JOUM: An Indexing Methodology for Improving Join in Hive Star schema

Hussien SH. Abdel Azez,Mohamed H. Khafagy, Fatma A. Omara

Abstract— Now a day, Big data represents an important and complex Issue for information extraction/retrieval due to required analysis computation power. Also, Database Star schema is considered one of the complicated data models due to the using of Join queries to extract information and generate requested reports. These Join queries need to scan a vast amount of data (tera, peta, zeta bytes). On the other hand, HIVE is one of the important and efficient Big data SQL querying tools built on the top of Hadoop to translate SQL queries into Map/Reduce tasks. By using indexing data for Join queries could speed up HIVE Join query (map/reduce) tasks especially in Star Schema. According to the work in this paper, JOUM (Join once Use Many) methodology has been introduced to pre-join the star schema data and build an index for Joined data. Based on JOUM, SQL queries execution time in HIVE has been improved without changing HIVE framework. TPC-H benchmark has been used to evaluate the performance of JOUM methodology. The experimental result proves that JOUM methodology outperforms traditional Join execution time. Also, JOUM performance is improved by increasing data size. Generally, JOUM can be considered one of the suitable methodologies for Big data analysis.

Index Terms— Big data, disturbed processing, map/reduce, Hadoop, Hive, start schema, Join query, Database.

1. Introduction

New applications such as machine learning, web searches, recommendation engines, and social networks generate enormous amounts of logs, email, and other technical structured/unstructured information streams. Such applications need fast processing of data which is involved in today's business processes analysis. These applications might contain several thousand tables with over hundreds of terabytes of data which are used heavily for both reporting and decision-making which often don't need update or deletion operations [1].

Map Reduce is a programming model for large-scale distributed data processing with simple and elegant concepts which are used to build blocks for other parallel programming tools. In the same time, it is considered extensible for different applications providing advantages of Concurrency/Parallelism, tolerating failures and hiding any complexity from the user. So, Map Reduce has become the important standard for large-scale data processing in many enterprises. Also, it is used for developing new solutions on massive datasets such as relational data analytics, web analytics, machine learning, real-time analytics and data mining [2, 3].

Hadoop is considered a framework based on Map Reduce programming model for large-scale distributed data processing. According to Hadoop, the applications run on large clusters. These clusters are built from a variety of homogeneous hardware. They provide the applications both reliability and data mobility. Therefore, Hadoop implements Map Reduce concepts Where the application is divided into small tasks, every task could run or re-executed on any cluster's node. Also, Hadoop uses a distributed file system that stores data on the compute nodes as a tree of distributed blocks, and provides data privacy and handles software node failures [4, 5].

Many approaches have been introduced to improve the performance of Hadoop. In particular, they have focused on supporting efficient index access in Hadoop. But, most of these indexing approaches have three main weaknesses:

- A high upfront cost required for index creation.
- Only one physical data sort order per dataset.
- The users need to have a high knowledge of data work-loads to choose the perfect index to create.

Therefore, structured database start-schema (Star Join schema) is the simplest style of the data mart schema. This star schema has one or more fact tables referencing large number of dimension tables. The star schema is more efficient for creating a simple query. Also, it represents one of the complicated scheme that requires almost Joining schema tables to gather information for decision makers [6,7].

On the other hands, TPC-H is one of the massively used Star Schema decision support benchmarks. It consists of a set of business-oriented tables and concurrent data modifications. This benchmark simulates decision support systems that use massive data set, execute queries with a high degree of complexity, and provide answers to critical business questions. Also, TPC-H benchmark provides multiple aspects of the capability of the system to process queries. These aspects contain; the selected database size, the query processing complexity when queries are submitted by a single flow, and the query throughput when multiple concurrent users send queries [7].

According to TPC-H benchmark, 19 prepared queries have been involved to measure DB performance. These queries will be used to evaluate the performance of our proposed JOUM methodology relative to the existed systems. TPC-H star schema tables are presented in Figure 1 [8].

- *Hussien SH is currently pursuing masters degree program in computer science in Cairo university, Egypt, PH-01004860780. E-mail: hsa01@Fayoum.edu.eg*

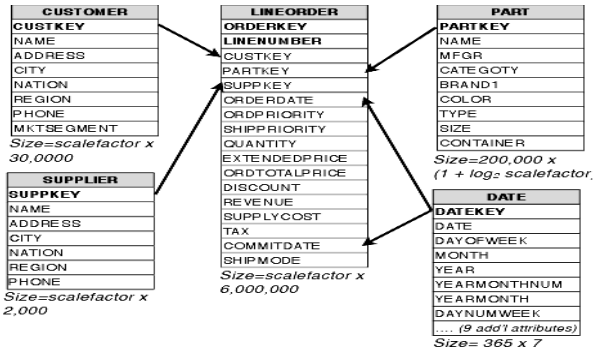

Figure 1:PC-H Star Schema completely [8].

On the other hands, Join operation is considered one of the high-cost operations in database systems which are used to gather information from two or more tables. Also, Join operation represents a special case of Cartesian product that need to be optimized . These Joined tables are checked against specific conditions before concatenating them. These conditions have different forms like equi-Join, self-Join, Outer Join , Inner Join, etc. [9]. All of these Join forms are used to extract data from Join tables with complexity O(n^k) where n represents an average number of records in all tables, and k represents the number of tables in Join operation. This complexity is non-reasonable for computation of significantly sized star schema data models [10].

