International Journal of Grid and Distributed Computing
Vol. 9, No. 5 (2016), pp.229-240
http://dx.doi.org/10.14257/ijgdc.2016.9.5.20

Comparative Study of Multi-query Optimization Techniques using Shared Predicate-based for Big Data

Radhya Sahal[1], Mohamed H. Khafagy[2] and Fatma A. Omara[3]

[1,3] Department of Computer Science, Faculty of Computers & Information, Cairo University, Egypt
[2] Department of Computer Science, Faculty of Computers & Information, Fayoum University, Egypt
[1]radhya.sahal@grad.fci-cu.edu.eg, [2]mhk00@fayoum.edu.eg
[3]f.omara@fci-cu.edu.eg

Abstract

Big data analytical systems, such as MapReduce, have become main issues for many enterprises and research groups. Currently, multi-query which translated into MapReduce jobs is submitted repeatedly with similar tasks. So, exploiting these similar tasks can offer possibilities to avoid repeated computations of MapReduce jobs. Therefore, many researches have addressed the sharing opportunity to optimize multi-query processing. Consequently, the main goal of this work is to study and compare comprehensively two existed sharing opportunity techniques using predicate-based filters; MRShare and relaxed MRShare. The comparative study has been performed over TPC-H benchmark and confirmed that the relaxed MRShare technique significantly outperforms the MRShare for shared data in terms of predicate-based filters among multi-query.

Keywords*: Big data, MapReduce, Sharing Opportunity, Multi-Query Optimization, filter, predicates*

1. Introduction

Big Data has been grown rapidly in many domains such as social networks and other information systems. The need for distributed computing is become an important issue due to the increasing of workstations power and the data sets sizes. The development and implementation of distributed system for Big Data applications are considered a challenge [1-3]. One of the popular frameworks that have emerged for Big Data processing is MapReduce. It was first introduced by Google in 2004 [4]. The main concept of MapReduce is to abstract the details of a large cluster of machines to facilitate the computation on large datasets.

On the other hand, Multiple Query Optimization (MQO) problem has been introduced in the 1980s which considered as well-known database research problem [5]. It dominates in many business applications, artificial intelligent and many research areas in relational databases. Regarding single query in database optimization era, the query optimizer selects the cheaper costly execution plan (*i.e.*, a method to get correctly exact answer for query) [6] . By extending the query to be multiple queries, the problem denotes as how to define the optimal execution plan with a minimum cost which can answer all the entire queries and improve the overall time of execution plans. It aims to identify the shared common subexpressions (CSEs) among queries and exploit them to reduce the query evaluation cost [6].

Fundamentally, MapReduce system is one of the most popular parallel processing systems in Big Data. However, the parallel processing systems consume an enormous

ISSN: 2005-4262 IJGDC

amount of resources to process large data size. The optimization of data processing systems especially redundant computation tasks can efficiently improve the performance, as well as, the resource utilization. On the other words, sharing similar tasks can reduce the overall amount cost of computation, which can contribute to reducing incurred financial charges while utilizing the processing infrastructure [7]. Many applications often involve complex multiple shared queries which share a lot CSEs. Identifying and exploiting the CSEs to improve query performance is essential for these applications. Multi-query optimizations, which aims to detect and exploit the CSEs among queries in order to reduce the overall query evaluation cost, has been extensively studied for over two decades and demonstrated to be an effective technique in both Relational Database Management Systems and MapReduce contexts [7-13].

Principally, MRShare is considered the state-of-the-art for improving the performance of multi-query optimization using MapReduce [7]. On the other hand, the relaxed MRShare introduces more comprehensive with additional optimization techniques [9]. Both the MRShare and relaxed MRShare are proposed for computational aggregated multi-query optimization. However, the shared predicate-based filters for large-scale data analytics in MapReduce can also benefit to exploit the sharing. Consequently, the goal of this paper is to study and compare two existed works; the MRShare and relaxed MRShare techniques [7, 9]. The aim of this study is to investigate that not only the shared computation can gain optimization in Big Data environment, but also the multi-query optimization for shared data using predicate-based filters can open up new opportunities to gain significant improvement by reducing redundant filtering tasks.

The rest of this paper is organized as follows. Related work is described in Section 2. The multi-query optimization in MapReduce is introduced in Section 3. The comparative techniques for multi-query optimization MapReduce based are described in Section 4. The predicate-based query filter concept with an illustrative example of comparative techniques are described Section 5. The analysis of comparative techniques is presented in Section 6. Finally, conclusions are presented in Section 7.

