International Journal of New Computer Architectures and their Applications (IJNCAA) 8(1): 11-20
The Society of Digital Information and Wireless Communications, 2017 ISSN 2220-9085 (Online); ISSN 2412-3587 (Print)

Measuring the Performance of Data Placement Structures for MapReduce-based Data Warehousing Systems

S. Kami Makki and M. Rakibul Hasan
Department of Computer Science
Lamar University
Texas, USA
kmakki@lamar.edu, mhasan4@lamar.edu

***Abstract* -** The exponential growth of data requires systems that are able to provide a scalable and fault-tolerant infrastructure for storage and processing of vast amount of data efficiently. Hive is a MapReduce-based data warehouse for data aggregation and query analysis. This data warehousing system can arrange millions of rows of data into tables, and its data placement structures play a significant role for increasing the performance of this data warehouse. Hive also provides SQL-like language called HiveQL, which is able to compile MapReduce jobs into queries on Hadoop. In this paper, we measure the efficiency of these data placement structures (Record Columnar File (RCFile) and Optimize Record Columnar File (ORCFile)) in terms of data loading, storage and query processing using MapReduce framework. The experimental results showed the effectiveness of these data placement structures for Hive data warehousing systems.

***Index Terms* - Big Data; Hive; MapReduce; ORCFile; RCFile**

1. INTRODUCTION

We are in the phase of fast technological advancement, where every sectors of any business is generating an unprecedented amount of data through its daily processes. Digitalization of every device and escalating the number of Internet users help to grow the data exponentially which makes the traditional data processing technology obsolete. According to ACI in 2012, 2.5 Exabyte's of data were generated in every day [3], and the total volume of data in the world is doubled every two years [10]. Statistics show that two billion people connected to the Internet in 2015, which is 100 times more than 1999 [8].

Therefore, the stunning growth in the number of users as well as variety of different devices that easily can connect to Internet generate huge and complex data or Big data. The authors in [1] defined the characteristics of Big data by the volume, variety, velocity and value. The Volume indicates the amount of data which is generated by the users, where this data has unknown value and low density. The Velocity refers to the rate of speed at which the data are generated and evaluated in real time manner to meet the users' demand and challenges. The Variety refers to the massive amount of data which is accumulated with huge speed, and can be of different types and nature such as structured, semi-structured and unstructured data. The Veracity indicates the quality of captured data, which totally depends on the source of data.

MapReduce framework provides a fault-tolerant infrastructure for processing of Big Data on large clusters. The MapReduce-based warehouse systems (such as Hive) are playing an important role for the effectiveness of web service providers and performance of social network websites. Hive stores large amount of data using distributed clusters [5], and efficient data placement structures are much needed factor for proper data organization. Hive uses two types of file format structures to store the data, such as RCFile (Record Columnar File) and ORCFile (Optimize Record Columnar File).

In this research, we investigate the effectiveness of RCFile and ORCFile of Hive data warehousing system. The goal of this research was to find out the most useful data placement structures that satisfy the requirements such as fast data loading, fast query processing, highly efficient storage space utilization, and strong adaptability to highly dynamic workload patterns. The rest of the paper has been organized as follow. Section 2, provides the background information on Big Data Technology, the Hadoop system architecture and Hive. In Section 3 we explore different data placement structures for Hive data warehousing system. Section 4 presents the performance evaluation of the RCFile and ORCFile for data loading, query processing and data storage. Finally Section 5 presents the conclusion.

2. BIG DATA TECHNOLOGY

2.1 *Hadoop*

Hadoop is a Java based open source system developed by Apache Software Foundation, which can store, process, and analyze a large volume of datasets in parallel on clusters of computers. Hadoop can scale up easily from a single server to hundred servers, where each server provides local computation and storage capability. Hadoop has four core modules: MapReduce, HDFS (Distributed storage), Yarn framework, and common utilities.

MapReduce is a programming model to process a large volume of datasets in parallel on clusters of computers [13]. MapReduce uses two methods: Map and Reduce. The Map method takes the raw data as input and breaks the data into numbers of smaller data sets. Each data set in a Map method receives a key/value pair and produces intermediate key/value pairs, and stores the output of a temporary storage system for further processing. The Reduce method combines all the intermediate key/value pairs based on the intermediate key and generates new sets of output. Besides Map and Reduce methods, shuffling is another process to transfer the data from Map processes to Reducers. The MapReduce tasks are executed in the local disk to avoid the network congestions, and the results will be sent to the appropriate servers [16, 17].

