ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 10(19), DOI: 10.17485/ijst/2017/v10i19/108088, May 2017* ISSN (Online) : 0974-5645

Relational Query Optimization Technique using Space Efficient File Formats of Hadoop for the Big Data Warehouse System

Sudhanshu Shekhar Bisoyi*, Pragnyaban Mishra and S. N. Mishra

[1]Department of Computer Science and Engineering Science and Engineering, GIET Gunupur, Gunupur – 765022, Odisha, India; sudhanshu.bisoyi@gmail.com, pragnyaban@gmail.com, sarose.mishra@gmail.com

Abstract

Objectives: File structure and storage becomes a challenging issue while processing huge amount of data in parallel and distributed environment. To increase processing capabilities an appropriate file structure must be implemented. **Methods/Statistical Analysis:** In our approach we have imported the data from the available relational databases like Oracle or MySql to Hive using Sqoop and analyzed the query processing based upon different file storage formats. We have focused on the Parquet, Sequence, RC file and ORC file format for query analysis in MapReduce framework on top of Hadoop. **Findings:** Understanding dynamic behavior of user buying habits in different web services and product recommendation using social media, e-marketing etc. the MapReduce based data warehousing system plays vital role to perform the Big Data analytic in a parallel and distributed environment. In such type of analysis the data structure used to store the data for parallel query processing effect the performance of Big Data warehouse system. During the analysis of huge amount of relational data in a parallel and distributed system few issues should be taken care to increase the query performance and optimization. These are 1. Faster loading of huge amount of relational data into the Big Data warehouse. 2. Optimized file format to efficiently manage the storage system. 3. Faster query processing by increasing the throughput. Our findings explained appropriate file formats to store the huge amount of relational data in the Big Data warehouse system based upon HDFS and MapReduce framework known as Hive and evaluated the performance of query processing in multi node Hadoop cluster. **Application/Improvements:** The cost of parallel query processing has been reduced as well as distributed storage efficiency increased by choosing appropriate file structure in Big Data warehouse systems.

Keywords: Big Data, HDFS, Hive, Hadoop, MapReduce, ORC File, Sqoop

1. Introduction

In the recent years the large scale relational data analysis becomes very much popular because of the online marketing and social networking. Different marketing agencies take the help of the internet and the social media to enhance their sales of products in various region which leads to gather large amount of relational data. This large amount of relational data leads to the difficulty for the storage and analysis in general purpose machine and framework. For this reason the distributed framework like Hadoop gains popularity among all other frameworks available for storage and processing[1-4]. This framework stores huge amount of data and process it very first in a general i.e. commodity machine in parallel and distributed fashion. In order to perform the operation in a distributed environment Hadoop use the HDFS for the data storage and MapReduce for distributed processing. One critical task in social media and the e-commerce web sites is to understand quickly the dynamics of user behavior, trends and user needs based on their activities and buying habits.

For analyzing relational data the Hive gain popularity in the Hadoop ecosystem as a distributed data warehousing system[5,6]. The Hive is basically used on top of Hadoop with HDFS for storage and MapReduce for processing large

* *Author for correspondence*

relational data in a distributed environment. Hive uses the HQL known as Hive Query Language which is more similar to the SQL used in other relational database for the query processing[7,8]. In order to increase the performance of query the read-intensive queries[9] are widely used for analysis by considering the read performance as an important factor for the efficient query processing. The general architecture of Hive shows the complete storage and execution framework on top of Hadoop as shown in Figure 1. Hive uses two interfaces to submit the query which are CLI and HiveServer2. The HQL statement will be submitted to the Driver with the help of these two interfaces. First of all the parsing operation of the supplied HQL will be performed by the Driver and then the Abstract Syntax Tree (AST) corresponding to this statement will be passed to the Planner. The Planner then chooses a specific query plan implementation to analyze different types of statements. During the selection and processing of the planner implementation, the required metadata need to be referred from the Metastore of Hive which is present in a RDBMS like MySql.

