Volume 6, Issue 12, December 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Managing Data Using Table Partitioning and Querying with Mapreduce Software

Dina Darwish
Associate Professor, Multimedia and Internet Department, International Academy for Engineering and Media Science, 6th October City, Egypt

Abstract— MapReduce is considered as a programming tool and an associated implementation for processing and generating large data sets. Developers find the system easy to use: hundreds of MapReduce programs have been used and upwards of one thousand MapReduce jobs are processed on large companies cluster. In this paper, handling data using table partitioning and querying with mapreduce software is investigated where table partitioning algorithm is developed allowing table partitioning to split a table into smaller parts that can be reached, saved, and kept independent of one another. From their traditional use in enhancing query performance, partitioning strategies have developed into a powerful mechanism to improve the overall manageability of database systems. Table partitioning makes administrative tasks easier like data loading, removal, backup, statistics maintenance, and storage provisioning. Query language extensions allow applications and user queries to determine how their results should be partitioned before use. An optimization merging technique is created to generate efficient plans for querying data over partitioned tables in a mapreduce environment, where tables are partitioned and being queried after merge using SQL language from inside the HadoopDB program, which is a hybrid environment between parallel DBMSs and mapreduce, then, transforming the SQL indexing and query commands into the mapreduce program, which depends on mapreduce jobs represented as graphs. In this paper, an evaluation is done to compare the mapreduce performance using partitioned tables with the original mapreduce performance, and the results were discussed and analyzed. The results showed that the performance of mapreduce is good with the use of table partitioning and querying.

Keywords— HadoopDB environment, table partitioning, querying partitioned tables, partition-aware optimization technique, map-reduce performance.

I. INTRODUCTION

Today, there exist a flood of data. In a broad range of application areas, data is gathered at unprecedented size. Decisions that were based on guesswork, or on models constructed of reality, can now be taken based on the data itself. Big Data analysis leads almost every aspect of our modern society, including mobile services, retail, manufacturing, financial services, life sciences, and physical sciences. Scientific research has been advanced by Big Data [1]. In the field of Astronomy, the astronomer's task has been changed from taking pictures of the sky to finding interesting objects and phenomena in the pictures stored in a database. In the biological sciences, a tradition of depositing scientific data into a public repository has been established to create public databases for use by other scientists. An entire discipline of bioinformatics is oriented to the curation and analysis of these data. The size and number of experimental data sets available is growing gradually, as the technology advances, particularly in the field of Next Generation Sequencing.

Big Data has the capability of revolutionizing not just research, but also education [2]. It is found by a recent detailed quantitative studies that one of the top five policies related to measuring academic effectiveness is using data to guide instruction [3]. In education, it is better to collect detailed measures of every student's academic performance inside a huge database, which everyone can access easily. There is a big benefit from this data, because, it can be used in the design of the most effective approaches in education, starting from reading, writing, and math, to advanced, college-level, courses. It is still far from having access to such data, but there are powerful directions in this way. Massive Web deployment of educational activities is a strong trend, and this will produce a large amount of detailed data about students' performance. The use of information technology can decrease the cost of healthcare and improve its quality [4], by making care more personalized and basing it on home-based continuous monitoring.

The potential benefits of Big Data are real and important, and some initial successes have already been established, but, there are still many technical challenges that need to be treated to fully realize this potential.

Table partitioning, has become a standard characteristic in database systems [5, 6, 7, 8], and is one of the Big Data challenges. For example, a table can be partitioned horizontally based on value ranges or be partitioned vertically with each partition containing a subset of columns in the table. Hierarchical combinations of horizontal and vertical partitioning can also be used. The direction towards growing data sizes has increased the use of partitioned tables in database systems. Apart from providing major performance enhancements, partitioning makes a number of common administrative tasks more simple in database systems.

Many companies have chosen MapReduce, because it is simple, and scales to large clusters of commodity machines. MapReduce, and particularly its free open-source implementation Hadoop, is widely used, and depends on its frequent utilization on data centers of Google, Yahoo, Facebook, and others. MapReduce has been implemented to create search indices, analyze log files, extract information from large bodies of unstructured text, run graph algorithms on networks, and execute many other data intensive jobs. The MapReduce programming model contains two functions: The map function transforms input data into (key, value) pairs, and the reduce function is implemented to each list of values that correspond to the same key. This programming model avoids complex distributed systems issues, then, giving users rapid utilization of computing resources.

In this paper, the focus will be on horizontal table partitioning depending on alphabetical (lexicographic) order in centralized row-store database systems using MapReduce platform. The growing usage of table partitioning has been accompanied by efforts to give applications and users the ability to specify partitioning conditions for tables that they derive from base data. SQL extensions from database vendors now enable queries to specify how derived tables can be partitioned [9]. The rest of this paper is divided as follows. Section 2 provides a descriptive background of Hadoop and MapReduce. Section 3 provides a detailed description of the proposed table partitioning technique. Section 4 analyzes the results obtained from implementing this technique. Section 5 gives conclusions and future work.