Hadoop has a file system named Hadoop Distributed File System (HDFS). The HDFS has master-slave architecture, which introduces master as NameNode, and slaves as a number of DataNodes [16]. NameNode controls the file system namespace and stores the metadata across the clusters. Metadata contains the information about where and which DataNode has stored which data files. The data files are broken into multiple pieces of blocks, and the size of each block is 128 MB by default. NameNode is responsible for namespace operation such as opening, closing, renaming files directories, and mapping blocks to DataNodes [4]. It does not control the block operation since DataNodes arrange the blocks whenever the system starts. Hadoop provides two mechanisms to make a NameNode consistent and protect it from the single point of failure. The first one is creating backup files of metadata to multiple file systems, and the other one maintains a secondary NameNode in a different machine. The secondary NameNode periodically merges the namespace images and keeps an updated copy in its own spaces. It provides a backup NameNode when original NameNode fails. Figure 1 shows the architecture of HDFS. A DataNode also stores data in different blocks in HDFS and allows read /write operations by clients [4].

DataNodes are responsible for performing block creation, deletion and replication according to the instructions by the NameNode. The NameNode and DataNodes are communicating by heartbeat messages every 3 seconds. A DataNode is considered to be dead if it does not receive a message within a few seconds.

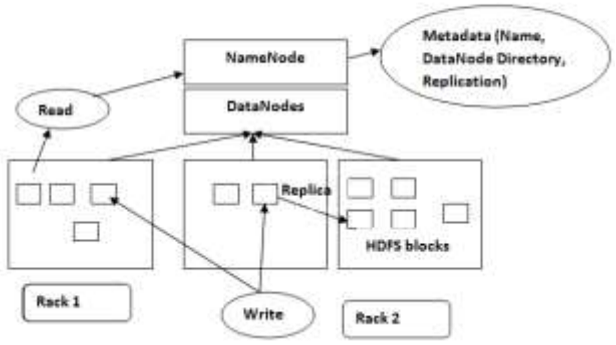

Figure 1. HDFS architecture [2]

2.2 Hive Data Warehousing System

The Apache Hive was developed by Facebook to manage its growing volume of data that Facebook produces everyday from its social networking activities [16]. Hive is included in Hadoop as a data warehouse. Hive supports SQL like scripts called HiveQL, to run a query on a large volume of data using MapReduce. Hive maintains metastore to contain schema and statistics for data exploration, query optimization and query compilation.

In Hive, data are organized in three formats, which are tables, partitions, and buckets. The tables are much like to the relational databases system. The second format is the partitions, which divides the data tables into subdirectories, which are defined by the data type characteristics. The last of data format is buckets, which can store data in both partitions and table's directory that depends on whether the table is partitioned or not [12,14]. Figure 2 shows the architecture and integration of Hive and Hadoop.

The External Interfaces supports different types of interfaces to initiate the works between users and HDFS, such as Command Line Interfaces (CLI), Web Interfaces, and programming interfaces (JDBC, ODBC).The Thrift Server supports cross-language services, which works with clients API to execute query statement. Metastore helps Hive to store the system catalog and metadata, where they contain details information about tables, columns, and partitions and so on [12]. Driver maintains sessions and statistics of HiveQL statement, which moves to Hadoop through Hive. Query Compiler compiles user defined HiveQLs into MapReduce tasks. Execution Engine is responsible for executing the tasks produced by compiler and MapReduce, which maintains the dependency order to communicate with the Hadoop modules.