2. Related Work

MapReduce is one of the most popular parallel processing platforms. There are several works have been done in query optimization MapReduce-based. One of the preliminary work concerns concurrent batch exaction queries MapReduce is MRShare [7]. MRShare is a concurrent sharing framework. However, the proposed work in [9] has relaxed and generalized MRShare overlapping queries to increase the sharing opportunities in a single job. Moreover, they have proposed a novel cost-based two-phase approach to finding optimal evaluation plans.

ReStore is a non-concurrent sharing system built on top of Pig [8]. It optimizes query evaluation using materialized results. According to a space budget for storing materialized results, ReStore uses heuristics algorithm to choose the suitable materialized results even the complete or part of the map and/or reduce the output of each job. The materialized output which is produced by ReStore might not be reused at all if the query workloads are not repeated again. Thus, the concurrent shared queries need more investigation to overcome storing unused results.

More recent works in [12, 14] consider reusing results stored by exploiting MapReduce intermediate results for failure resilience reasons as materialized views. In particular, semantic UDF models based on Hive has been used to enable effectively reuse views where subsequent queries can be evaluated faster [12]. On the other side, a multi-query optimization framework, SharedHive, is proposed to transform a set of correlated HiveQL into new optimized queries sets with respect to sharing scan and computation tasks [10]. Because Pig is considered the popular language within the data management in the parallel processing of large data volumes, reused-base optimization has been

addressed for Pig scripts [15]. According to the work in [15], PigReuse has been proposed to identify and reuse common subexpressions occurring in Pig Latin scripts and select the best ones to be merged based on a cost-based search process which is implemented with the help of a linear program solver.

Indeed, both MRShare and relaxed MRShare are used to optimize aggregation queries. Regarding this paper, the MRShare and relaxed MRShare are chosen to compare the evaluation of the multi-query optimization for shared predicate-based filters which have a same key idea and applied for shared computation optimization.

3. Multi-Query Optimization in MapReduce

In this section, the overview of multi-query optimization will be introduced. Then, the MapReduce query processing will be described which is considered the backbone of the comparable techniques in this work. In addition, the general differences between multi-query optimization in Database Management Systems (DBMS) and MapReduce will be discussed.

3.1 Overview of Multi-Query Optimization

Multiple Query Optimization (MQO) is a well-known problem in database research. It describes how to efficiently produce answers to a set of queries with common tasks. By given a set of queries, each query has a set of alternative execution plans and each plan has a set of tasks. MQO aims to choose an appropriate plan for each query and minimize the total execution time by performing common tasks only once [16, 17].

MQO is considered as an NP-Hard optimization problem where their algorithms such as heuristics and genetic algorithms are mostly approximate or produce near optimal solution. Since these solutions for MQO problem should find the set of plans that produce the answer of each query and included in the minimum cost global execution plan (*i.e.*, a set of plans that answer every and each one of a set of queries) [6]. Whereas, many optimization solutions have been introduced to find the optimal plan based on materialized results (*i.e.*, intermediate results and final answer) which produced from earlier queries [17, 18]. Figure 1 depicts the three basic multi-query optimization techniques listed as follows:

1) **Naive technique**; each query runs isolated from other queries.
2) **Grouping technique;** only one instance of the query will be run for a set of identical queries.
3) **Materialization technique**; when some queries materialized their results or part of their results to other queries.