II. HADOOP AND MAPREDUCE AS DEVELOPMENT ENVIRONMENTS

A. Hadoop architecture

Hadoop is considered as an open-source, Apache Software foundation project dependent on Java. It is one of the most important third-party applications of MapReduce for distributed computing. Hadoop presents a framework to support large dataset distributed computing on a cluster of servers. Hadoop has its own distributed file system called Hadoop Distributed File System (HDFS), HDFS is a distributed file system developed to save multiple copies of data block on a cluster of nodes, to perform reliable and rapid computations. Hadoop is executed in a distributed environment. A simple Hadoop cluster is composed of one or more machines executing the Hadoop software [10]. Hadoop implements HDFS to save a dataset and makes use of the MapReduce's power to partition and perform the processing of this dataset in a parallel way. Stream data is first divided into typical HDFS-sized block (64MB) and then smaller packets (64KB) by the user [11]. In other words, a file in HDFS is partitioned into 64 MB chunk and each chunk is kept on a different working machine.

Hadoop is considered really as a very useful tool for developing distributed programs that achieve better computational efficiency [12]. Hadoop presents two main services: reliable data storage (HDFS) and high-performance parallel and distributed data processing using MapReduce. Hadoop can execute on different types of machines.

Hadoop cluster configuration runs using a master/slave concept. The master server manages all activities within a cluster and slaves. Workers work for the master node. There exist two master nodes, including namenode and job tracker and N number of slave nodes, including datanodes and task trackers in the master/slave architecture of Hadoop as shown in Fig. 1.

Namenode, which is considered as a single master server in a cluster environment, handles the file system data inside the whole Hadoop's distributed file system and adjusts the read/write access to the data file by the user. There is a huge number of datanodes, one datanode exits per computer node in the cluster, these datanodes are used to save and extract the data, and they will be executing the read and write commands from the file system clients. Datanodes are assigned the role of processing replication tasks and more importantly ordering the file system data [13]. Namenode and datanodes components have fundamental roles in a Hadoop file system. Job tracker takes and manages submitted jobs in a cluster environment, and is assigned the role of scheduling and monitoring all running tasks on the cluster. It partitions the input data into many pieces of data to be executed by the Map and Reduce tasks. Task trackers run Map tasks and Reduce tasks via received instructions from the job tracker. They are assigned the role of partitioning Map outputs and ordering/grouping of reducer inputs. The job tracker and task trackers constitute the backbone of running a MapReduce program.

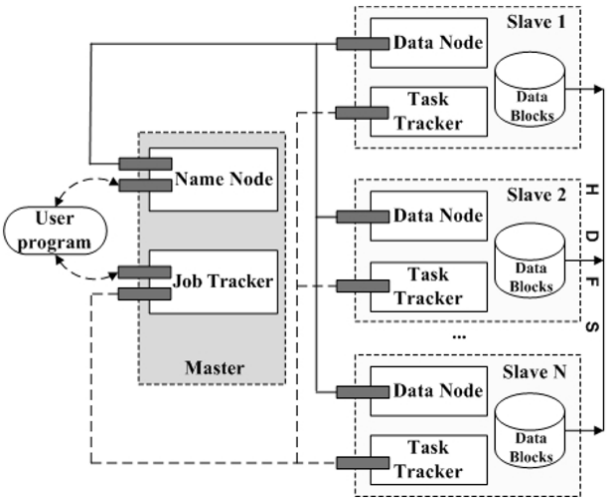


Fig. 1 Hadoop architecture

B. Hadoop implementation of MapReduce

At the API level, the MapReduce programming model simply contains two functions: The map function converts input data into (key, value) pairs, and the reduce function is implemented to each list of values that are related to the same key. This programming model avoids complex distributed systems problems, thereby, giving users rapid utilization of computing resources. To obtain parallelism, the MapReduce system essentially uses "group data by key, then implement the reduce function to each group". This computation model, called MapReduce group by, allows parallelism, because both the extraction of (key, value) pairs and the application of the reduce function to each group can be done in parallel on many nodes. The system code of MapReduce makes use of this computation model and other functionality such as scheduling, load balancing, and fault tolerance. The MapReduce program of an analytical query is composed of both the map and reduce functions generated from the query of a MapReduce based query compiler [14] and the MapReduce system's code for parallelism.