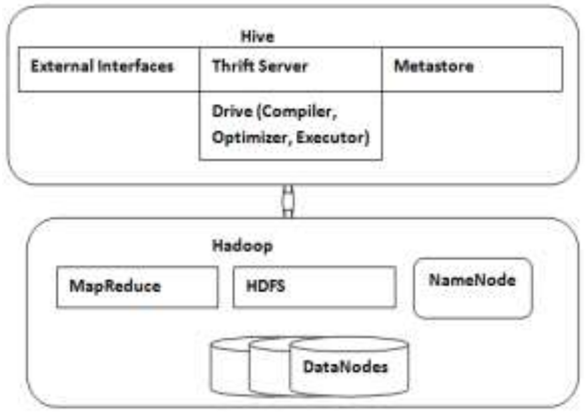

Figure 2. Architecture and integration of Hive and Hadoop

3. DATA PLACEMENT STRUCTURES

3.1 Record Columnar File (RCFile)

RCFile (Record Columnar File) is a data placement structure for MapReduce-based data warehouse systems (HIVE) which can organize a large volume of data on HDFS clusters. The RCFile is a combination of multiple features such as data storage format, data compression and data optimization techniques [2]. In data storage format, tables are stored first horizontally, and then vertically to organize each column independently in the clusters. RCFile supports column-wise data compression technique (lazy decompression) within each row group that helps to avoid unnecessary column reads during query execution. Furthermore, RCFile allows to select flexible row group size and arrange the same row data in the same node that increases the performance of data compression and query execution [5]. Figure 3 shows the layout strucure of a RCFile.

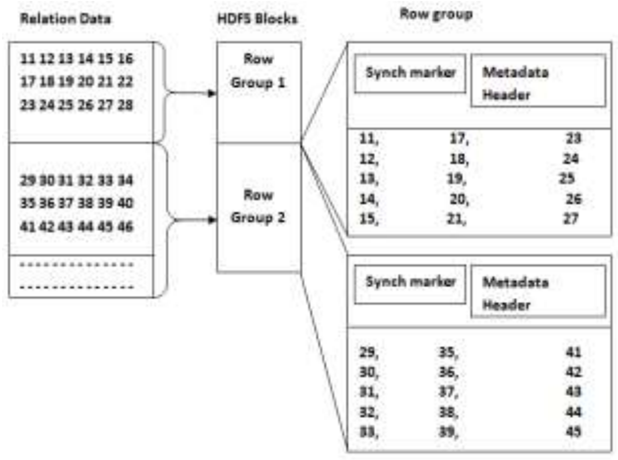

Figure 3. RCFile layout structure

A row group in RCFile has three components, which are synch marker, metadata header and column store. Synch marker is the beginning placement and helps to isolate the two-row groups in an HDFS block. Metadata stores the information of all row groups, such as how many row groups are placed, the size of each column as well as the size of each field in a column. The last one is column store, which helps to arrange all the fields in the same column together.

3.1.1 RCFile Data Compression

In RCFile, metadata header and table data sections are compressed separately. The metadata header section uses RLE (Run Length Encoding) algorithm to compress as a whole unit and the data table section compress each column independently [5].The RLE can find long run repeated values because all the values in a column are stored in continuously. For each column, RCFile supports separate algorithm to compress data in each table section. RCFile does not support random writing operation rather than it provides an interface for appending to the end of the file. RCFile maintains a memory column holder for writing data in each column. Before writing data to the disks, RCFile uses two parameters to control the memory buffer. The first parameter is to control the number of records, and the second parameter is to control the size of the memory buffer.

3.1.2 RCFile Lazy Decompression

In MapReduce framework for each HDFS block, a mapper is worked sequentially to process each row group. For reading, RCFile does not read the whole file rather than it reads only metadata and the corresponding columns to avoid the reading of unnecessary columns in the row group. Then the metadata header and compressed columns are loaded into the memory for decompression. RCFile uses lazy decompression technique that remains in memory until the other row groups are processed. Lazy decompression becomes useful when there is a `where` condition in a query, because if some row groups do not satisfy the `where` condition then those row groups do not need to be decompressed. For example, consider a table (T) with the following columns (`col1, col2, col3, col4,col5, …`) and if there is a query such as “`select col1, col3 from T where col2 = 2`”. Then the RCFile only reads the metadata header in the row group and decompresses only those row groups that match the `where` condition (i.e. `col2 = 2`) of the above query, not other row groups that do not match the `where` condition, and therefore it saves time.

3.2 Optimize Record Columnar File (ORCFile)

3.2.1 ORCFile Structure

ORCFile is a data placement structure similar to RCFile for Apache Hive for organizing and storing large data. ORCFile maintains one file for collections of rows, which is arranged in a columnar format that allows parallel processing of row collections in clusters [11]. ORCFile structure has three parts; these are stripes, file footer and postscripts. Figure 4 shows the structure of ORCFile [9].