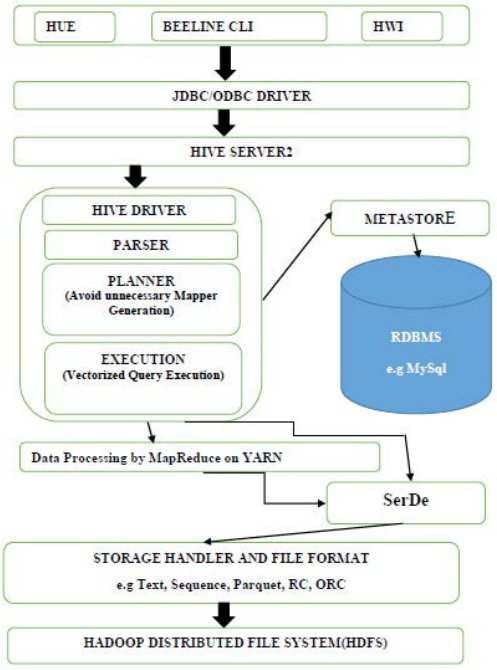

Figure 1. Architecture of hive.

2. Background Work

In this section, we provide a brief overview of the available data placement structures in HDFS and the Big Data warehouse system. The Hadoop provide a scalable and fault-tolerant infrastructure for Big Data analysis on large clusters which is based upon the open source implementation of MapReduce. Furthermore, MapReduce based data warehouse systems have been successfully built in major web service providers and social networking websites, and are playing critical roles for executing various daily operations including Web click-stream analysis, advertisement analysis, data mining applications and many others.

In the Big Data analysis, Hadoop based distributed systems are proposed for efficient analysis of large-scale structured, semi-structure and unstructured data stored on Hadoop Distributed File System (HDFS)[1,3]. Therefore, in building such a system to find an efficient data placement strategy that determines how to organize relational data in the underlying HDFS is a serious challenging task. Being a critical factor that can affect warehouse performance in a fundamental way, such a data placement strategy, storage format must be well optimized to meet the Big Data processing requirements[10] and the relational database reading operation need to be optimized to increase the query performance[11].

2.1 Data Placement Strategies for Big-Data Warehouse Systems

Every data warehouse system based upon MapReduce framework need to have efficient data placement strategy, proper storage formats and storage structure for fulfilling requirements like faster data loading in HDFS, faster query processing in the distributed environments, increasing storage optimization etc. In traditional database systems, three data placement structures have been widely studied[9,10], which are 1. Horizontal row-store structure, 2. Vertical column-store structure and 3. Hybrid PAX store structure.

2.2 Different Storage Formats used in Hive

The Storage formats available for the Big Data warehouse system like Hive are given as follows:

2.2.1 Sequence File

It is a binary file which is more compact then the text file. It enables the compression at block and record level. The

Sequence file can be split and processed in parallel. This file format is used to avoid small file problem of Hadoop framework by combining small files during processing. It consists of a header followed by one or more records. The first three bytes of a sequence file are the bytes SEQ, which act as a magic number; these are followed by a single byte representing the version number. The header contains other fields, including the names of the key and value classes, compression codec, user-defined metadata, and the sync marker. The sequence file doesn't solve the problem of tuple reconstruction and query optimization.

2.2.2 Apache Parquet

It is a columnar storage format that can efficiently store nested data. It is generally used where projection operations are in majority during query processing. Columnar formats are attractive since they enable greater efficiency, in terms of both file size and query performance. Query performance is improved too since a query engine can skip over columns that are not required.

2.2.3 RC File

The RC file (Record Columnar File)[10,12] is an optimized data storage structure for the MapReduce based data warehouse system that determines how to minimize the space required for relational data in HDFS (Hadoop Distributed File System). The RC file generally partitions the relational data horizontally first then convert into vertical partition as shown in Figure 2. It is guaranteed that columns of a particular record are stored in particular DataNode in the cluster by reducing the cost of tuple reconstruction and avoiding unwanted column read operation. It combines multiple functions such as data storage formatting, data compression, and

2.2.4 Data access optimization

This allows fetching only the specific fields that are required for analysis, thereby eliminating the standard time needed to analyze the whole table in a database. The overall data size reduction can be as large as 14% of the original data format. Apart from this entire RC file is data-type-agnostic and single-row serialization which leads to the inefficient utilization of various compression codecs[10]. Though it is a column oriented storage structure, indexing is not specified and the complex types e.g. Structure, Map, can't be decomposed by the SerDe. Because of which all fields of complex types need to be read by query.