Hadoop is the most popular open-source implementation of MapReduce. Hadoop implements block-level scheduling and a sort-merge technique [15] to apply the group-by functionality used for parallel processing (Google's MapReduce system is considered to use a similar implementation [16], but there are some lacking details due to the use of proprietary code). The Hadoop Distributed File System (HDFS) manages the reading of job input data and writing of job output data. The unit used for storing data in HDFS is a 64MB block by default, and can be adjusted to other values during configuration. These blocks constitute the task granularity for MapReduce jobs. For a given query job, several map tasks (mappers) and reduce tasks (reducers) begin to run at the same time on each node. Fig. 2 shows that each mapper reads a portion of input data, implements the map function to extract (key, value) pairs, then gives these data items to partitions that relate to different reducers, and finally orders the data items in each partition by the key. Hadoop currently provides a sort on the compound (partition, key) to reach both partitioning and sorting in each partition. Knowing the relatively small block size, a properly-tuned buffer can allow sorting to be finished in memory. Then, the ordered map output is written to disk for fault tolerance. A mapper finishes after the write finishes. Then, Map output is shifted to the reducers. To do so, reducers send periodically a centralized service inquiring about completed mappers and once notified, request data directly from the completed mappers.

In most cases, this data transfer occurs after a mapper finishes, and so this data is available at the mapper's memory. A reducer gathers parts of sorted output from many completed mappers. This data cannot be considered as before to fit in memory for large workloads. As the reducer's buffer becomes full, these ordered pieces of data are combined and written to a file on disk. A background thread integrates these on-disk files whenever the number of such files exceeds a threshold (in a so-called multi-pass integration phase). When a reducer has gathered all of the map output, it will proceed to finish the multi-pass integration, so that the number of on-disk files becomes less than the threshold. Then, it will make a final integration to generate all (key, value) pairs in sorted order of the key. As the final integration continues, the reducer implements the reduce function to each group of values that possess the same key, and writes the reduce output back to HDFS. Also, if the reduce function is commutative and associative, as shown in Fig. 2, a merge function is implemented after the map function to provide partial aggregation. It can be also applied in each reducer when its input data buffer becomes full. MapReduce online, which is an advanced system, that adopts a Hadoop Online Prototype (HOP) with pipelining of data [17]. This prototype possesses two unique characteristics: it can bring data to the reducers, with the granularity of transmission managed by a parameter, since each mapper generates output. An adaptive mechanism is adopted to balance the work between the mappers and reducers. A potential benefit of HOP is that with pipelining, reducers obtain map output earlier, and can start multi-pass integration earlier, thereby, decreasing the time required for the multi-pass integration after all mappers complete.

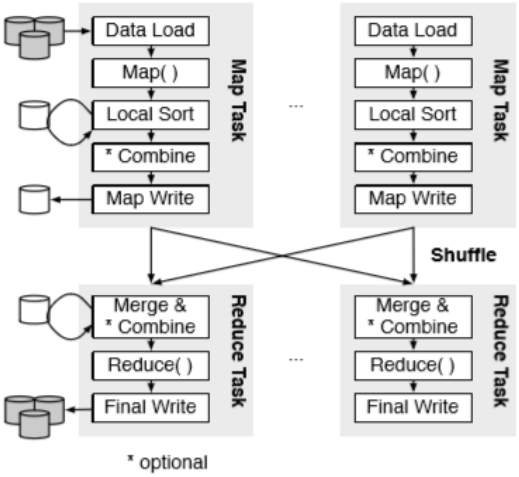


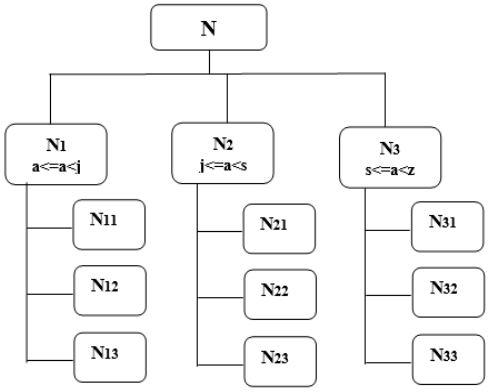
Fig. 2 Architecture of the Hadoop implementation of MapReduce

III. PROPOSED ALGORITHM FOR TABLE PARTITIONING

Table partitioning is considered as a standard characteristic in database systems today. A table can be partitioned horizontally based on value ranges of a column or, can also be partitioned vertically with each partition including a subset of columns in the table. Hierarchical combinations of horizontal and vertical partitioning can also be adopted. The rapidly growing data sizes has intensified the implementation of partitioned tables in database systems. Away from

giving major performance improvements, a number of common administrative tasks in database systems are simplified by partitioning.

Horizontal table partitioning in centralized row-store database systems, such as popular open-source systems, MySQL and PostgreSQL, have been studied. The increasing use of table partitioning has been combined with efforts to give applications and users the ability to determine partitioning conditions for tables generated from base data. SQL extensions from known database companies enable queries to determine how to partition derived tables. With these extensions, Database Administrators (DBAs) may not be able to manage or restrict how tables treated in a query are partitioned. Objectives and constraints may require to be applied during choosing how each table in the database is partitioned. Query optimization technology has continued to grow with the growing use of table partitioning. Query optimizers had to take into consideration the restricted partitioning schemes determined by the DBA on base tables. Today, the optimizer has to deal with diverse mixture of partitioning techniques that extend traditional techniques like hash and equi-range partitioning. Hierarchical partitioning is a scheme adopted to manage multiple granularities or hierarchies in the data. A table is first partitioned based on one attribute. Each partition is further partitioned based on a different attribute; and so on for two or more levels. Fig. 3 illustrates an example of hierarchical partitioning for a table N(a,b), where attribute a contains a character and attribute b contains a character.