Stripes hold the groups of row data and file footer maintains a list of stripes in the file, which contains the information of a number of rows per stripe and column's data type that includes aggregate functions such as count, min, max and sum. Each stripe size is 256 MB by default, which is suitable for a sequential read on HDFS. The larger block sizes will reduce the load of NameNode because the users can read more data from a single file.The last part is postscripts, which maintains compression parameter and the size of the compressed footer.

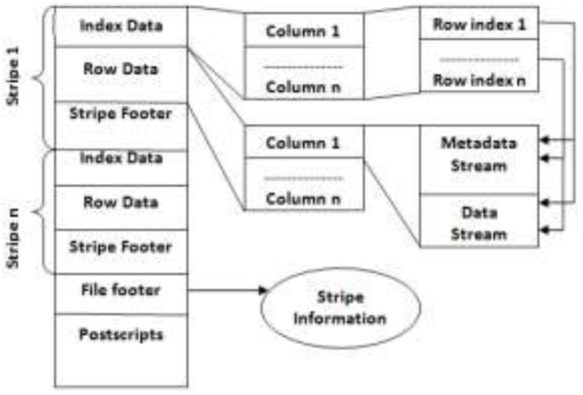

Figure 4. ORCFile structure

The stripes have divided further into three parts, which are index data, row data and stripe footer. Index data maintains the information of min, max values and the row positions for each column. These row positions are very efficient to find the specific compression and decompression blocks by providing the offset of indexes, where indexes are used to select the stripes and row groups. The row data stores multiple streams per column independently and uses them for table scans. Stripe footer provides directory services such as encoding types and stream locations.

3.2.2 Data Write and Compression

The ORCFile writer does not shrink the tables or whole stripes at a time rather than it applies data encoding and compression techniques [7]. To write data into HDFS, ORC uses memory manager to buffer the whole stripe in the memory. Due to large stripe size, ORCFile uses multiple writers concurrently to write data in a single MapReduce task, where memory manager controls the memory consumption of writers [6].

It supports two types of compression techniques. The first one is automatically used as type-specific encoding methods for columns with various data types and the second one is optional compression codecs set by users called generic compression. A column in a stripe can contain multiple streams, where each stream can be divided into four primitive types. These primitive types are bytes, run length bytes, integers, and bit field streams. Each stream uses the own encoding technique, which depends on the streams' data types. For example, integer columns data type are serialized into two streams, which are bit stream and data stream. For one bit or small integers, the variable length encoding is used, and for the long data streams of integers, the run length encoding technique is used. Besides using these type-specific encoding, users can also compress an ORCFile by general purpose codecs such as ZLIB, Snappy, and LZO [9] or others.

3.2.3 Data Read, Lazy Decompression and Lazy Decoding

In an ORCFile, the performance of data read is enhanced by lazy decompression technique [15]. Without lazy decompression and lazy decoding, a query seeks all the stripes to bring out a specific column, which will take a long time to finish the MapReduce tasks. This decoding technique is used index stride that already existed in ORCFile format. The index stride helps the reader to skip unnecessary stripes and only decompress and decodes the target columns needed by the query. The footer in the ORCFile has all the stripes that contain the streams location. Thus, the users query only read the stripe lists to find the appropriate stripe location.

4. Performance Evaluation of RCFile and ORCFile

4.1 Experimental Setup

The effectiveness of distributed systems depends on how one can perform the read and write operations, where the format of stored data is an important metric for completing these operations successfully. In this paper, we choose the following three aspects (data storage space, data loading time, and query execution time) to determine the best data structure between RCFile and ORCFile.

In this work, we have set up a virtual Hadoop cluster, which consists of three nodes. This cluster works as a master-slave architecture, where master node maintains the workspace to distribute, store and replicate data to slave nodes. We have installed virtual box software in each machine to configure Hadoop environment. The operating system in each virtual box was Ubuntu 14.04.2 LTS 64-bit. The host windows machine has 8 GB memory with 3 GHz Intel Core i7 CPU, where each node in the virtual box has shared 8 GB memory with 60 GB disk.

In our experiments, we used Hadoop 2.7.1 and Hive data warehouse 1.2.1. The MapReduce was chosen as an execution engine since it is the default data processing engine used by Hive. The HDFS block size was set to 128 MB, and its replication factor was set to three. For these experiments, we have used 6 GB dataset, which contained movie reviews (5 million reviews) from Amazon. The dataset consisted of eight columns: Productid, Userid, Profilename, Helpfulness, Score, Time, Summary, and Text. All of them were string data type.