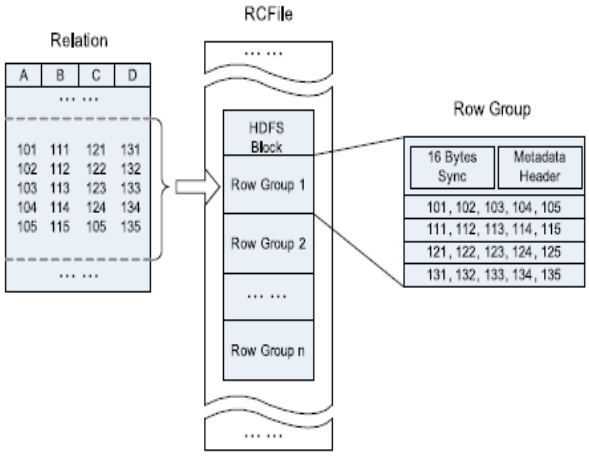

Figure 2. Structure of RC file.

2.2.5 ORC File

It is known as the Optimized Row Columnar file format. It provides highly efficient and intelligent way to store huge amount of data in Hive. ORC file format overcomes most of the limitations of all above file formats used in Hive for enhancing the storage and query execution in a parallel and distributed environment[13]. It reduces load of the NameNode by combining the output of each task into a single file. It also provides the light weight index storage within the file, block mode compression, concurrent read using separate RecordReader and memory binding for reading as well as writing operation[14]. The structure of ORC file is shown in Figure 3.

3. Proposed Work

It has been observed that the relational data analysis becomes very much popular because of its uses in online marketing, social media etc. During the query execution if multiple relations need to be joined then a large number of unwanted tuples are usually generated which leads to the increase in query execution time[7,8]. In this paper we have analyzed various query optimization technique in terms of file storage format and job execution strategy for relational data analysis using Hive and Sqoop on top of Hadoop. We have taken both the horizontal and vertical data format representation for the data storage and analysis in the Big Data warehouse in a distributed

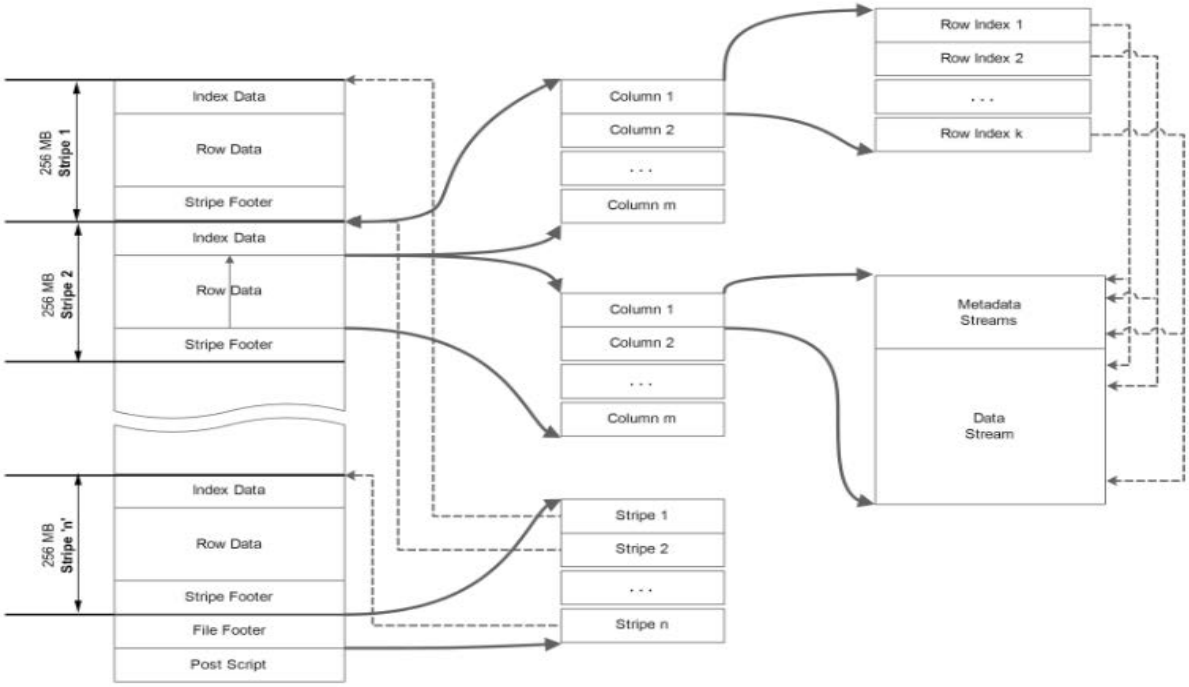

Figure 3. The structure of an ORC file.

environment. The file structures that we have analyzed have the following features.