HIVE is a DB-like software running on top of Hadoop framework to facilitate executing query and managing large datasets which are stored in a distributed storage. HIVE uses a simple SQL-like query language, called HIVE QL that enables users who are familiar with SQL to query HIVE data. Also, HIVE QL allows Map-Reduce framework programmers to be able to write their custom mappers and reducers to process more complex and sophisticated jobs. HIVE QL programmers can write their custom scalar User-Defined Functions (UDF's) which are more like database scripts or functions that can query database schemas, write UDAF (User-Defined Aggregations Function) for custom aggregation operations, and write UDTF(User Defined Table Functions) for online tables creation [11].

HIVE/Hadoop architecture is shown in Figure 2 explains how HIVE translating SQL query given by JDBC or ODBC connection to HIVE thrift server or given through CLI or Web Interface into Hadoop Map-Reduce job. Translation steps (i.e., compiler - Optimizer - Executor) translate SQL Query into Set of Map-Reduce tasks, optimize them, and then execute it over Hadoop cluster [12,13].

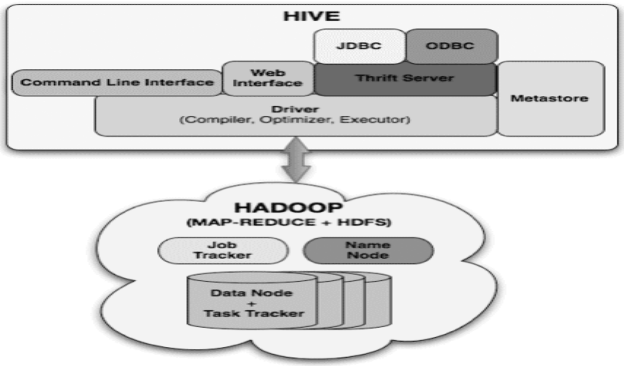

Figure 2: Hive/Hadoop Architecture Framework

Also, Join Operation is used to match the rows of two or more tables. By performing Join operation, it will produce all rows from all tables related to some specific fields or properties. In order to, understand what is happening internally by performing the Join operation using HIVE. it needs to imagine this operation like a Map Reduce task. So, mapper will read the data from Join Tables then return the Join key and Join value pair into an intermediate file. This intermediate file will be sorted and merged in the shuffle stage. Now, reducer takes this sorted result as input and completes the task of Join. But, the shuffle step is expensive since it needs to sort and merge all records. Therefore, the shuffle operation steps need to save which will improve the performance, reduce the total storage required to complete the task [14].

Since, HIVE Join operation translates Join query into Map Reduce task that visits physical files and selects all data required by query (e.g., TPC-H dataset Parts table and their customers, suppliers, Date of Pay, Line of production). Joining the date of these tables is considered a complicated task which needs high memory, and computation power resources.
This paper is organized as;Section 2 discussesthe related workand their pros/cons. Section 3 discusses the proposed methodology (JOUM). The comparative study of HIVE and JOUM methodology is explained in section 4.Section 5 discusses system setup, data set, cluster machines, Hadoop and HIVE configuration. Also, the performance evaluation is presented in section 5.Conclusions and future work are presented in Section 6.

2. RELATED WORK

Several approaches have been introduced to improve Hadoop Map-Reduce tasks performance. Some of them depend on good knowledge of data structures and their relations (data schema), while other introduce general solutions for Hadoop tasks execution pipeline and data scanning pipeline.

2.1. HadoopDB

Two schools of thought are used for data analysis in Big Data environment. Consultants of parallel databases claimed that the performance and efficiency of parallel databases did more suitable to perform such analysis. Others said that Map Reduce based systems are more suited because of their scalability, fault tolerance, and flexibility to handle unstructured data. By considering both technologies, the performance could be improved. But, it is still suffered from full block scanning and indexing overhead in most cases [15].

2.2. Hadoop++

Hadoop++ is based on enhancing task performance without changing the Hadoop structure by injecting Trojan Index at the right places through the user defined functions only, but it affects Hadoop from inside. Figure 3 presents presenting Hadoop ++ Trojan index.

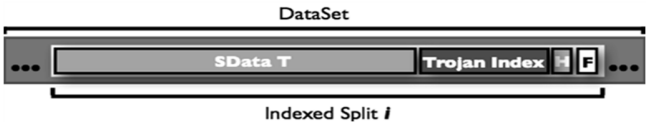

Figure 3: Hadoop ++ Trojan Index injection

Hadoop++ has three important consequences:

- Hadoop++ performs better than Hadoop.
- Future changes of Hadoop will directly use Hadoop++ without rewriting any new code.
- No need to change the Hadoop interface.

Hadoop++ is considered more suitable for tasks related to indexing and Join processing than that Hadoop and HadoopDB. However, it still suffers from non-suitable workloads because there is only one Indexed Attribute, so queries which not related to that attribute will turn it back into Hadoop for execution [16].

2.3. HAIL (Hadoop Aggressive Indexing Library)

HAIL changes data uploading pipeline of HDFS to create different clustered indexes for every data block replica, which improves run times of several classes of queries by choosing the most suitable index/replica to run Map-Reduce jobs [17]. HAIL creates a win-win situation by improving both uploading data to HDFS and the runtime of the actual Hadoop Map-Reduce job. HAIL increases data uploading over HDFS up to 60% with the default replication factor of three. HAIL demonstrates that runs up to 68x faster than Hadoop and even out-performs Hadoop++ as a result. So, the users can speed up their Map Reduce jobs by almost two orders of magnitude. However, this improvement only happens if the users create the most suitable indexes when uploading their datasets to HDFS. This means that HAIL requires users to decide before uploading which indexes suitable to create. So, HAIL is not suitable

for unpredictable/dynamic workloads, as well as, traditional indexing techniques for users who often do not know which indexes to be created beforehand [18].

