets being used while experimentation.Table 3 shows a comparison of the execution time reduced with comparision to Hive of the translators proposed.

YEAR	TRANS LATOR	ALGORITHM	PARAMETERS
2014	Shared Hive	Map-Reduce algorithm and Merged Query algorithm using correlated queries on Hive	Execution time reduced by: 13.2% (on average)
2017	JHive	Map-Reduce algorithm and Query Rewriting using Hive	Execution time reduced by: 17.62% (on average)

Table3. Comparison between SharedHive and QRMapper

In comparison between the SQL semantics of Hive, QRMapper and SharedHive where they use the different query sets while experimentation.Table 4 shows a comparison of the features of the translators proposed in different papers.

SQL SEMANTICS	HIVE	SHIVE	JHIVE
SELECT,INSERT and LOAD from query	T	T	T
GROUP BY, ORDER BY	T	T	T
Sub query in WHERE clause	F	T	T
LEFT,RIGHT,FULL,CROSS JOIN	T	T	T

Table4. Comparison for acceptable SQL semantics between translators.

6. EXPERIMENTAL EVALUATION

6.1 DATASETS AND QUERIES:

We use dataset and queries from TPC-H Benchmark. This benchmark illustrates decision support systems that provides large volumes of data, execute complex queries and give answers to critical business questions [10]. Dataset is split to a different size for running TPC-H queries on this dataset.

6.2 ENVIRONMENT SETUP:

We perform the experiments on Linux 6 64-bit OS on workstation 8.x virtual machine. RAM is 4GB and 64GB disk space for one Master node and Worker node. Hive 0.12.0 and Hadoop 2.6.0 are installed and 2GB TPCH datasets are generated as workload.

Queries from TPC-H are used to explore different semantics that can be used for mathematical calculations. These complex queries can help us explore different aspects of any database and calculate the performance for different software applications. Here using Hive we have shown a performance improvement in it by reducing the execution time of such complex queries up to 17.6% on average for the corresponding 22 SQL queries. Here we have used query re-writing using insert queries and this technique shows reduction in all queries.

6.3 EXECUTION TIME AND REDUCTION %

Query	Select Query	Rewritten query	Reduction%
Q1	11.06s	9.45s	14.56%
Q2	12.23s	7.1s	41.95%
Q3	7.97s	7.54s	5.4%
Q4	7.28s	7.19s	1.24%
Q5	9.3s	8.76s	5.81%
Q6	4.28s	3.89s	9.11%
Q7	10.06s	8.58s	14.71%
Q8	14.8s	13.89s	6.13%
Q9	13.2s	7.58s	42.58%
Q10	7.61s	7.51s	1.31%
Q11	16.5s	8.46s	48.73%
Q12	10.38s	9.76s	5.97%
Q13	10.34s	9.82s	5.12%
Q14	7.05s	6.13s	13.05%
Q15	21.3s	13.05s	38.73%
Q16	17.26s	14.78s	14.37%
Q17	13.6s	10.41s	23.46%
Q18	16.82s	10.88s	35.32%
Q19	5.2s	4.57s	12.12%
Q20	25.71s	21.18s	17.62%
Q21	23.63s	18.77s	20.57%
Q22	20.18s	18.27s	9.46%