- The data in a particular file format need to be shared over multiple data nodes and is based upon relational schema.
- The split size is decided basing upon the rows or columns as per the file format used over the distributed processing framework.
- The file formats effectively supports the optimization techniques of storage systems such as materialization, column-specific compression, and direct operation on compressed data. We have proposed a model by integrating the Hive, Sqoop, Oozie ecosystems on top of Hadoop to process large relational data in distributed environment. We need to have the seamless integration while evaluating any new infrastructure. Hence, it's important to carry out the processing without breaking any regular BI reports or workloads over the dataset planned to migrate. To work with our proposed model we have taken the real time retail database in MySql. The data from the MySql database need to be ingest into Hadoop Distributed File System (HDFS) with the help of Sqoop using different file format like Text, Sequence, Parquet, Avro, RC and ORC. The benefit of Sqoop is that all the relations from any RDBMS can be automatically loaded into HDFS, while preserving the structure. With a few additional configuration parameters, we can take this one step further and load this relational data directly into a form ready to be queried by Impala from Cloudera, the MPP analytic database, or Apache Hive the Big Data warehouse.

The steps followed for our proposed method are:

- Import the relations from MySql to Hive using Sqoop as Text file.
- Import the relations from MySql to Hive using Sqoop as Sequence file.
- Import the relations from MySql to Hive using Sqoop as Paraquet file.
- Create the tables using RC File and ORC File format and insert all the relational data respectively.

In the following we have represented a portion of scripts as part of our approach

- Importing as Text file

```
sqoop import-all-tables -m 12 --connect jdbc:mysql://
masterdb:3306/xxxxxx_db
--username=xxxxxx --password=xxxxxx
--as-textfile
--warehouse-dir=/user/hive/warehouse
--hive-import
```

- Importing as Sequence file

```
sqoop import-all-tables -m 12 --connect jdbc:mysql://
masterdb:3306/xxxxxx_db
--username=xxxxxx --password=xxxxxx
--compression-codec=snappy
--as-sequencefile
--warehouse-dir=/user/hive/warehouse/seq_db
--hive-import
```

Importing as Paraquet file

```
sqoop import-all-tables -m 12 --connect jdbc:mysql://
masterdb:3306/xxxxxx_db
--username=xxxxxx
--password=xxxxxx
--compression-codec=snappy
--as-parquetfile
--warehouse-dir=/user/hive/warehouse/paraq_db --hive-
import
```

The above Sqoop script may take a while to complete, because in backend it launch MapReduce jobs to pull the data from our MySql database and write the data to HDFS across the Hadoop cluster as per specified format. It is also creating tables to represent the HDFS files in Apache Hive with matching schema.

In order to import the data using ORC File and RC File format we can use Sqoop to first import the data using the Text File format then convert it into ORC File format by creating the separate table to hold the data in RC and ORC file format.

```
drop table if exists xxxxxx_orc;
create table xxxxxx_orc stored as orc tblproperties ("orc.
compress"="SNAPPY") as  select * from xxxxxx_table;
```

Once the retail data is ingest into hive, we execute the hive query to get the following information as part of the ETL work.

Query-1: Top 1000 frequently purchased items.

Query-2: Most popular product categories in the warehouse.

Query-3: Top 1000 revenue generating product.

All the above queries need to be evaluated in the entire mentioned file format.

The whole process of our proposed work has been carried out through the Oozie scheduler by scheduling the Sqoop and Hive job on top of Hadoop for the ETL operation as mentioned in Oozie workflow in Figure 4.

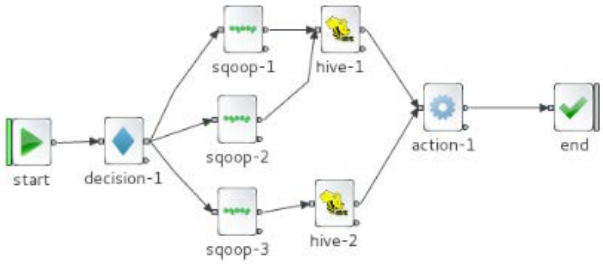

Figure 4. OOZIE workflow.

4. Performance Evaluation

The experiments have been performed to demonstrate the efficiency of various file formats used in the Big Data warehouse known as Hive to optimize the query processing for various BI reporting purposes. We have carried out our experiment by considering the Text, Sequence, Parquet, Avro, RC and ORC file formats.