2.4. LIAH (Lazy Indexing and Adaptively in Hadoop)

Based on LIAH, building adaptive indexes at minimum costs for Map Reduce systems, automatically and incrementally adapt users' workloads by creating clustered indexes on HDFS data blocks. Besides, distributing indexing costs over multiple nodes. All these operations are done without any extra data copies in the main memory and with minimal synchronization. Also, LIAH Postpones index creation on map tasks that read related data from disk to main memory anyways. So, LIAH doesn't consume any extra read I/O-costs and also has a very small indexing overhead, usually for the first job. LIAH can speedily converge to a complete index where all HDFS data blocks are indexed with low overhead of 11% than HAIL for the first Map/Reduce job only. However, LIAH is considered better than Hadoop and HAIL by factor of 52 and 24 respectively. The main drawbacks of LIAH are; the users must have good knowledge about their data and choose index fields carefully. It has. Index overhead because of cluster nodes synchronization for online/dynamic index building which consumes I/O and network bandwidth. For every execution time, the same job/query is needed to rebuild the index at run time which needs high memory and consumes more computation power [19].

3. The proposed Join Once Use Many (JOUM) Methodology

By executing Join, all related operations suffer from re-joining all data tables. Then, extra computing resources will be consumed (e.g. CPU, I/O, and Network Bandwidth).This is considered one of the main issues because an index is needed to be built for each pipeline execution which will consume the computing resources and time. Therefore, the main principle of our proposed solution is to build joined data table for all-star schema tables to be one table at the data uploading phase. The joined data table contains all schema Fact/Dimension tables which will be uploaded into HDFS. Every Join query will be executed like a normal selection statement using the Joined-Indexed table. Unfortunately, the structure of The Joined table will affect the performance of Join query execution, and the needed storage of this table will be a problem.SQL query parser is needed to parse SQL statement and convert it into an equivalent one that will run over Joined data table and produce the same result. In fact, the Joined data table needs to be re-structured and loaded into HIVE, and then it will be processed using a new parser on HIVE. At this moment, there is no need to modify HIVE or Hadoop. A new parser

will convert SQL Join query into a suitable form for the modified Joined table structure then pass it to the HIVE which will process it as one task of Map Reduce over Hadoop. So, the proposed system won't change HIVE architecture, but some up-front parser is added for HIVE and some index technique will be built inside Joined table by some user defined function.

3.1 The Proposed System Framework

The proposed system framework components are presented in Figure 4. Showing that The proposed system Framework is independent of HIVE because Added components are on top or at back end of HIVE engine. these components interact with hive without need of hive architecture changes as explained below in details.

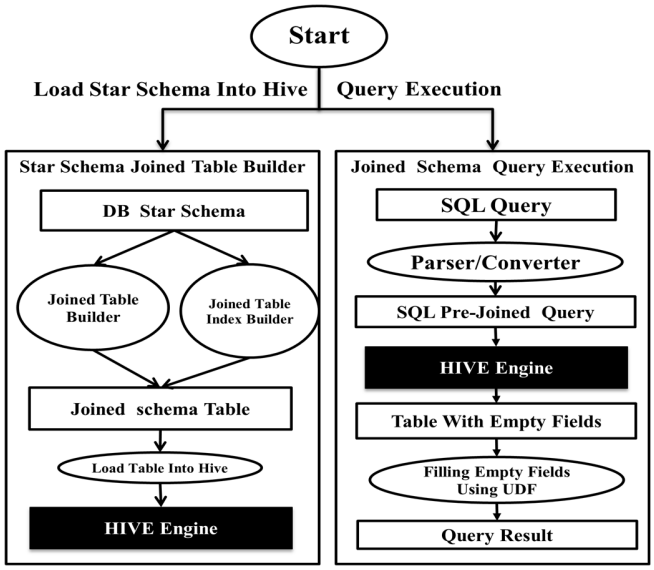

Figure 4: Proposed system framework

3.1.1 Star Schema Joined Table Builder

Joined table schema builder converts star-schema Fact/dimension tables into one table contains all schema data in a Joined form. Figure 5 illustrates TPC-H schema data.

Customer

CUSTKEY	NAME
1	Customer#01
2	Customer#02
-----	---------

PART

PARTKEY	NAME
1	lace spring
2	rosy metallic
-----	---------

SUPPLIER

SUPPKEY	NAME
1	Supplier#01
2	Supplier#02
-----	---------

Date

DATEKEY	DATE
19920101	1-Jan-92
19920102	2-Jan-92
-----	---------

Line Order

ORDERKEY	LINENUMBER	CUSTKEY	PARTKEY	SUPPKEY	DATEKEY
1	1	1	2	2	19920101
2	3	2	1	1	19920102
3	2	2	2	1	19920101
-------	-------	-------	-------	-------	-------

Figure 5: TPC-H Star schema Sample Data

In order to convert schema data into one Joined table Cartesian product of lineorder Table will be constructed from all other dimension tables by substituting each dimension

Line Order

Property Fields		Dimension Fields							
ORDERKEY	LINENUMBER	CUSTKEY	CUSTNAME	PARTKEY	PARTNAME	SUPPKEY	SUPPNAME	DATEKEY	DATE
1	1	1	Cus#01	2	rosy metallic	2	Supp#02	19920101	1-Jan-92
2	3	2	Cus#02	1	lace spring	1	Supp#01	19920102	2-Jan-92
3	2	2	Cus#02	2	rosy metallic	1	Supp#01	19920101	1-Jan-92
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

Figure 6: Line Order Table Joined data

key in the fact table by dimension record data. Figure 6 illustrates Joined TPC-H table. All dimension tables fields are added to line order Joined table by naming pattern contain dimension table name followed by field name. So, line order table fields will increase the count of non-key fields in all dimension tables (e.g.fields CustName, PartName, SuppName, Date)are added into line order table by considering the first record online order table as a sample which will substitute each dimension field by its value according to its dimension key from dimension table such as table customer key customer#01 have custname Cust#01 and so on (see Figure6).