4.2 Performance Analysis

4.2.1 Data Storage Space

We have configured Hive with RCFile and ORCFile to measure the compression efficiency. Figure 5 shows their storage sizes with compression and without compression. We have used LZ4 compression technique to compress the data using RCFile and ORCFile. The figure shows both file formats have reduced the size of data set significantly. The RCFile reduces the data size from 6 GB to 3.29 GB, where ORCFile reduces even more to 2.01 GB. That is because ORCFile uses larger data blocks than RCFile.

Therefore, each block can arrange more data in column format which allows compressing each column independently. Without compression, there is not much difference between a text file and RCFile, but ORCFile can decrease the file size significantly compare with other file formats because ORCFile uses a default compression technique (ZLIB). So, the ORCFile provides better storage efficiency than RCFile, whether using compression technique or not.

4.2.2 Data Loading Time

To demonstrate the data loading time of RCFile and ORCFile, we have measured both MapReduce time and the total time to finish the job. Figure 6 shows the data loading time after compression, where it shows that ORCFile take more time to load data than RCFile. We have taken another result shown in Figure 7, which shows the loading time for these two files before using compression technique.

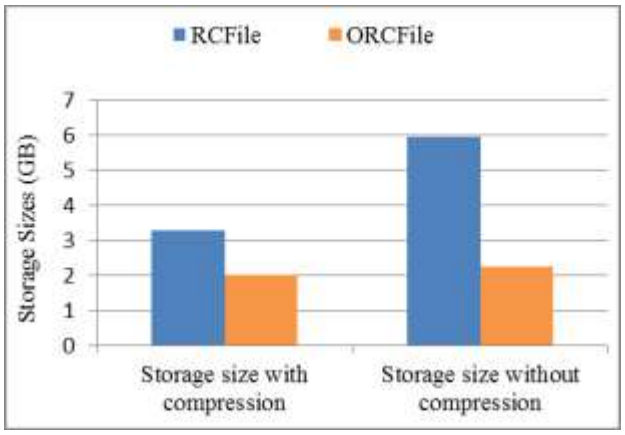

Figure 5. Data storage space of RCFile and ORCFile

4.2.3 Data Loading Time

To demonstrate the data loading time of RCFile and ORCFile, we have measured both MapReduce time and the total time to finish the job. Figure 6 shows the data loading time after compression, where it shows that ORCFile take more time to load data than RCFile. We have taken another result shown in Figure 7, which shows the loading time for these two files before using compression technique.

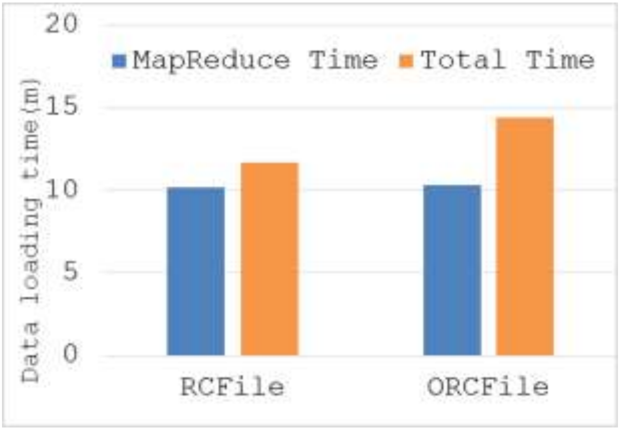

Figure 6. Data loading time with compression

4.2.4 Query Execution Time

For measuring the query execution time we used four queries as follow:

Query 1: `select productid from movie_rc;`

Query 2: `select reviewsummary from movie_rc where Profilename = "review/profileName: Jessica Lux" and userid = "review/userId: A2EBLL2OYEQJN9";`

Query 3: `select productid, userid, profilename from movie_rc where Score = "review/score: 50";`

Query 4: `select t1.productid, t2.userid from movie_rct1 right outer join movie_rc1 t2 on (t1.productid = t2.productid) where t1.profilename = "review/profileName: Jessica Lux";`