The mentioned queries have been executed by Hive on top of Hadoop using MapReduse processing framework. Here we have used retail data for our analysis. In this experiment we have setup 5 node Hadoop cluster from Cloudera distribution CDH-5.5.0. Among all these 5 nodes the NameNode is having 8 GB of Primary Memory, 2.5GHz Processor and 500GB of Secondary Memory. The other 4 DataNodes are having 4 GB of Primary Memory, 2.5 GHz Processor and 500 GB of Secondary Memory.

We have imported all the retail data present in the relational database into Hive warehouse, but the data actually stored in HDFS. We have prepared separate database partition for all different file formats. The data are also stored by using different compression codecs like snappy, LZO etc. so that the efficiency in SerDe can be increased while executing the query in the parallel and distributed environment.

For our experimental evaluation we have considered few parameters like number of tuple read, number of bytes read/written, time required for the query execution in terms SerDe, CPU time required for execution etc. for all the file formats with different compression codecs.

The ORC file select less bytes as compared to the other file formats during the ETL operation of the Query-1 and 2 as shown in the Figure 5 and 6.

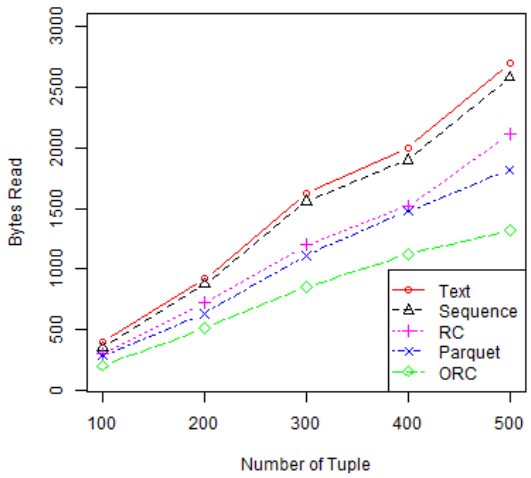

Figure 5. Total Bytes read during Query-1 processing.

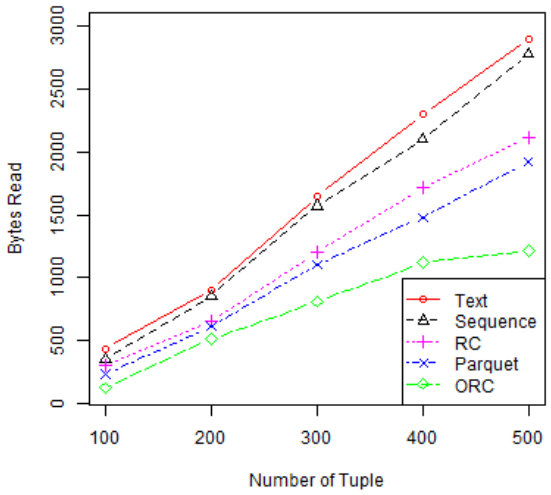

Figure 6. Total bytes read during Query-2 processing.

The ORC file generally takes long time while writing data into HDFS blocks which is shown in Figure 7 but the read performances while executing the query is greatly appreciable as shown in Figure 8 and 9. It also has been tested that the Parquet file format sows better performance if we have complex nesting data types used in the relations in the Because of its advancement in the columnar storage ORC file avoids unnecessary join operation and hence it reduce the generation of spurious tuple during query execution.

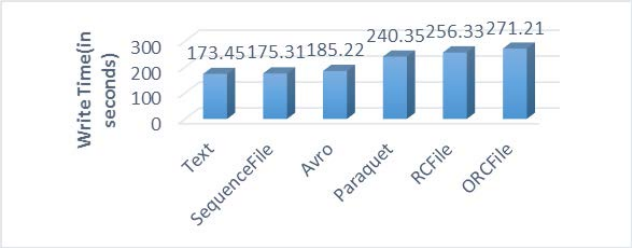

Figure 7. Write performance of the file formats.