By generating Joined table redundant data will be introduced in the star schema which produces storage overhead to store this redundant data. In order to solve this problem, the dimension fields of a record that appear for the first time will be stored, then new repeated records will be stored as a reference to that record. This reference can be named a pointer/reference/index because it directly points to the required record that contains the fully stored information for this dimension key. By jumping into referenced record by index and substituting empty fields by their original values called Fill-In operation. We can implement this operation by simple User Defined Function (UDF) and plug it into HIVE.

3.1.2 Query Execution (Parser/ Converter/ Executor)

After concatenating Star schema data into one table and loaded into hive. old join query needs to be parsed to define each table/fields within the new query, and convert it into a new equivalent query for new Joined table using the following steps.

1. Convert every field name to tablename_fieldname.
2. Delete all Join tables and replace all of them by select from Joined table name.
3. Use Fill-in UDF function to fill in empty values.

Using these steps, Join Query1 is converted into Query 2 which can run over JOUM schema.

```
select count (*), customer.custkey, customer.name,
part.partkey, part.name, supplier.suppkey, supplier.name,
date.datekey, date.date
from customer, part, supplier, date, lineorder    where
lineorder.custkey=customer.custkey and lineor-
der.partkey=part.partkey and lineor-
der.suppkey=supplier.suppkey and lineor-
der.datekey=date.datekey and date.datekey<1/1/2001
group by customer.custkey, customer.name, part.partkey,
part.name, supplier.suppkey, supplier.name, date.datekey,
date.date
order by customer.name
```

Query 1: Join query from TPC-H tables

```
select count (*), customer_custkey, customer_name,
part_partkey, part_name, supplier_suppkey, supplier.name,
date_datekey, date_date
From joined_table where date_datekey<1/1/2001
group by customer_custkey, customer_name, part_partkey,
part_name, supplier_suppkey, supplier.name, date_datekey,
date_date
order by customer  name
```

Query 2: Joined schema select query

3.1.3 Fill-in UDF:

The redundant data in the new joined table schema needs to be removed, and keep its place as empty fields. Although, these empty fields need to be re-filled during the processing of this schema. So, empty fields can fill in by iterating through previously referenced records from joined table data process. During this iteration these records may be referenced again by next records. So, these referenced records have to cache in memory and be aware of memory size limitations. So, some memory caching algorithms should be used to solve memory overflow like Most Recently Used records (MRU) or Least Recently Used records (LRU) [20] (see Figure 7).

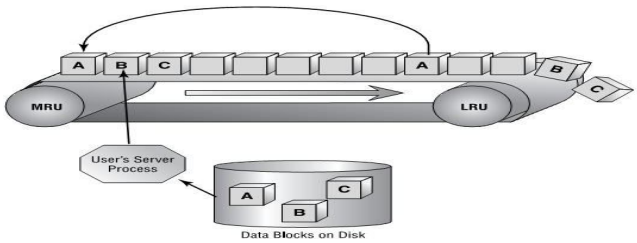

Figure 7: MRU-LRU caching[20].

Using caching to fill in empty fields will be faster, easier and saving memory starvation. Huge number of records as in the case of Big Data, fill in process will be divided into three steps as shown in Figure 8

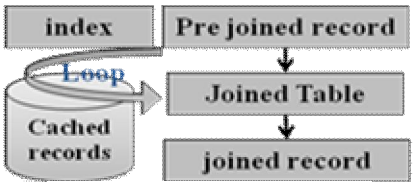

Figure 8: Fill-in function

3.2 Joined Table Structure

Joining star schema data into one table will avoid HIVE to be translated into complex map-reduce task by reducing the number of Map-Reduce tasks which are needed by Hadoop into one Map Reduce task such a select statement from one table. So, this will affect query execution time/performance.

3.2.1 Redundant Joined Table Structure:

All fields from dimension tables will be gathered into one table with all property fields in the Fact Table by iterating through all records, and gathering all related dimensions data into one record. Then, all data will be written by comma separated file format(see Figure 9).

Line Order									
Property Fields		Dimension Fields							
ORD ERKEY	LINENU MBER	CUST KEY	CUST NAME	PART KEY	PART NAME	SUPP KEY	SUPP NAME	DATE KEY	DATE
1	1	1	Cust#01	2	rosy metallic	2	Supp#02	19920101	1-Jan-92
2	3	2	Cust#02	1	lace spring	1	Supp#01	19920102	2-Jan-92
3	2	2	Cust#02	2	rosy metallic	1	Supp#01	19920101	1-Jan-92
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

Figure 9: Line Order Table Redundant Joined data.

Redundant Joined Table data converted into comma separated values (CSV) format is shown in Figure 10 which acceptable format by hive to be loaded as hive table.

```
1, 1, 1,Cust#01,2,rosy metallic,2,Supp#02,19920101,1-jan-92
2,3,2,Cust#02,1,lace sping,1,Supp#01,19920102,2-jan-92
```

Figure 10: Line Order Table Redundant Joined data CSV format.