Partition ranges for second level (i=1,2,3): Ni1: a<=b<j Ni2: j<=b<s Ni3: s<=b<z

Fig. 3 A hierarchical partitioning of table N

A hierarchical (multidimensional) partitioning of table N is partitioned equally on ranges of a into three partitions N1-N3, each of which is also partitioned into three partitions on ranges of b. Fig. 4 illustrates how the hierarchical partitioning of table N can be considered as a two dimensional partitioning. The figure illustrates partitions for tables M(a), O(a), and P(b). M, N, and O are all divided based on a condition for multiple related data sources and with different ranges due to data characteristics and storage considerations. Nine or eight letters ranges are used for recent partitions of P. The flexible nature and increasing complexity of partitioning methods constitute new challenges and opportunities during the optimization of queries over partitioned tables. Let's Consider an example query Q1 over the partitioned tables M, N, and O in Figure 5. Q1: Select * from M, N, O Where M.a=N.a and N.a=O.a and N.b≥j and O.a<s. Most current optimizers will stop here as far as exploiting partitions during the optimization of Q1 is taken into consideration; and produce a plan like Q1P1 shown in Fig. 5.

In a plan like Q1P1, the leaf operators combine together (i.e., UNION ALL) the partitions for each table. Each partition is reached using regular table or index scans. The appended partitions are merged using operators like hash, merge, and (index) nested-loop joins. Based on the data characteristics, physical design, and storage characteristics in the database system, a plan like Q1P2 illustrated in Fig. 5 can remarkably perform better than plan Q1P1. Q1P2 makes benefit from a number of characteristics coming from partitioning, as follows:

A. Records in partition M1 can merge only with N12 ∪ N13 and O1 ∪ O2. Similarly, records in partition M2 can merge only with N22 ∪ N23 and O3 ∪ O4. Thus, the full M⋈ N⋈ O join can be divided into smaller and more efficient partition-wise joins.

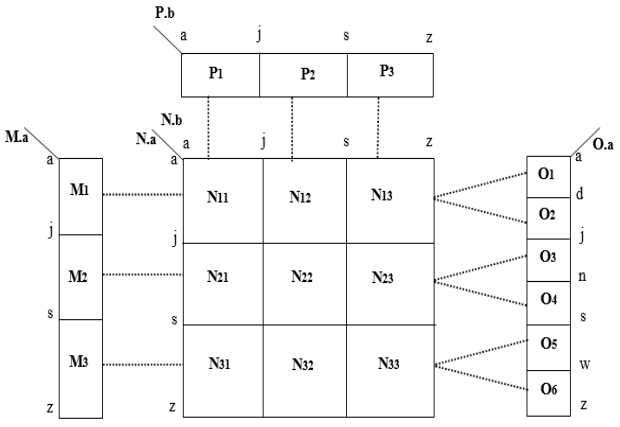


Fig. 4 Partitioning of tables M, N, O, P. Dotted lines show partitions with potentially joining records

B. The best join order for M1 ⋈ (N12 ∪ N13) ⋈ (O1 ∪ O2) is not the same as that for M2 ⋈ (N22 ∪ N23) ⋈ (O3 ∪ O4).

One likely reason is change in the data characteristics of tables N and O over time, resulting in differences in statistics across partitions.

C. The best choice of join operators for M1 ⋈(N12 ∪ N13)⋈ (O1 ∪ O2) may vary from that for M2 ⋈ (N22 ∪ N23)⋈(O3 ∪ O4), e.g., due to storage or physical design variations across partitions. Let us study query Q2 to understand the optimization of queries over partitioned tables. Q2 is a typical query in traditional star schemas where a fact table is joined with several dimension tables on different attributes. Q2: Select * From M, N, P Where M.a=N.a and N.b=P.b and P.b≥j and M.a<s; Plan Q2P1 from Fig. 5 illustrates the plan that combines the partitions before executing the joins. Given the join and partitioning conditions for M and N, the optimizer has the option of creating the partition-wise joins M1⋈(N12 ∪ N13), M2⋈ (N22 ∪ N23). The output of these joins is partitioned on attribute a, and that does not influence the later join with table P, producing the plan Q2P2 in Fig. 5. Also, the optimizer can provide partition-wise joins between N and P first, producing the plan Q2P3.