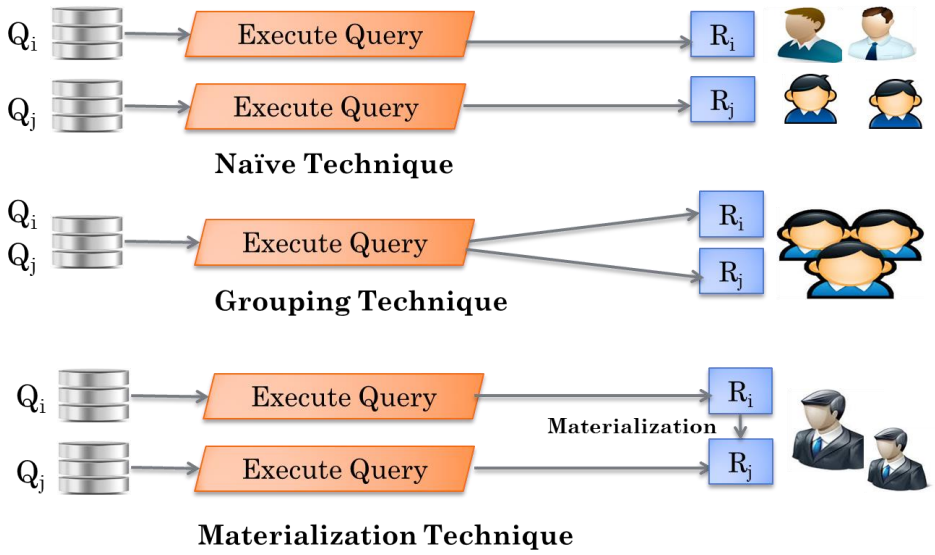

Figure 1. Multi-Query Optimization Techniques

3.2 MapReduce Query Processing

Besides efficiency, MapReduce provides two simple user-friendly interfaces; map and reduce function. Furthermore, MapReduce supports query processing by integrating with high-level declarative languages such as Hive [19, 20] and Pig [21] to simplify application programming. These MapReduce-based query languages hide the implementation details (*e.g.*, access methods, query plan optimization) and offer an SQL-like interface for developers and Big Data analysts. However, the optimizer of MapReduce-based query languages suffers from the high cost of saving intermediate results. For more details, each MapReduce job is flushed back to HDFS as a backup for fault tolerance. The next MapReduce job reads the intermediate results of the previous job to continue processing. The HDFS I/O cost is significantly higher than local storage (*i.e.*, there is Network cost) that use load balance [22, 23]. So, exploiting shared jobs can reduce intermediate results, and can be cheaper than generating too large size of intermediate results in the case of using an original data source for each query separately [24, 25].

3.3 Difference between Multi-Query Optimization on DBMS and Map Reduce

There are several works have been done for optimizing query processing based on MapReduce jobs. Multi-query optimization MapReduce-based is significantly different from multi-query optimization in DBMS in the following three points; 1) MapReduce is based on a block operator, with all internal results of map phase required to be physically materialized to reduce phase; 2) Parallel scanning processing is the main MapReduce strategy and no pipeline is supported; and 3) Shuffling phase which concerns of data transferring between mappers and reducers [24]. Therefore, there are differences of multi-query optimization solutions for pruning bad plans in DBMS and MapReduce which considered the popular processing analytical system in Big Data era. Table 1 identifies a brief of comparison between multi-query optimization in DBMS and multi-query optimization MapReduce-based with respect to two different optimization types; non-concurrent optimization and concurrent optimization [25].

Table 1. Comparison between Multi-Query Optimization in DBMS and Map Reduce

Optimization Type	DBMS	MapReduce
Non-concurrent	Use Materialized Views.	Translates the input multi-query into jobs. Uses the output from past sub jobs such as map output, reduce input or combined with them.
	Matcher: matches all candidate views and the query optimizer select the best one to optimize the global plan.	**Matcher**: searches in the previous executed sub-jobs or jobs that can be used to answer all or part of new jobs.
	Rewriter generates the new expression using the enumerated views to output the original query by adding operators, predicates, and indices.	**Rewriter** rewrites the input jobs using the enumerated reused jobs outputs which stored previously.
Concurrent	Identifies the common subqueries and executes them only once.	Identifies the sharing tasks or sub-jobs and grouping them for running only once. Due, the optimization in MapReduce had different optimization levels such as shared scan and shared shuffling, the concurrent optimization causes extra I/O cost for large file size especially in the case of low I/O speed Hadoop environment.

4. Comparative Techniques of Multi-Query Optimization on MapReduce Environment

The aim of the work in this paper is to compare the performance evaluation of the two existed techniques of multi-query optimization MapReduce-based for shared computation; MRShare and Relaxed MRShare. These two techniques will be discussed in details.

4.1 MRShare

MRShare is considered one of the preliminary work concerning concurrent batch execution queries in MapReduce [7]. MRShare is a concurrent sharing framework which can share portions of identical work to avoid redundant computation. It considers the following sharing opportunities:

1) **Sharing Scans;** for the same map tasks against the same input file, the input data would be scanned only once.
2) **Sharing Map Outputs;** for the key-values pairs of the same mapping tasks, only one sorted and transferred tasks would be performed.
3) **Sharing Map Functions;** it is similar to multi-query optimization and avoids redundancy computation for a batch of queries executed at the same time.