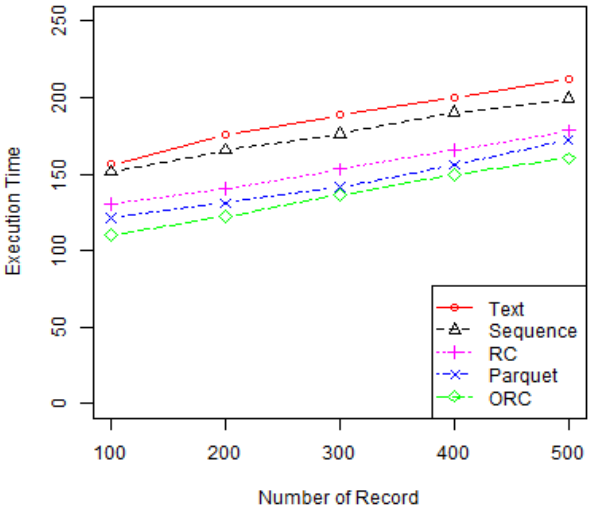

Figure 8. Read performance of Query-1.

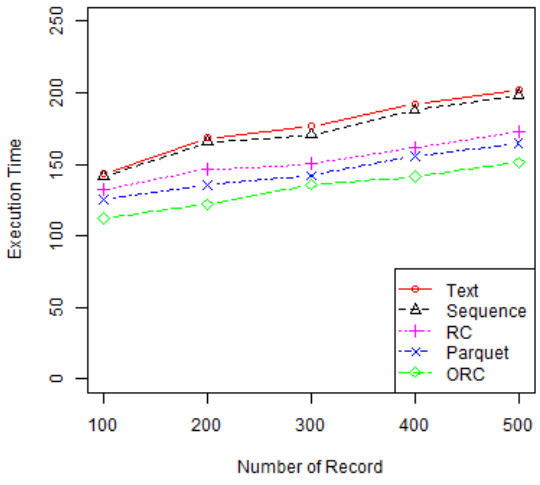

Figure 9. Read performance of Query-2.

5. Conclusion

In this paper we have evaluated the bulk data loading operation from the RDBMS to the Hive using Sqoop with the help of bounded values queries. The storage efficiency and query processing is also evaluated with the help of optimized file formats. We have performed a comparative analysis of all the file formats used in Hive during bulk data loading and performing various ETL operations. In general it is very difficult to finalize which file format along with compression codecs should be used. The query executed on the data stored in Hive with RC and ORC file format has shown better performance while extracting the report as part of the ETL operation. From our experiment it has been shown that ORC file takes long time while doing the writing operation but takes less time while reading data from the table as compared to all other file formats available. This increase the user recommendation system by providing better and quick result over a huge amount of retail data stored in HDFS. The analytical efforts in various file systems implementation have addressed several challenges in order to gain significant enhancement in Big Data warehousing system like Hive.

6. References

1. The Apache Software Foundation. Hadoop MapReduce. 2017. Available from: Crossref
2. Choi H, Son J, Yang H, Ryu H, Lim B, Kim S, Chung YD. A distributed data warehouse system on large clusters. Proceedings of the IEEE International Conference on Data Engineering; 2013. p. 1320–3.
3. Chen CP, Zhang CY. Data-intensive applications, challenges, techniques and technologies: A survey on big data. Information Science. 2014; 275: 314-47. Crossref
4. Doulkeridis C, Norvag K. Surveys of large-scale analytical query processing in map reduce. VLDB J. 2014; 23(3):355–80. Crossref
5. The Apache Software Foundation. Hive. 2016. Available from: Crossref
6. The Apache Software Foundation. Hive Wikipedia. 2011. Available from: Crossref
7. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wycko P, Murthy R. Hive: A warehousing solution over a map reduce framework. VLDB. 2009:1-4.
8. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wycko P, Murthy R. Hive: A petabyte scale data warehouse using hadoop. IEEE. 2010:1-10.
9. Thusoo A, Shao Z, Anthony S, Borthakur D, Jain N, Sarma JS, Murthy R, Liu H. Data warehousing and analytics infrastructure at Facebook. SIGMOD; 2010. p. 1013-20. Crossref
10. He Y, Lee R, Huai Y, Shao Z, Jain N, Zhang X, Xu Z. RCFile: A fast and space-efficient data placement structure in map reduce-based warehouse systems. ICDE; 2011. p. 1-10.
11. Harizopoulos S, Liang V, Abadi DJ, Madden S. Performance tradeoffs in read-optimized databases. VLDB; 2006. p. 487–98.
12. The Apache Software Foundation. 2016. Available from: Crossref
13. The Apache Software Foundation. 2016. Available from: Crossref
14. Holloway AL, DeWitt DJ. Read-optimized databases. PVLDB. 2008; 1(1):502–13.