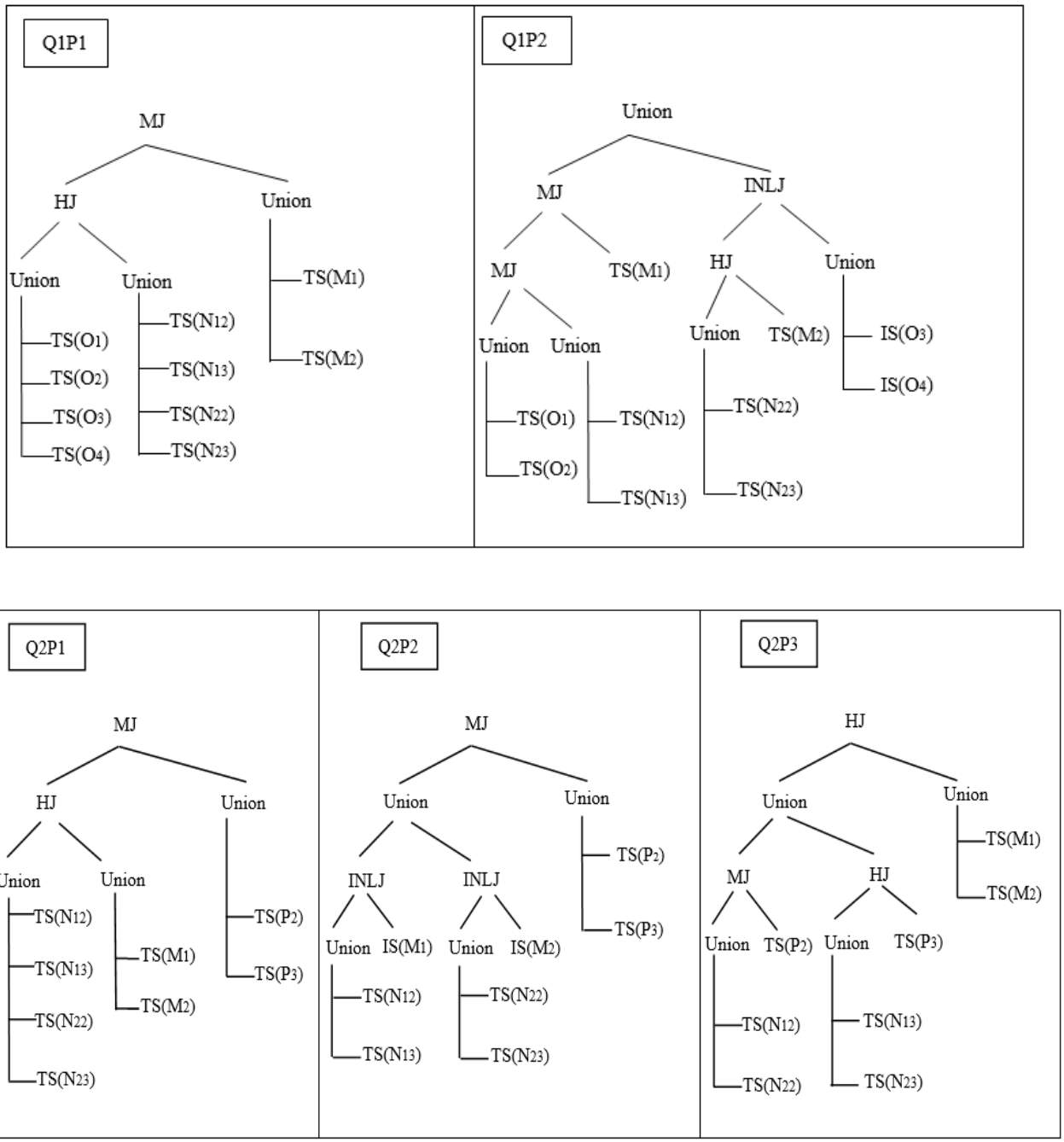


Fig. 5 Q1P1 and Q2P1 are plans generated by current optimizers for example queries Q1 and Q2 respectively. Q1P2, Q2P2, and Q2P3 are possible plans generated by the proposed optimizer. IS and TS are respectively index and tables can operators. HJ, MJ, and INLJ are respectively hash, merge, and index-nested-loop join operators. Union is a bag union operator