The key idea of MRShare is that sets of submitted jobs are transformed into groups where each group is considered as a single job. Furthermore, the selection of each group is solved as an optimization problem with the objective of maximizing the total savings. More specifically, MRShare can process a group of jobs as a single job by applying tagging process. For shared scans, map output tuples are tagged with respect to the original individual jobs, then the multiple output files are written on the reduce side [7, 9, 25].

4.2 Relaxed MRShare

Although, MRShare is considered as the state-of-the-art work in multi-query optimization MapReduce-based, there is a set of research works have been proposed to extend MRShare. However, the proposed work in [9] has relaxed and generalized MRShare is overlapping queries to increase sharing opportunities in a single job. Compared with MRShare, they have introduced more comprehensive with additional optimization techniques (*i.e.*, Generalized Grouping Technique and Materialization Technique). Indeed, MRShare's grouping technique can only share map input of similar jobs. Since it considers that the two jobs produce entirely different map output even there is some sort of overlapping among these jobs. To illustrate the difference between MRShare and relaxed MRShare techniques, consider the following example of two jobs/queries [26]:

J_1: ***select*** *a, b, c* ***from*** *T* ***where*** $a \leq 10$
J_2: ***select*** *a, b, c* ***from*** *T* ***where*** $a \geq 5$

It may be noted that the map output of J_2 for $5 \leq a \leq 10$ can be reused to derive the partial map output of J_1. Therefore, MRShare's grouping technique suffers from the limitation of exploiting the sharing opportunities among multiple jobs. The relaxed MRShare presents a more comprehensive study of multi-query/job optimization techniques to share map input scan and map output and algorithms to choose an evaluation plan for a batch of jobs in the MapReduce context [9, 26]. Therefore, the relaxed MRShare concerns about multi-query optimization with respect to the aggregated queries. The comparative study through this work focuses on predicate-based filtering to show that the overlapping consideration among multi-query is also beneficial in case of plain queries and can reduce redundant filtering tasks.

5. Predicate-Based Filters on Shared Data

In this section, the overview of predicate based filters which are applied to shared data will be introduced followed by the used example of the comparative techniques.

5.1 Predicate-Based Filters

Ultimately, querying large data could produce some overheads, as I/O bottleneck and network traffic. Therefore, the performance of the system would be degraded. So data pruning is an important part of query processing which can drastically cut down the time spent on analyzing data and then improves the system performance. On the other hands, predicates filtering have been applied in the traditional database systems which can cause significant cost saving for data-intensive applications. In more details, when predicates with a high filter factor are processed, the unmatched rows are assessed and eliminated as early as possible, which can reduce processing cost. Worthwhile, different types of filters can be defined according to SQL language as; i) comparison operators such as logic operations, ii) range filters such as between, and iii) exact matching such as specific value and list values using exists, in, like, all, some and any. According to the context of this

paper, the predicate-based range filters are concerned. Therefore, these predicates can also benefit from shared multi-query in the case of applying for only once.

The shared multi-query can be broadly classified into two categories; shared data and shared computation. The multi-query optimization MapReduce-based on shared computation has been explored in different research works [7, 9, 10, 12]. However, the shared data which pruned by predicate-based filters can also benefit to exploit the sharing opportunity. On the other hand, a predicate defines a logical condition being applied to rows in a table. Regarding multi-query, the common predicate-based filters perform redundant filtering-rows tasks against the large input files. Therefore, grouping these predicate-based filters can avoid redundant and wasteful filtering tasks on the same input files.

5.2 Example of Comparative Techniques

An illustrative example to clarify the difference between MRShare and relaxed MRShare using a predicate-based filter is presented in this section. Eight queries are considered to retrieve data from different input files (*i.e.*, tables) as described in Table 2. For simplicity, assumed that, the filters have ten distinct values ranged from 1 to 10.