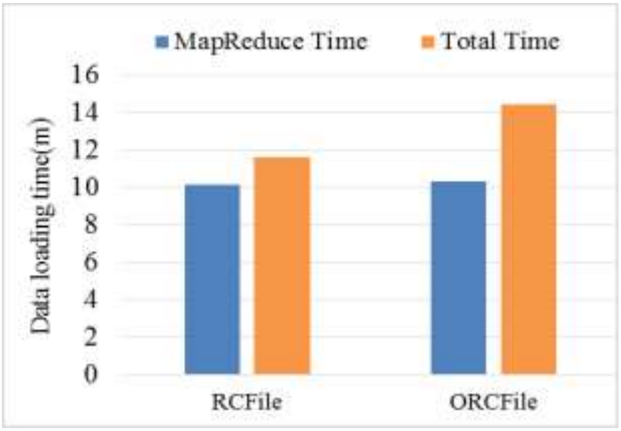

Figure 7. Data loading time without compression

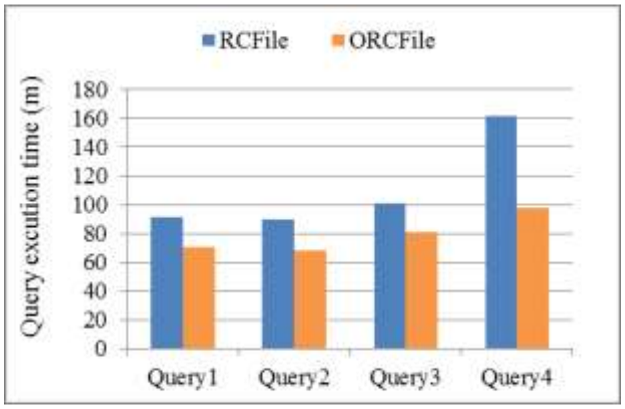

Figure 8. Query execution time with compression

Figure 8 shows the query execution time after data is compressed and Figure 9 shows the query execution time where data compression technique have been used. In both cases, ORCFile outperforms the RCFile significantly because the lazy decompression technique, which helps the ORCFile to skip a larger block of data if it does not match a query.

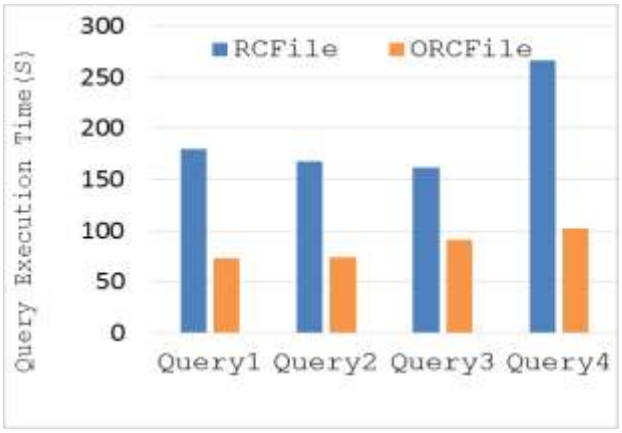

Figure 9. Query execution time without compression

4.3 RCFile and ORCFile with Different Row Group Size

Both RCFile and ORCFile allow the user to set flexible data block sizes because large data block can have better compression efficiency than a small one, where small data block may have better read or query performance than a large one. Furthermore, a large data block consumes more memory and can affect MapReduce tasks. In this experiment, we have used the same movie review database as above, and size of the dataset is 1.2 GB.

4.3.1 Data Storage Space

In this section, we have used different row group sizes for RCFile and different stripe sizes for ORCFile to demonstrate how they affect the storage space. Figure 10 shows the data storage efficiency of RCFile of different row group sizes (from 512 KB to 48 MB). Figure 11 demonstrates the data storage efficiency of ORCFile of different stripe sizes (from 4 MB to 256 MB).