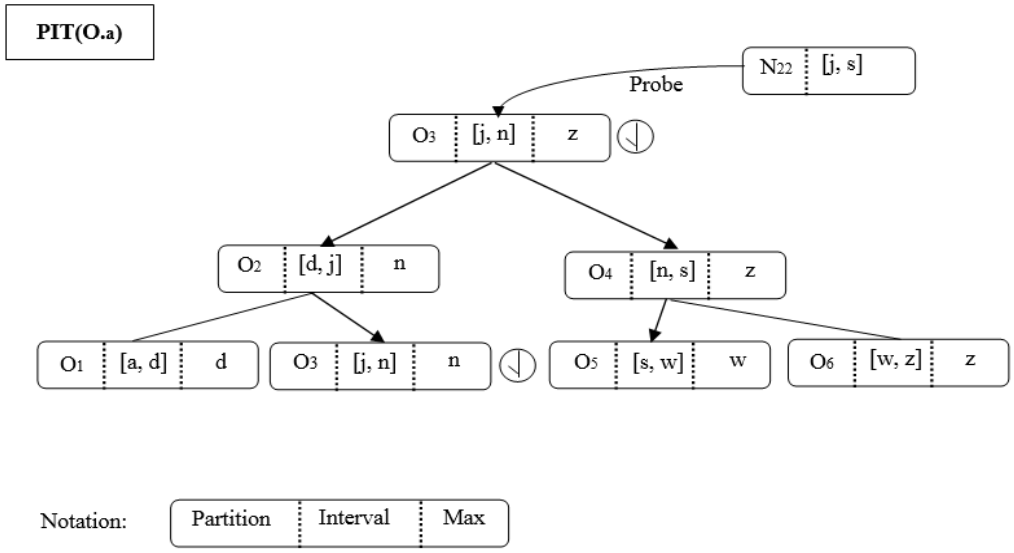


Fig. 6: A partition index tree containing intervals for all child tables (partitions) of O from Figure 4

D. Most PITs (partition index trees) are created once and then reused several times during the bottom-up optimization process. PIT, at a basic level, is considered as an increased red-black tree. The tree is organized by the Low values of the intervals, and an extra annotation is appended to every node registering the maximum High value across both its subtrees. Fig. 6 illustrates the PIT built on attribute O.a based on the partitioning conditions of all child tables of O (see Fig. 4).

The Low and Max values on each node are implemented during investigation to efficiently orient the search for finding the overlapping intervals. When the interval [j, s] is implemented to investigate the PIT, five intervals are checked (shown in Fig. 6) and the two overlapping intervals [j, n] and [n, s] are returned. A number of nontrivial enhancements to PITs were required to support complex partitioning conditions that may appear in practice. First, PITs require help for multiple types of intervals: open, closed, partially closed, one sided, and single values. Also, using non equi-joins need help from PITs to find all intervals in the tree that are to the left or to the right of the probe interval. Both partitioning and join conditions may provide complex combinations of AND and OR subexpressions, as well as including any other comparison operator.

IV. CASE STUDY AND EXPERIMENTAL RESULTS

During the experiments, the TCP-H benchmark dataset and its queries were adopted to obtain the results. The TCP-H dataset and queries were tested inside MapReduce and MapReduce using table partitioning. And, the results were used to compare the performance in each case. The following table (Table 1) shows the execution times including total MapReduce jobs of running TCP-H query number 2, 3, 7, 11, 13, 20 and 22 using the following: MapReduce and MapReduce using the proposed technique in this paper for table partitioning.

The results showed that running TCP-H query 2 using table partitioning using the proposed technique inside MapReduce takes 25 seconds and 150 milliseconds longer than running the original query without partitioning. TCP-H query 3 with table partitioning took approximately 12 minutes longer than without table partitioning. TCP-H query 7 with table partitioning took 17 minutes longer than without using table partitioning. TCP-H query 11 with table partitioning took around 10 seconds longer than without using table partitioning. TCP-H query 13 with table partitioning took around 6 minutes longer than without using table partitioning. Also, the results showed that TCP-H query 20 with table partitioning took about 8 minutes longer than without using table partitioning. Finally, the results showed that running TCP-H query 22 with table partitioning took 1 minute, 9 seconds and 900 milliseconds longer than running the original query without table partitioning. These results are due to that usually querying data with table partitioning takes longer than simply running ordinary queries that does not require table partitioning. Because, querying multiple tables takes longer than running a single table, but, the difference in execution times vary from one query to another; the difference between using table partitioning and without using table partitioning is small in some queries, such as, query 2, and in other cases, is big, such as in query 7, due to the query itself.

But, in general, table partitioning is better in cases when there is not sufficient spaces in some hard disks, because, table parts can be saved independently.

Table I Execution Times in Both Cases

	MR	MR Using Table Partitioning
Q2	2 min 52 sec 500 msec	3 min 16 sec 650 msec
Q3	2 min 40 sec 20 msec	14 min 27 sec 100 msec
Q7	31 sec 590 msec	17 min 46 sec 460 msec
Q11	57 sec 600 msec	1 min 7 sec 210 msec

Q13	2 min 34 sec 700 msec	8 min 14 sec 630 msec
Q20	54 sec 440 msec	8 min 45 sec 700 msec
Q22	6 min 5 sec 170 msec	7 min 15 sec 70 msec

The following figures show the execution times of query 2, 3, 7, 11, 13, 20 and 22. Fig. 7, Fig. 8, Fig. 9, Fig. 10, Fig. 11, Fig. 12 and Fig. 13 show the execution times of queries 2, 3, 7, 11, 13, 20 and 22 with and without table partitioning. These figures are derived from Table 1, which shows the execution times for both cases using the proposed table partitioning technique and without using table partitioning. In these figures, the total execution time is measured in seconds, and is represented by the x-axis, while the MR with and without the proposed table partitioning technique is represented by the y-axis.