Table 2. Example of Input Multi-Query

Query ID	Query
Q1	Select a, b, c from T where $1 \leq a \leq 3$
Q2	Select *b, c, d* from *T* where $4 \leq b \leq 6$
Q3	Select a, b, c from t where $1 \leq a \leq 3$
Q4	Select *b, c, d* from *T* where $4 \leq b \leq 6$
Q5	Select *a, e, f* from *R* where $7 \leq a \leq 8$
Q6	Select *b, d* from *S* where $8 \leq b \leq 9$
Q7	Select *b, d* from *T* where $4 \leq b \leq 5$
Q8	Select *a, c* from *T* where $2 \leq a \leq 3$

Regarding the shared filters, the MRShare technique can merge both similar queries Q_1 and Q_3 in the same group, as well as, combine the similar queries Q_2 and Q_4 in another group (see Figure 2). Each group will execute one job, which applies the same filter on the same input file only once then produces the same output for both queries within the one group. The reset queries are detected as non-shared by MRShare technique.

On the other hand, the relaxed MRShare technique can detect large sharing opportunities by considering the overlapping among multi-query. Thus, the number of shared queries within the same group is increased which can eliminate the redundant scanning, as well as, filtering tasks. Therefore, the total multi-query optimization will be improved by reducing the fraction of the total processing time which incurred without overlapping considerations.

Again here, regarding the previous example, the relaxed MRShare can merge queries Q_1, Q_3, and Q_8 in the same group, and the queries Q_2, Q_4 and Q_7 in another group (see Figure 3). More specifically, the relaxed MRShare considers the overlapping between Q_1, Q_3, and Q_8 regarding their shared predicates filters such as $1 \leq 2 \leq 3$. Similarly, the same overlapping consideration is taken with respect to Q_2, Q_4, and Q_7 regarding their shared predicates filters such as $4 \leq 5 \leq 6$. Finally, each group will execute one job, which applies the same filter on the same input file only once, benefits from overlapping

predicted-based filter, then produces the desired output for all queries within the one group. Conversely, Q_5 and Q_6 are querying on different input files, so each query will be executed independently even the input multi-query are evaluated by MRShare or relaxed MRShare technique.

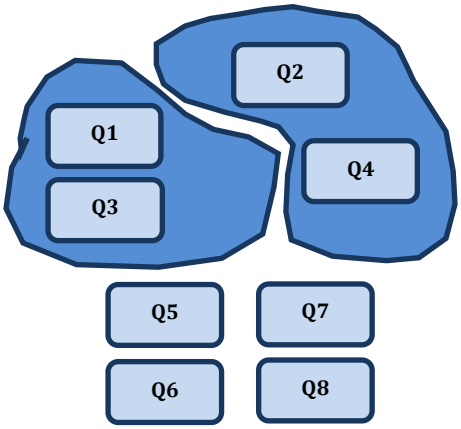

Figure 2. MRShare Multi-Query Optimization Technique

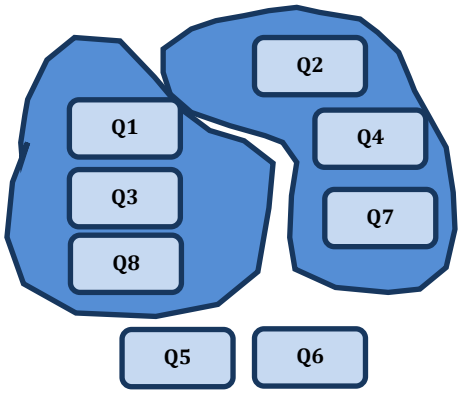

Figure 3. Relaxed MRShare Multi-Query Optimization Technique

6. Analysis of the Comparative Techniques

In this section, the experimental evaluation of the comparative study is presented. Starting by describing the experiment setup and used metrics.

6.1 Experiment Setup

The experiments have been performed using Hadoop version 2.6.0 on a cluster of 10 nodes configured with a 3 GB of RAM, 2 cores, and 200 GB disk and run Ubuntu Linux 12.04.2 LTS.

6.2 Datasets and Queries

The structured data is used through the experiments by generating different files size of Star Schema Benchmark. Star schema is based on TPC-H benchmark, and it is designed to measure the performance of database products that support data warehouse applications [27]. Lineitem table is used and *l_discount* attribute values (*i.e.*, range attribute is used as

the selection attribute which stores some disjoint of distinct values) for predicates within filters. A set of synthetic queries are used which generated from the following query template:

SELECT *list of attributes* FROM *Lineitem* WHERE $a \leq l\_discount \leq b$

6.3 Performance Evaluation

In this sub section, the comparative study is evaluated by data size in terms of a different number of tuples [28]. Two metrics is measured; execution time of queries which means the elapsed time duration from query submission to query completion and the reduction of filtered data. More specifically, the filtered data indicates the tuples which could be filtered to answer the query. The multi-query execution plans that have been evaluated and compared are (1) the MRShare plan and (2) the relaxed MRShare plan. Moreover, the number of queries which used is 20 queries for all conducted experiments.