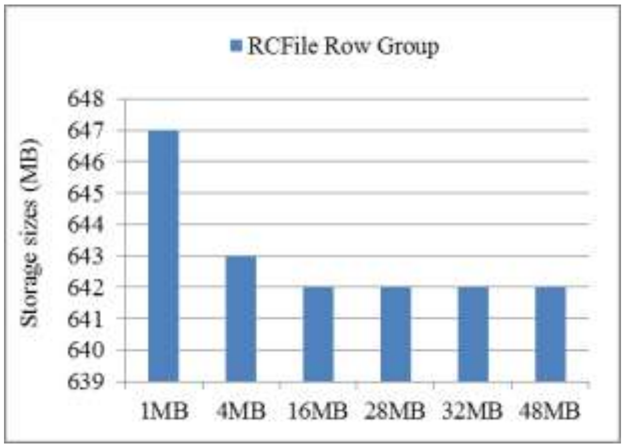

Figure 10. RCFile storage space with different row group sizes

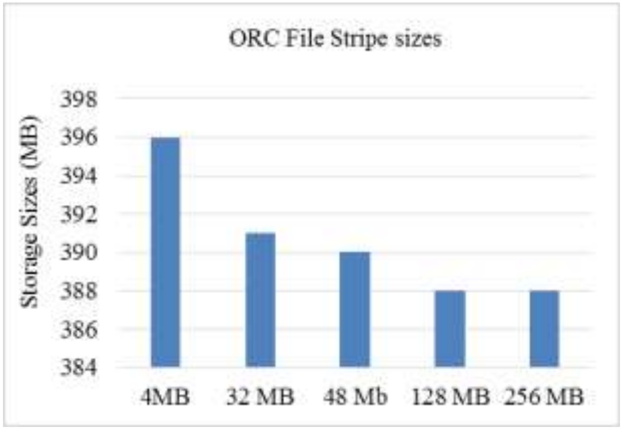

Figure 11. ORCFile storage space with different stripe sizes.

Figures 10 and 11 show that the larger block sizes can be compressed more than smaller block sizes. As a result, this reduces the storage space for both file formats. However, in the RCFile, when the row group size is larger than 4 MB, the storage space stays almost constant. As shown in Figure 11, compression is more in ORCFile when the stripe sizes are larger than 48 MB. So, the

ORCFile provides better storage space than the RCFile with larger data blocks.

4.3.2 Query Execution Time

In this experiment, we have evaluated the performance of lazy decompression technique for both RCFile and ORCFile. Figure 12 and 13 show the query execution time for RCFile and ORCFile. We have designed a query as below with different characteristic according to the `where` condition of a query.

```
Query 1: select review summary from movie_rc where PROFILENAME = "review/profileName: Jessica Lux";
```

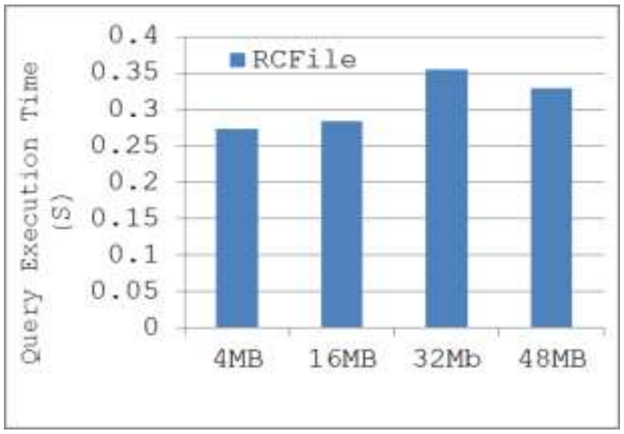

Figure 12. Query execution times of different data block sizes of RCFile

Figure 12 and 13 demonstrate that, when row group sizes are large, RCFile shows lower performance than ORCFile for query execution. The ORCFile has the better query performance when data block size is large (256 MB), because the ORCFile can skip large data block if it does not match the query based on lazy decompression technique.

5. CONCLUSION

There are four essential requirements for data placement structures, which are: to reduce the data loading time and storage space as well as enhancing the query performance and adaptivity of dynamic workload pattern. Our experimental findings showed that both file formats have significant characteristics which satisfy all four requirements of data placement structures mentioned above.

Between RCFile and ORCFile, the RCFile has a major inherent advantage in data loading time over ORCFiles. Since the RCFile has small row-group size than the ORCFile, which effectively reduces the data loading time. However, in the case of storage space and query execution time, the ORCFile outperform the RCFile. Though both data placement structures (i.e. RCFile and ORCFile) use column-wise compression, the large row-group size inside each stripe in ORCFile can hold and compress more data at a single time and reduces more storage space for ORCFile than RCFile. We have also observed that ORCFile can skip large numbers of unnecessary columns during a query, which significantly improves query performance. Thus, the ORCFile contains most of the performance benefits features and will play important role for data placement structures in MapReduce-based data warehouse systems on Hadoop.