Loading Data into Hive

Creating a new table called "LineOrderJoined" using redundant Joined table structure, the data will be loaded using HIVE load statement and then rewrite join query to execute on HIVE.

HIVE QL Query parsing and conversion

Parsed query will be re-written as selection from one table without Joins as shown in Query 3.

```
Select custkey, Custname, partkey, partname, suppkey,
suppname, datekey, date, linenumber, orderkey
from LineOrderJoined
```

Query 3

Redundant Joined table structure Pros/Cons

Pros:

- Joining data will be done inconstant time at the first time of data loading.
- Selecting from Join table will reduce the time relative to HIVE traditional Join map/reduce tasks.

Cons:

- Data redundancy represents overhead for data size because every dimension record Joined with fact record will be repeated by the number of Joins such as Cust#02 and Supp#01.

3.2.2 Indexed Joined Table

To overcome Redundant Joined Table structure repeated fields overhead, we can remove this repetition by referencing repeated dimension records by an index field. So, repetitions for new records will be ignored, but still their non-key fields represented as null fields which mandatory to be structured in CSV format and can be loaded into HIVE as table, also can run SQL Join query and refill null fields on result dataset from HIVE Joined table using index. To create indexed table structure, all fields from dimension tables will be gathered into one table plus all property fields in Fact Table by iterating through all records in Fact table and gather related dimensions data into one record as shown in Figure 11

Line Order										
Property Fields		Dimension Fields								Index
ORD ERKEY	LINENU MBER	CUST KEY	CUST NAME	PART KEY	PART NAME	SUPP KEY	SUPP NAME	DATE KEY	DATE	Index
1	1	1	Cus#01	2	rosy metallic	2	Supp#02	19920101	1-Jan-92	
2	3	1		1	lace spring	1	Supp#01	19920102	2-Jan-92	1
3	2	2	Cus#02	2		1		19920101		1,2
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	

Figure 11: Line Order Indexed Joined Table data.

by iterating through all star schema tables and gather every related record from all schema tables into one record and All repeated dimension records non-key fields will be null to overcome data redundancy. Adding a new column called index that hold referenced record index (pointer) then all data will be written to CSV format as shown in Figure 12.

```
1,1,1,Cust#01,2,rosy metallic,2,Supp#02,19920101,1-jan-92,0
2,3,1,          ,1,lace sping , 1,Supp#01,19920102,2-jan-92,1
3,2,1,Cust#02,2,          ,1,          ,19920101,          ,1 2
```

Figure 12: Line Order Table Indexed Joined data in CSV

Load Data into Hive

Creating a new table called "LineOrderJoinedIndexed", load data into HIVE using load statement then re-write Join query to run on HIVE.

HiveQL Query Parsing and conversion

Parsed query re-written as selection from one table without Joins as shown in query 4:

```
Select custkey, Custname, partkey, partname, suppkey,
suppname, datekey, date, linenumber,
orderkey,index from LineOrderJoined where
date=19920101
```

Query4

Fill in Result Phase

There are null fields in select query result need to be re-filled using Fill-in function (UDF Filling) for each record in result read index field and iterate through referenced records on the index field and fill null fields by their data from corresponding referenced records. Figure 13 explains how to fill second record from the result by iterating through record 1, and 2 from Joined table and fill missing fields (see Figure 13).

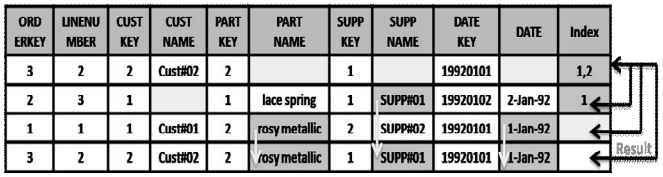

Figure 13: Null Record Filling process using Filling UDF.

Indexed Joined table structure Pros/Cons

Pros:

- Joined Schema Creation Time represents up-front time overhead for the first time only.
- Selecting from Joined table will reduce the time relative to HIVE traditional Join map/reduce tasks.

Cons:

- Index column represents Data overhead.
- Selecting Result Requires Fill in Step which requires loop through all Joined table records.
- Empty dimensions fields need separators character in CSV file format which represents also data overhead.

4. HIVE VS JOUM

HIVE execution of Query 1 will partition each two Join tables over all cluster nodes, then map each table partition data using projection and local predicates the result data will be shuffled into cluster nodes. Each shuffled partition will be sorted by tables keys and reduced by Join keys result now contains partitions each of them contains Joined data by their keys and other selection fields based on selection criteria. In the case of Joining more than two tables, previous operation will be repeated for each table by Joining first table with the second table then Join the result with the third table and vice versa. For aggregation and ordering by clauses, it needs to re-map result into cluster nodes and combine each map according to aggregation fields, shuffle combined data by aggregate fields, sort each partition then reduce each partition aggregate/ordered data.

Figure 14 represents the complex and repetitive operations of Join execution pipeline in Hive.