6.3.1 Effect of Data Size

The performance as a function of data size will be examined in the following experiments. Figure 4, 5, 6 and 7 depicts that the relaxed MRShare technique significantly outperforms the MRShare technique for 100 million, 250 million, 500 million and 1 billion tuples respectively.

Furthermore, Figure 8 shows the performance improvement of the relaxed MRShare technique with respect to MRShare technique that is 46%, 62%, 70% and 76% against 100 million 250 million, 500 million and 1 billion respectively. It depicts that the performance improvement of the relaxed MRShare technique increases linearly with respect to data size (*i.e.*, the number of tuples). The reason behind the improvement of the relaxed MRShare technique multi-query execution time is that the increasing of the data sharing in terms of overlapping consideration, which in turn reduces the loading data from HDFS, as well as, filtering operations compared to the loading and filtering of the MRShare technique.

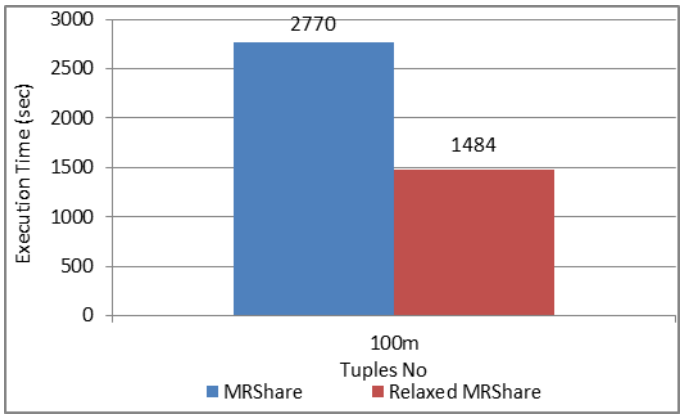

Figure 4. The Execution Time of 100 Million Tuples

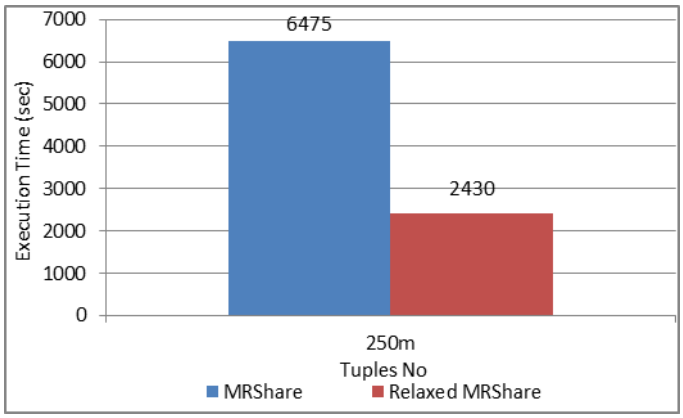

Figure 5. The Execution Time of 250 Million Tuples

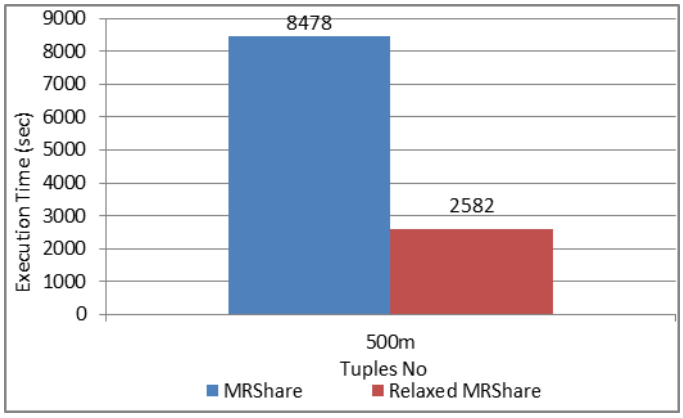

Figure 6. The execution time of 500 million tuples

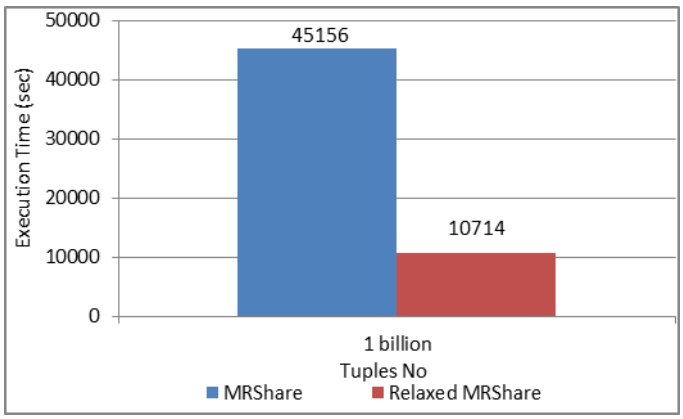

Figure 7. The Execution Time of 1 Billion Tuples

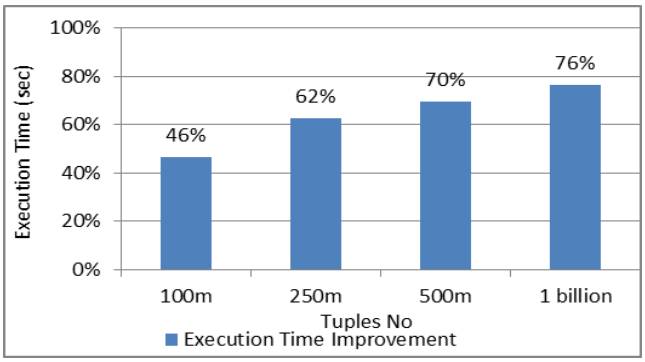

Figure 8. Execution Time Improvement w.r.t. MRShare Technique of 100, 250,500 Million and 1 Billion