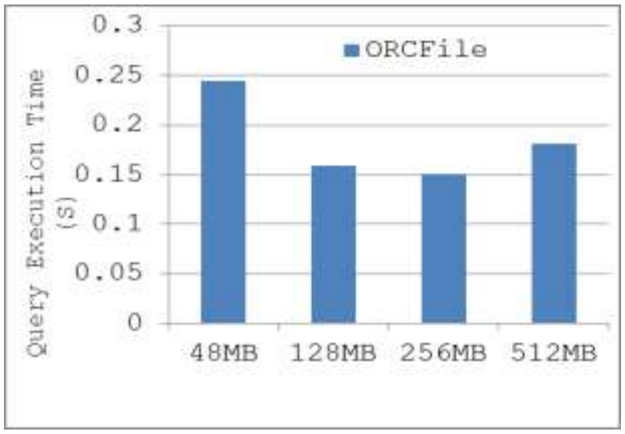

Figure 13. Query execution times of different data block sizes of ORCFile.

6. REFERENCES

[1] An Enterprise Architect ’ s Guide to Big Data.

ORACLE ENTERPRISE ARCHITECTURE WHITE PAPER.

[2] RC/ORC File Format.http://datametica.com/rcorc-file-format/

[3] The Data Explosion in 2014 Minute by Minute – Infographic. Retrieved January 8, 2017, from http://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic/

[4] HDFS Architecturehttps://hadoop.apache.org/docs/ stable/hadoop-project-dist/hadoophdfs/HdfsDesign. html

[5] He, Y., Lee, R., Huai, Y., Shao, Z., Jain, N., Zhang, X., & Xu, Z. (2011). RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. *Proceedings - International Conference on Data Engineering*, 1199–1208. http://doi.org/10.1109/ICDE.2011.57679 33

[6] Huai, Y., Chauhan, A., Gates, A., Hagleitner, G., Hanson, E. N., Malley, O. O., … Zhang, X. (2014). Major Technical Advancements in Apache Hive. *SIGMOD '14 - Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 1235–1246. http://doi.org/10.1145/2588555.2595630

[7] Huai, Y., Ma, S., Lee, R., O'Malley, O., & Zhang, X. (2013). Understanding insights into the basic structure and essential issues of table placement methods in clusters. *Proceedings of the VLDB Endowment*, *6*(14), 1750–1761. http://doi.org/10.147 78/2556549.2556559

[8] Internet live user. http://www.internetlivestats.com/internet-users/

[9] LanguageManual ORC https://cwiki.apache.org/confluence/display/Hive/Language Manu a l+ ORC#LanguageManualORC-orc-spec

[10] Big Data Solutions http://www.nttdata.com/global/ en/services/bds/index.html

[11] ORC: An Intelligent Big Data file format for Hadoop and Hive http://www.semantikoz.com/blog/orc-intelligentbig-data-file-format-hadoop-hive/

[12] Thusoo, A., Sarma, J., & Jain, Zheng S., Prasad C., Zhang N., Antony S., Liu H, and Murthy R., (2010). Hive-a petabyte scale data warehouse using hadoop. IEEE 26th International Conference on *Data Engineering (ICDE 2010), pp.* 996–1005. http://doi. org/10.1109/ICDE.2010.5447738

[13] Hadoop - MapReduce. http://www.tutorialspoint. com/hadoop/hadoop_mapreduce.htm

[14] Hive Introduction http://www.tutorialspoint.com/hive/hive_introduction.htm

[15] Vagata, P., & Wilfong, K. (2014). Scaling the Facebook data warehouse to 300 PB. Retrieved January 11, 2016, from https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300 pb/?attachment_canonical_url= https%3A%2F%2Fcode.facebook.com%2Fposts%2F229861827208629%2Fscaling-the-facebook-data-warehouse-to-300-pb%2F

[16] White, T. (2015). *Hadoop: The definitive guide* (Vol. 54). http://doi.org/citeulike-article-id:4882841

[17] Apache Hadoop https://en.wikipedia.org/wiki/ Apache_Hadoop