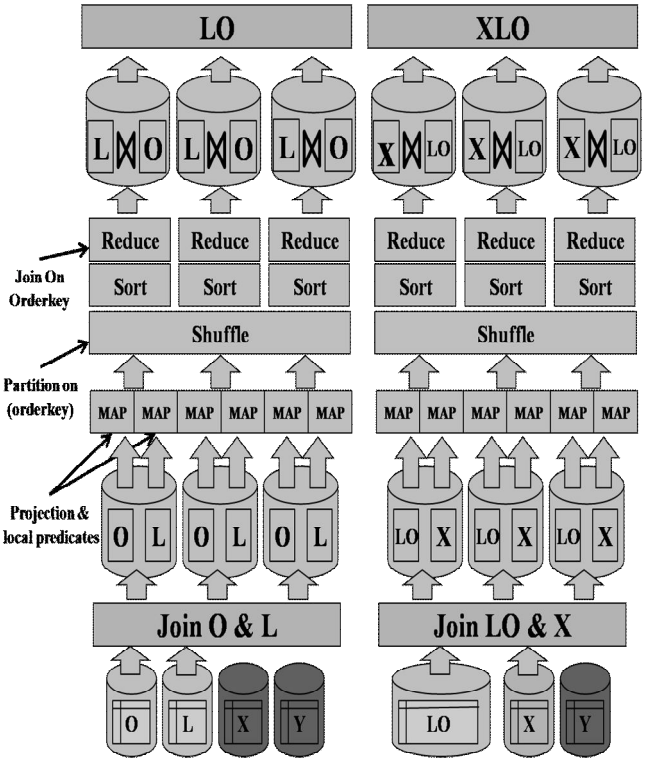

Figure 14: HIVE Join execution pipeline

Figure operation, how much slow Join query execution, how much temporary storage will be allocated for intermediate steps like sort and combine, and how much network bandwidth is needed for map and shuffle steps in HIVE execution pipeline. By executing previous query more times by different criteria, different aggregate clauses or different order by dimensions.

Every time of running this Join query needs to execute previous steps, allocating temporary storage, consuming networks bandwidth. For the node failure case, it needs to re-execute the whole tasks again and again. So, using this execution pipeline will be unreasonable for executing high dimensioned/complex Join query over Big Data sets.

The proposed JOUM system will do pre-Join of all schema data tables into one table using Join Table Builder then load this Joined table into HIVE framework. This two steps will only be executed at the first time of data loading. Through the execution of Join query JOUM first two Steps of partitioning and shuffling of data blocks will be discarded because JOUM system selects from one table as query 4. Figure 15 explains JOUM Join execution pipeline.

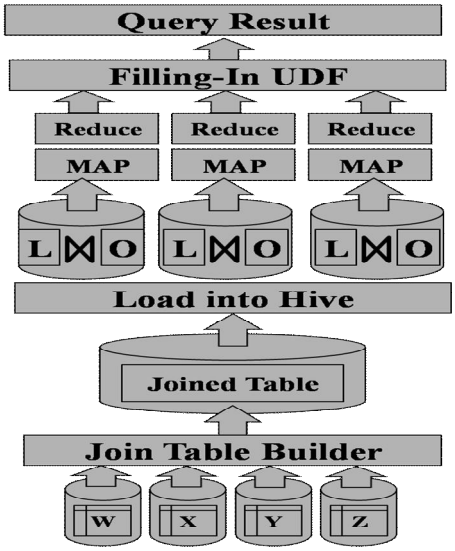

Figure 15: JOUM execution pipeline

By comparing Figure 15 and Figure 14 steps, the steps using JOUM will be reduced into three steps only (data projection –Map Reduce - fill in empty selected fields using UDF). this steps equivalently to normal select statement without Join plus extra steps which named Fill in Using UDF.

5. Experiment Setup

The proposed system is empirically evaluated effectiveness in speeding up analytical queries having selection predicates. The evaluation parameters which are used to evaluate the proposed JOUM system are:

1. The time savings in query response times.
2. The computation costs of building an index on load time.
3. The storage overhead of the indexes.

Overall, the experiments show that Indexed Joined tables yields significant speedups in query response times compared to HIVE star schema, while avoiding unreasonable overheads.

In the experiments, an index built over Joined table which in order to speeding up query response time by large factors, minimizing temporary storage required to execute Join query by large factor.

5.1 Cluster Setup

The proposed system is developed under Linux Ubuntu LTS 12.4 x64 server Installed on Hadoop cluster of 5 virtual machines over two PowerEdge T320 tower server Intel® Xeon® processor E5-2400 and E5-2400 v2 2.5MB cache per core with core options 4 with 2G RAM, 1G Ethernet network 200 GB Hard Disk over host OS Windows Server 2008 R2 using Java SE7 with JDBC connection over HIVE 11.0.0/Hadoop 2.2.0 thrift server. Hadoop master processes (Map Reduce Job Tracker, HDFS Name Node, and HIVE thrift server). The following configuration parameters are overridden in order to boost performance, JVM's

were re-used, speculative execution was turned off, and a maximum of 1GB JVM heap space was used per task. Repeating each experiment three times and reported the average of the results.

5.2 Dataset

Generated datasets from TPC-H star schema benchmark used the entity having the most records, namely, the Orders document from Line Items table from TPC-H. Generated datasets of 6 Million Record each Line Items consists of 16 distinct fields with no nesting—i.e., a flat structure—and a rigid schema.

5.3 The Performance Evaluation

According to Figures 16, and 17, it is found that temporary storage consumes up to 16% for Joined Indexed and up to 33% for Redundant Joined from HIVE required size. Permanent storage consumes up to 128 % for Joined Indexed and 186% for Redundant Joined form HIVE required size. In comparison between storage needed for permanent and temporary storage we notice that high storage is needed for permanent storage than HIVE because of index overhead but Low storage is needed for temporary storage than hive will be fair enough.