6.3.2 Effect of Filtered Data: On the other hand, Figure 9 depicts the number of filtered tuples for both the MRShare and the relaxed MRShare multi-query execution plan using 100 million, 250 million, 500 million and 1 billion tuples. It is noted that the relaxed MRShare technique can reduce the number of filtered tuples significantly with respect to MRShare technique. Also, the improvement of filtered tuples reduction in the relaxed MRShare technique with respect MRShare technique is 75% which gained by overlapping consideration.

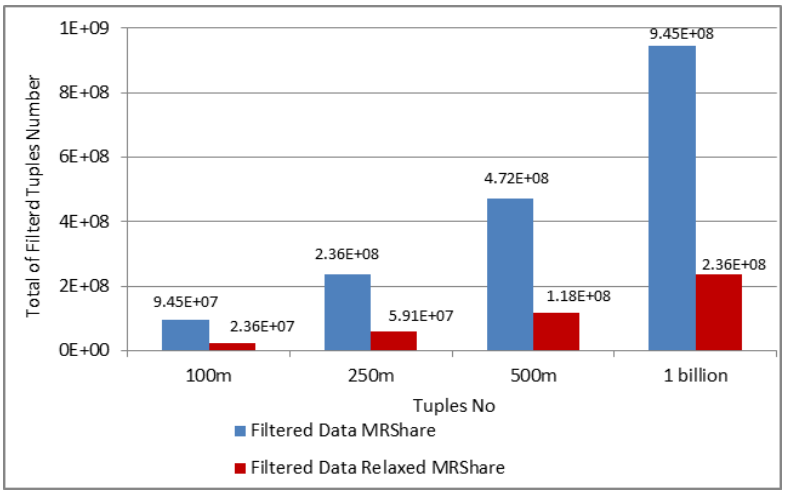

Figure 9. Improvement of Filtered Tuples Reduction using 100, 250, 500 Million and 1 Billion Tuples for Ten Queries

7. Conclusions

The work in this paper aims to compare the sharing data opportunity to optimize multi-query in Big Data environment. Two existed techniques are compared; the MRShare and the relaxed MRShare using predicate-based filter among multi-query. The experimental results show that the execution time of the relaxed MRShare technique is improved comparing to the MRShare technique. Meanwhile, the performance of multi-query will be enhanced by increasing data size and number of shared queries such as increasing a portion of overlapping through multi-query. On the other hand, the reduction of filtered data of the relaxed MRShare technique is improved with respect to MRShare technique. Therefore, the comparative study using predicate-based filter proves that the exploiting of shared data is also beneficial, as well as, the shared computation for multi-query optimization MapReduce-based by reducing redundant filtering tasks.