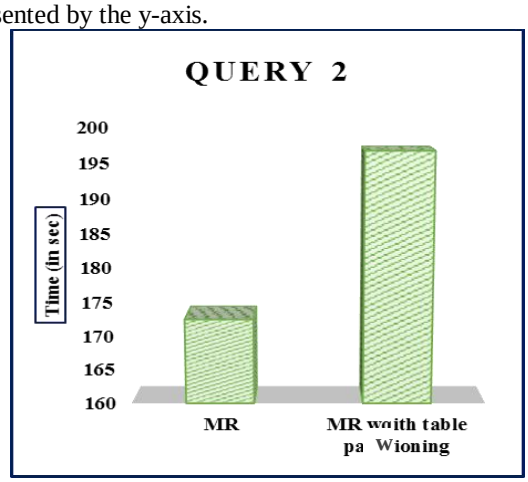


Fig. 7 shows execution times of query 2 with and without table partitioning

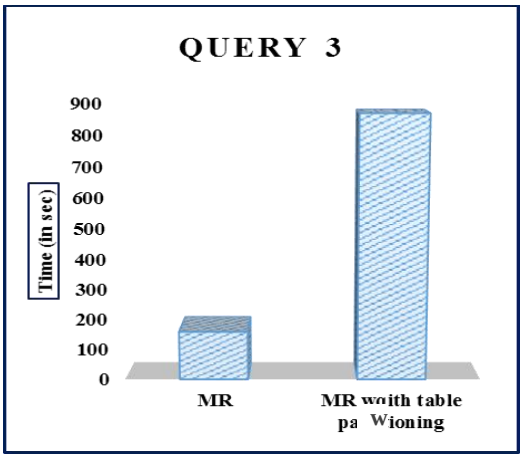


Fig. 8 shows execution times of query 3 with and without table partitioning

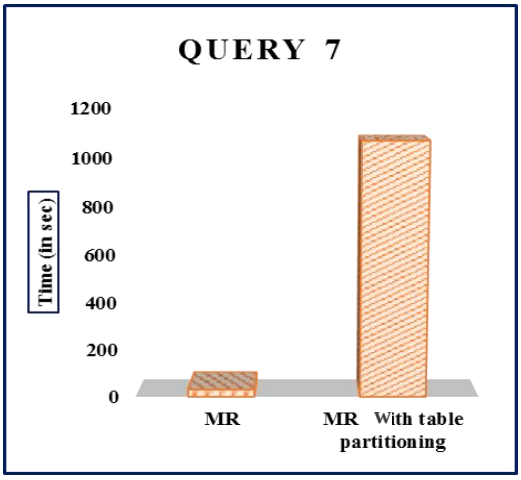


Fig. 9 shows execution times of query 7 with and without table partitioning

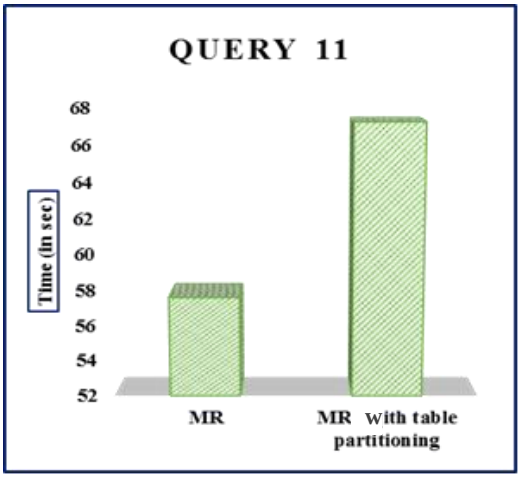


Fig. 10 shows execution times of query 11 with and without table partitioning

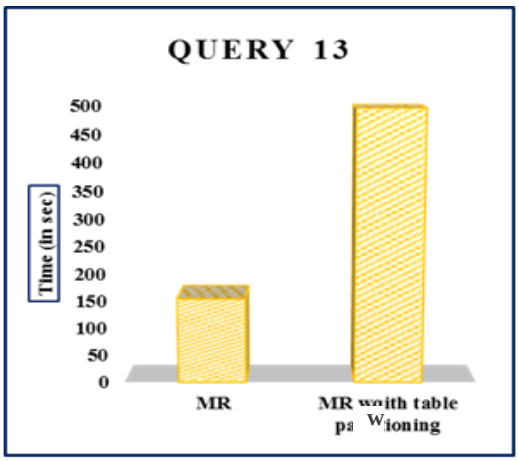


Fig. 11 shows execution times of query 13 with and without table partitioning

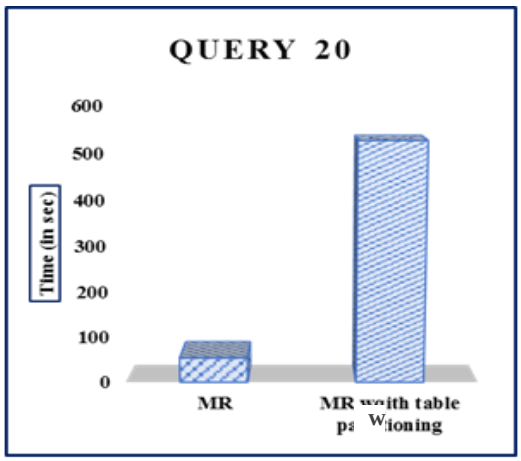


Fig. 12 shows execution times of query 20 with and without table partitioning

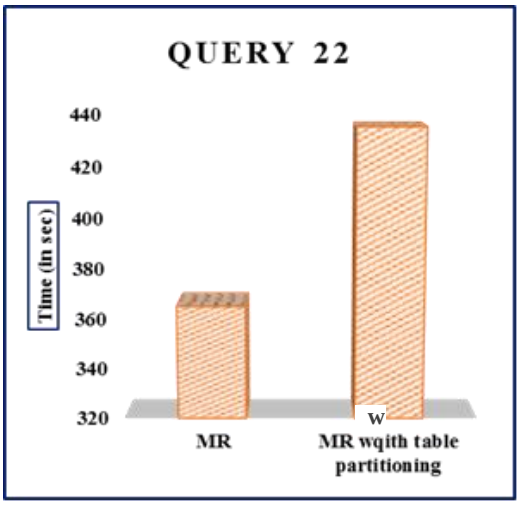


Fig. 13 shows execution times of query 22 with and without table partitioning

V. CONCLUSIONS AND FUTURE WORK

In this paper a technique for table partitioning is developed, different case studies and experiments are conducted in this paper to compared table partitioning using the proposed technique and original TCP-H query. Running queries using table partitioning has been proven to have good execution times compared to running queries without table partitioning, but using table partitioning has important benefits of enabling large databases to be distributed on different storages when there are not sufficient storage spaces for them. In the future, other table partitioning techniques can be applied and tested inside MapReduce environment to select the best technique.

REFERENCES

[1] *Advancing Discovery in Science and Engineering*. Computing Community Consortium. Spring 2011.
[2] *Advancing Personalized Education*. Computing Community Consortium. Spring 2011.
[3] Will Dobbie, Roland G. Fryer, Jr. Nber. *Getting Beneath the Veil of Effective Schools: Evidence from New York City*. Working Paper No. 17632. Issued Dec. 2011.
[4] *Smart Health and Wellbeing*. Computing Community Consortium. Spring 2011.