Table5. TPCH Queries Performance Reduction%

6.4 OVERALL PERFORMANCE

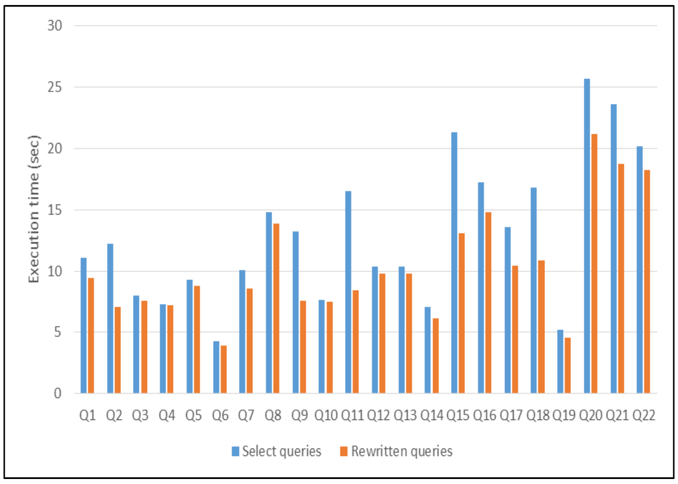


Fig 3. Performance of full TPCH Queries

On the X-axis we have comparision between 22 TPCH select and rewritten quries respectively. While on Y-axis we have execution time in seconds that clearly shows the reduction in execution time in rewritten queries visible in orange colour bars comparative to select queries visible in blue colour bars. The reduction is achived to be 17.62% as on average execution time.

7. CONCLUSION

In the future, we will try to explore more possible optimization techniques to further improve the performance. Thus, we can plan to work on different optimization models and tools for the Hadoop MapReduce execution framework using Hive. We can also work on cost-aware models and approach for optimizing the MapReduce job scheduler in terms of workloads for different types of applications. Finally, we plan to integrate all these new optimizations with the optimizations proposed in this paper to achieve more performance improvement.

REFERENCES

[1] Tim Kaldewey, Eugene J. Shekita, Sandeep Tata, "Clydesdale: Structured Data Processing on MapReduce", Proceedings of 15th International Conference on Extenting Database Technology Pages 15-25, Berlin, Germany-March 27-30, 2012, ISBN: 978-1-4503-0790-1.

[2] Wu-Chun Chung, Hung-Pin Lin Shih-Chang Chen, Mon-Fong Jiang, Yeh-Ching Chung, "JackHare: A framework for SQL to NoSQL translation using MapReduce" Autom Softw Eng DOI 10.1007/s10515-013-0135-x Springer Science+Business Media New York 2013.

[3]Junbo Zhang, Dong Xiang, Tianrui Li and Yi Pan et al. "M2M: A Simple Matlab-to-MapReduce Translator for Cloud Computing" Tsinghua science and technology, pp 1- 9 Volume 18, Number 1, February 2013, ISSN 1007-0214 01/12.

[4]Tansel Dokeroglu, Serkan Ozal, Murat Ali Bayir, Muhammet Serkan Cinar and Ahmet Cosar "Improving the performance of Hadoop Hive by sharing scan and computation tasks" Journal of Cloud Computing: Advances, Systems and Applications 2014, DOI: 10.1186/s13677-014-0012-6,licensee Springer 2014, Published: 29 July 2014.

[5] Fawzya Ramadan Sayed and Mohamed Helmy Khafagy, "SQL TO Flink Translator", IJCSI International Journal of Computer Science Issues, Volume 12, Issue 1, No 1, January 2015, ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784.

[6] Fawzy Ramadan Sayed, Mohamed Khafagy "QRMapper: Optimized Tool for Advanced SQL Mapper on Hive" International Journal of Computer Science and Information Security (IJCSIS), Vol. 14, No. 5, May 2016, ISSN 1947-5500.

[7] Alireza Khoshkbarforoushha, Rajiv Ranjan, "Resource and Performance Distribution Prediction for Large Scale Analytics Queries"ACM New York, NY, USA, published on 12-3-2016, ISBN: 978-1-4503-4080-9.

[8] Dipti Shikha Singh, Garima Singh, "Big Data- a Review" International Research Journal of Engineering and Technology, Volume: 04 Issue: 04 | Apr -2017, ISSN (online): 2395 -0056.

[9] Edward Capriolo, Dean Wampler, and Jason Rutherglen, "Programming Hive", Published by O'Reilly Media, Inc., 2012-09-17, ISBN: 978-1-449-31933-5.

[10] Tom White, "Hadoop: The Definitive Guide", Published by O'Reilly, 2012-01-27, ISBN: 978-1-449-31152-0.

[11] Hadoop Web Page:hadoop.apache.org/

[12] TPC-H benchmark specification, URL: http://www.tpc.org/