References

[1] R. Akerkar, *Big data computing*: CRC Press, 2013.
[2] A. Gkoulalas-Divanis and A. Labbi, *Large-Scale Data Analytics*: Springer, 2014.
[3] R. Sahal, M. H. Khafagy, and F. A. Omara, "A Survey on SLA Management for Cloud Computing and Cloud-Hosted Big Data Analytic Applications," *International Journal of Database Theory and Application,* vol. 9, pp. 107-118, 2016.
[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, pp. 107-113, 2008.
[5] M. A. Bayir, I. H. Toroslu, and A. Cosar, "Genetic algorithm for the multiple-query optimization problem," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* vol. 37, pp. 147-153, 2007.
[6] J. F. García, "MQOP-A Tiny Reference to the Multiple-Query Optimization Problem," *Revista de Tecnología,* vol. 7, 2008.
[7] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas, "MRShare: sharing across multiple queries in MapReduce," *Proceedings of the VLDB Endowment,* vol. 3, pp. 494-505, 2010.
[8] I. Elghandour and A. Aboulnaga, "Restore: Reusing results of mapreduce jobs," *Proceedings of the VLDB Endowment,* vol. 5, pp. 586-597, 2012.
[9] G. Wang and C.-Y. Chan, "Multi-query optimization in mapreduce framework," *Proceedings of the VLDB Endowment,* vol. 7, pp. 145-156, 2013.
[10] T. Dokeroglu, S. Ozal, M. A. Bayir, M. S. Cinar, and A. Cosar, "Improving the performance of Hadoop Hive by sharing scan and computation tasks," *Journal of Cloud Computing,* vol. 3, pp. 1-11, 2014.
[11] M. N. Abdullah, M. H. Khafagy, and F. A. Omara, "HOME: HiveQL Optimization in Multi-Session Environment," in *Proceedings of the 5th European Conference of Computer Science (ECCS14)*, 2014, pp. PP. 80-89.
[12] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis, and M. J. Carey, "Opportunistic physical design for big data analytics," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 851-862.
[13] H. Yao, J. Xu, Z. Luo, and D. Zeng, "MEMoMR: Accelerate MapReduce via reuse of intermediate results," *Concurrency and Computation: Practice and Experience*, 2015.
[14] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis, and M. J. Carey, "MISO: souping up big data query processing with a multistore system," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 1591-1602.
[15] J. Camacho-Rodríguez, D. Colazzo, M. Herschel, I. Manolescu, and S. R. Chowdhury, "Reuse-based Optimization for Pig Latin," in *BDA'2014: 30e journées Bases de Données Avancées*, 2014.
[16] J. Grant and J. Minker, "On optimizing the evaluation of a set of expressions," *International Journal of Computer & Information Sciences,* vol. 11, pp. 179-191, 1982.
[17] S. Finkelstein, "Common expression analysis in database applications," in *Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, 1982, pp. 235-245.
[18] T. K. Sellis, "Multiple-query optimization," *ACM Transactions on Database Systems (TODS),* vol. 13, pp. 23-52, 1988.
[19] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment,* vol. 2, pp. 1626-1629, 2009.
[20] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive-a petabyte scale data warehouse using hadoop," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, 2010, pp. 996-1005.
[21] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1099-1110.
[22] E. Sarhan, A. Ghalwash, and M. Khafagy, "Queue weighting load-balancing technique for database replication in dynamic content web sites," in *Proceedings of the 9th WSEAS International Conference on APPLIED COMPUTER SCIENCE*, 2009, pp. 50-55.
[23] H. A. Hefny and M. H. Khafagy, "Comparative Study Load Balance Algorithms for Map Reduce Environment," *International Journal of Computer Applications,* vol. 106, pp. 41-50, 2014.
[24] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi, "Query optimization for massively parallel data processing," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, p. 12.
[25] C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in MapReduce," *The VLDB Journal,* vol. 23, pp. 355-380, 2014.
[26] W. Guoping, "Optimization Techniques for Complex Multi-query Applications," 2014.
[27] T. P. P. Council, "TPC-H benchmark specification," *Published at http://www.tcp.org/hspec.htm,* 2008.
[28] Y. Shi, "Finding Useful Information for Big Data," *International Journal of Grid and Distributed Computing,* vol. 8, pp. 11-22, 2015.