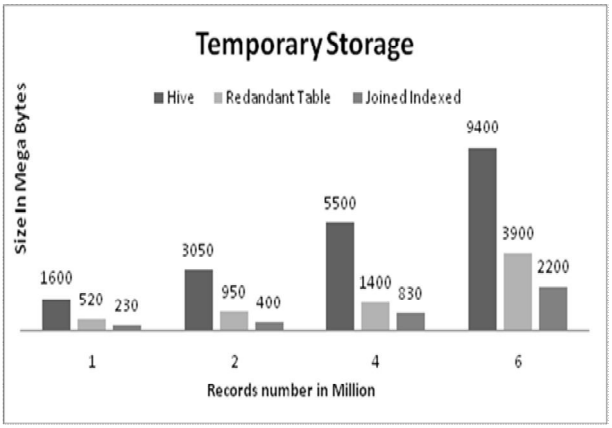

Figure 16:Temporary storage required for running PC-H Query #1.

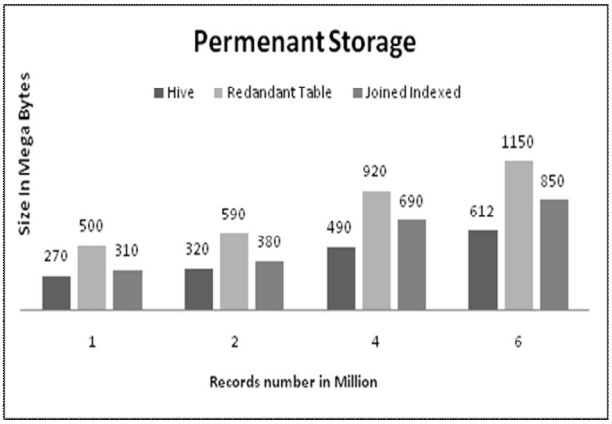

Figure 17:Permanent storage required for running TPC-H Query #1.

Figure 18, showing time required for building Joined Data table which upfront cost paid only at first time of loading data into HIVE. According to Figure 19, it is found that the execution time is reduced up to 50.5% for our Joined Indexed method and up to 68.1% for our Redundant Joined method. By increasing data size, our proposed methods computation time is fixed, while HIVE computation time will be increased. So, the performance of our proposed methods outperforms HIVE performance.

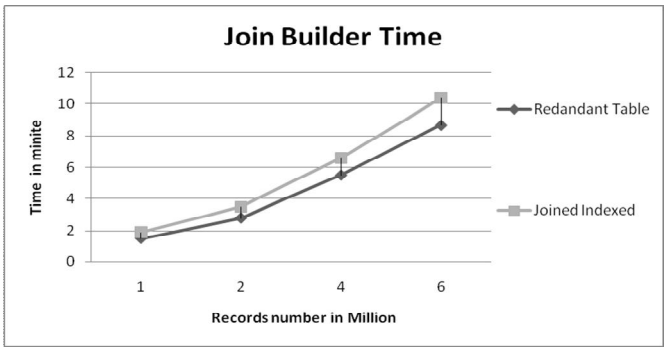

Figure 18: TPC-H Joined Table Building by Time

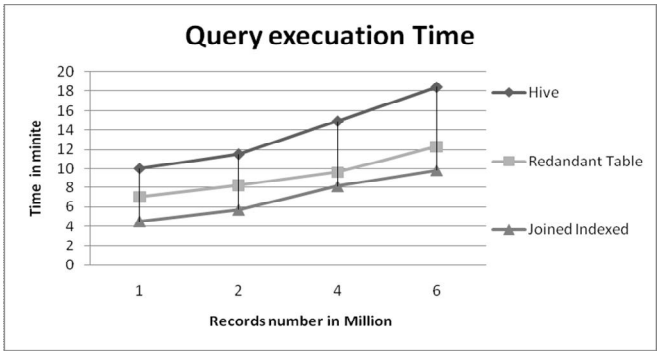

Figure 19: TPC-H Query #1 execution time by number of records.

Figure 20 indicates that on average of running different queries from TPC-H using the proposed methods doing well comparing to the HIVE by factors of 58.5% for Join Indexed and 66.1% for Redundant Joined Table. Figure 21 showing that large cluster size enhancing HIVE performance compared by the proposed methods because HIVE load balancing, and task distribution technique use cluster resources very well [21,22].

In small size cluster, the proposed solution is more powerful than HIVE. Also, for a large cluster, the proposed solution is still performing better enough than hive.

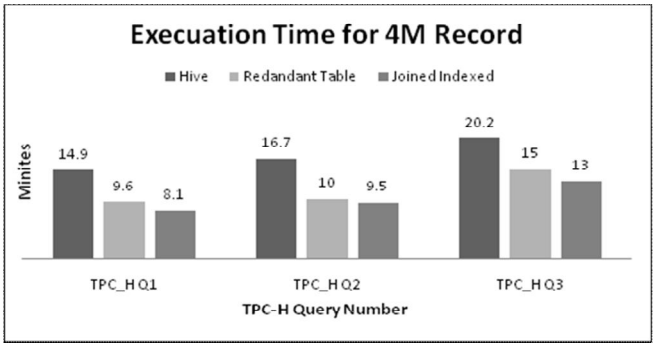

Figure 20: TPC-H Query #1 to #3 execution time by number of records for 1 million records.

