REDUCING DATA SHUFFLING AND IMPROVING MAP REDUCE PERFORMANCE USING ENHANCED DATA LOCALITY

[1]AHMED ABDEL-RAHIM EL-DOUH, [2]HATEM MOHAMED ELKADI, [3]MOHAMED HELMY KHAFAGY

[1]CAIRO UNIVERSITY, INFORMATION SYSTEM DEPARTMENT, 12585 OCTOBER, EGYPT.
[2] CAIRO UNIVERSITY, INFORMATION SYSTEM DEPARTMENT, 12613 CAIRO, EGYPT.
[3] FAYOUM UNIVERSITY, COMPUTER SCIENCE SYSTEM DEPARTMENT, FAYOUM, EGYPT.
E-mail: [1]Ahmed.eldouh.csis@o6u.edu.eg, [2]hkadi@fci-cu.edu.eg, [3]mhk00@fayoum.edu.eg

Abstract – Map Reduce is the most used solution depending on the parallel processing to handle big data. Hadoop is an open source software implementation of the MapReduce concept. The shuffling phase in Map Reduce increases the overhead on performance. In this work, we focus on an approach to improve the performance of Map Reduce by reducing the overhead caused by the shuffling phase. Improving the locality of data will lead to eliminating the network overhead in the shuffling phase for the MapReduce. We can achieve this by pre-partitioning data based on the query-based similarity through TF-IDF, Cosine similarity algorithms and grouping the related queries with each other using a K-means clustering algorithm. In this regard, we support HDFS with the related data and control where data are stored to collocate the related data files in the same nodes. In this paper, we present our new approach called the enhance locality and reduce shuffling (ELRS) technique to improve specific aspects of MapReduce such as fast processing. thus, reducing the execution time. The experimental results obtained by the implementation of ELRS technique showed, improve data locality by around 27.2 % and a reduction in the execution time of Hadoop jobs by around 40.1%.

Keywords – Map Reduce, Hadoop, Shuffling, Locality, Similarity, Big Data.

I. INTRODUCTION

Nowadays, the data generations from various domains led to huge data amounts. So, it was necessary that solutions be found to solve the main challenges of big data techniques, such as processing, storage, data access, and security, etc. MapReduce is the programming model to handle big data initiated by Google's Team. Hadoop is an open source software that implements the MapReduce concept. In this contribution, we focus on the design of a suitable approach to adapt the input file based on query-based similarity. This can be performed efficiently to optimize queries by grouping the related queries that fetch the same data and put it with each other in the same nods. [1, 2, and 3]. In this regard, this approach tries to improve performance by eliminating the overhead caused by the shuffling phase and achieving data locality to reach higher performance in MapReduce. We propose we present our new approach to improve specific aspects of MapReduce, such as faster processing to reduce the execution time of MapReduce jobs. In summary, this paper presents the following contributions:

•A method to calculate the similarity between queries using Cosine algorithm and assign weights using Term Frequency-Inverse Document Frequency (TF-IDF) algorithms.
• A method to partition the related queries into clusters using the K-means algorithm.
• Adaptation ofHadoop Distributed File System(HDFS) to control where data are stored to co-locate the related data files in the same nodes to enhance the data localization.

1.1. Materials and Procedures

Clustering is a task that groups the related queries with each other for more than one purpose such as speeding up search and partitioning. In this paper, we recur to the similarity measure (content-based similarity approach) through TF-IDF and Cosine algorithms to group the co-related queries in destined clusters [4]. This approach depends on the query terms for grouping the related queries in the same MapReduce cluster.

1.2. Locality

Data locality is one of the main elements of MapReduce and has a significant impact on its performance. The main concept of data locality is that ''moving computation towards data is cheaper and easier than moving data towards computation'' [5]. This means that the best two cases of locality, are 1) when data is stored in the same node and on the same rack, and 2) when data is stored in another node but on the same rack.

1.3. The Hadoop framework

It consists of two main components
- HDFS.
- MapReduce.