[5] (2007) IBM DB2. Partitioned tables. [Online]. Available: http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com. ibm.db2.luw.admin.partition.doc/doc/c0021560.html.
[6] T. Morales. (2007). *Oracle(R) Database VLDB and Partitioning Guide 11g Release 1 (11.1)*. Oracle Corporation. [Online]. Available: http://download-uk. oracle.com/docs/cd/B28359_01/server.111/b32024/toc.htm.
[7] (2003) Sybase, Inc. Performance and Tuning: Optimizer and Abstract Plans. [Online]. Available: http://infocenter.sybase.com/help/topic/com.sybase.dc20023\ _1251/pdf/optimizer.pdf.
[8] R. Talmage. *Partitioned Table and Index Strategies Using SQL Server 2008*. Microsoft, 2009.
[9] E. Friedman, P. Pawlowski, and J. Cieslewicz. *SQL/MapReduce: A Practical Approach to Self-Describing, Polymorphic, and Parallelizable User-Defined Functions*. In VLDB, 2009.
[10] S.N.Srirama, P.Jakovits, and E.Vainikko, *"Adapting scientific computing problems to clouds using MapReduce"*, Future Generation Computer Systems, Vol. 28, No. 1, pp184-192, 2011.
[11] J.Shafer, S. Rixner, and A.L. Cox, *"The Hadoop Distributed File system: Balancing Portability and Performance"*, IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS), ISBN: 978-1-4244-6023-6, 2010, pp122-133.
[12] K.Talattinis, A Sidiropoulou, K.Chalkias, and G.Stephanides, *"Parallel Collection of Live Data Using Hadoop"*, IEEE 14th PanHellenic Conference on Informatics (PCI), ISBN: 978-1-42447838-5, 2010, pp66-71.
[13] (February 2012) Hadoop MapReduce. [Online]. Available: http://wiki.apache.org/hadoop/MapReduce
[14] C. Olston, B. Reed, et al. *Pig latin: a not-so-foreign language for data processing*. In SIGMOD, 1099–1110, 2008.
[15] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 2009.
[16] J. Dean and S. Ghemawat. *MapReduce: simplified data processing on large clusters*. In OSDI, 10–10, 2004.
[17] T. Condie, N. Conway, et al. *MapReduce online*. In NSDI, 2010.