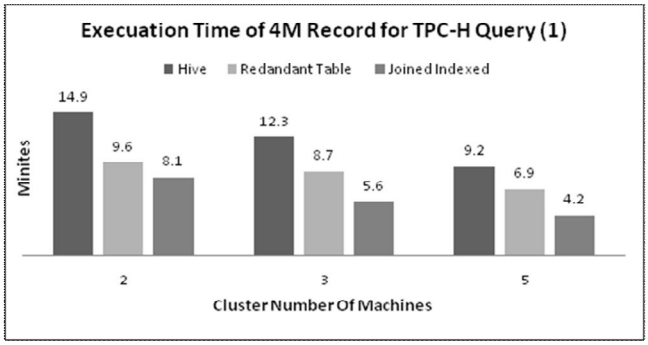

Figure 21: TPC-H Query #1 execution time for 4M record by cluster size.

6. Conclusions and Future Work

In this paper we introduced JOUM methodology which pre-Join the star schema data and build an index for Joined data are proposed. Based on JOUM methodology, SQL queries execution time in HIVE has been improved. Without changing HIVE framework. TPC-H benchmark has been used to evaluate the performance of JOUM methodology. The experimental results prove that JOUM methodology outperforms traditional Join execution time. Also, JOUM performance is improved by increasing data size. Generally, JOUM is one of the suitable methodologies for Big Data analysis..However, minimum overhead in permanent storage is produced because index is small compared to large size saved by temporary storage. In the future work, we need to minimize the storage and we want to implement JOUM in different Benchmarks [23].

References

[1] T. White, Hadoop: The Definitive Guide, O'Reilly, 2011.

[2] Dean, J., "Ghemawat Map-reduce: Simplified data processing on large clusters," OSDI: Proceedings of the 6th symposium on operating systems design, San Francisco, USA, PP. 137– 150, 2004.

[3] Dewitt, D. Stonebraker, M.,"Map-Reduce: A Major Step Backwards blog post, http://databasecolumnvertica.com/database-innovation/mapreduce-a-majorstep-backwards/Jan2007.

[4] Al Feel, H.T.; MohamedHelmyKhafagy,".OCSS: Ontology CloudStorage System", IEEE Network Cloud Computing andApplications (NCCA), 2011 First InternationalSymposium on Pages 9-13.

[5] Haytham Al Feel, Mohamed Khafagy, Search content viaCloud Storage System. International Journal of ComputerScience Issues (IJCSI) volume 8 Issue 6, 2011.

[6] Blumberg, Atre: The Problem with Unstructured Data. http://www.dmreview.com/issues/20030201/6287-1.html.

[7] Rolf Sint, Sebastian Scha ert,StephanieStroka and Roland Ferstl Combining Unstructured, Fully Structured and Semi-Structured Information in Semantic Wikis 2010.

[8] http://examples.citusdata.com/tpch_queries.html.

[9] Mina Samir Shenouda, Mohamed Helmy Khafagy, Samah Ahmed Senbel," JOMR: Multi-Join Optimizer Technique to Enhance Map-Reduce Job , the 9th International Conference on Informatics and Systems (INFOS2014),2014 pp 80-86.

[10] J. Lin et al. Full-Text Indexing for Optimizing Selection Operations in Large-Scale Data Analytics. In Map Reduce Workshop, pages 59–66, 2011.

[11] https://hive.apache.org.

[12] 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,pp 169:174

[13] Fatma A Omara, Marwah N Abdullah, Mohamed H Khafagy" HOME: HiveQL Optimization in Multi-Session Environment", 5th European Conference of Computer Science (ECCS '14),PP 80:89

[14] Join Processing in Relational Databases PRITI MISHRA and MARGARET H. EICH. ACM computing surveys Vol24. No. 1, March 1992.

[15] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. PVLDB, 2(1), 2009.

[16] J. Dittrich, J.-A. Quian´e-Ruiz, A. Jindal, Y. Kargin, VSetty, and J. Schad. Hadoop++: Making a YellowElephant Run Like a Cheetah (Without It Even Noticing). PVLDB, 3(1-2):515–529, 2010.

[17] E Sarhan, A Ghalwash, M Khafagy,"Agent-based replication for scaling back-end databases of dynamic content web sites", Proceedings of the 12th WSEAS international conference on Computers, 2008 pp 857-862.

[18] JensDittrich, JorgeArnulfo, Quian´eRuiz, Stefan Richter, Stefan Schuh, Alekh Jindal And J¨orgSchad Hail: Only Aggressive Elephants are Fast Elephants. arXiv:1208.0287v1 [cs.DB] Aug 2012.

[19] Stefan Richter, Jorge-ArnulfoQuian e-Ruiz, Stefan Schuh, JensDittrich: Towards Zero-Overhead Adaptive Indexing in Hadoop.http://infosys.cs.uni-saarland.de,arXiv:1212.3480v1 [cs.DB] 14 Dec 2012.

[20] Adaptive insertion policies for high performance caching Moinuddin K Qureshi, AamerJaleel, Yale N Patt ISCA07, June 9-13, 2007 ACM 978-1-59593-706- 3/07/0006.

[21] EbadaSarhan, AtifGhalwash, Mohamed Khafagy, Queue-WeightingLoad-Balancing Technique for Database Replication In Dynamic Content Web Sites ", APPLIED COMPUTER SCIENCE (ACS'09) University of Genova, Genova, Italy, 2009, Pages 50-55.

[22] Ahmed M WahdanHesham, A. Hefny, Mohamed Helmy Khafagy," Comparative Study Load Balance Algorithms for Map Reduce Environment" International Journal of Applied Information Systems,2014, Issues 7(11),pp 41-50.

[23] EbadaSarhan, AtifGhalwash, Mohamed Khafagy, Specification and implementation of dynamic web site benchmark in telecommunication area, Proceedings of the 12th WSEAS international conference on Computers 2008 Pages 863-867.