1.3.1 HDFS

In the process of storing large data, Hadoop uses the Hadoop Distributed File System [HDFS]. The large input file is broken into a set of blocks. The default size of each block is 64 MB, but the splitting process is done randomly. HDFS consists of two main components: Master is Name Node, and Slaves are Data Nodes. Name Node is the main node which

manages the other nodes which, act as workers. When the MapReduce process starts, the Name Node uses the Job Tracker process to assign tasks that are performed by DataNode, through Task Tracker processes. Several Map processes will be executed on each DataNode, and intermediate results will be provided to the combiner process, which generates, for example, the word count in a file on a machine. The values are randomly ordered and sent to Reduce processes, which generate the final output [5,6].

1.3.2 MapReduce

The MapReduce concept helps to understand how to handle big data using clustered scale-out data processing techniques. Hadoop is an Apache open source software to implement MapReduce technique. MapReduce has two processing stages: Map and Reduce. The Hadoop runtime splits the large input file and distributes Map tasks to individual cluster nodes for parallel execution. Each Map task is responsible for the processing of one split that was distributed from HDFS. Once all Maps tasks are completed each one creates its own intermediate results and the Shuffling phase starts. The Shuffling phase groups intermediate results obtained from Map task in Fig. 1. A Reduce task consists of two phases: Shuffle and Reduce.

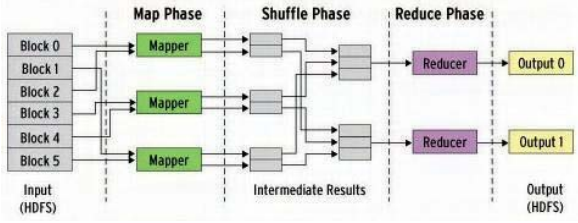


Fig.1. Native Hadoop

Shuffling is the process of transferring data from the Mappers to the Reducers grouping data by the key and passing it to Reducer [7, 8]. Shuffling increases the overhead and decreases performance. The Reduce phase is a process in which the results are aggregated from all Maps after the shuffling phase is ended and writes the result to HDFS [9].

II. RELATED WORK

A lot of effort has been spent to increase the performance of MapReduce based on data-aware techniques and reducing shuffling. In this paper, we present an overview of such work and then classify them according to the phase and the domain of application.

Li et al. [10] introduced two algorithms: Locality-Enhanced Load Balance (LELB) algorithm, and Map, Local Reduce, Shuffle and final Reduce (MLSR) algorithm to enhance the performance of MapReduce through overlapping between shuffle and local reduce, based on the key distribution of intermediate pairs, it schedules the workload in MapReduce, and it is used to reduce the communication costs in the shuffling phase through overlapping Mapping and Shuffling. It improved the execution performance up to 9.2% (for Merge Sort) and 14.4% (for Wordcount), using traditional Hadoop.

Yu et al. [11] introduce a novel virtual shuffling strategy that reduces MapReduce shuffled I / O and reduces power consumption and saves energy through the adapted of virtual shuffling and reduces power consumption and saves energy. The combination of the three-stage table, the near-demand consolidation, and the dynamic and balanced merging subtree causes the virtual shuffling algorithm cause data transferring in MapReduce to move faster, and the job execution is done quickly within a reasonable time. According to the results of the virtual shuffling I / O access saving 12% in power consumption for MapReduce applications.

A. Davidson et al. [12] proposed a solution based on a hybrid storage system that improves Shuffle I/O performance by using Hard Disk Drives (HDDs) and Solid-State Disks (SSDs) stored in large data sets to address the shuffle phase of MapReduce and enhance performance.

S. Gault et al. [13] introduced two algorithms with two-phase algorithms that can improve the data transfer between nodes in the shuffling phase based on bandwidth constraints when the switching bandwidth is known, but the list-based algorithm It is difficult to expand, it requires the switch bandwidth to be as an integer multiple of the link bandwidth.
Necolae, Bogdan et al. [14] proposed a strategy called novel shuffle data transmission, which can be achieved by using spark as a framework for minimal memory utilization. The advice of this strategy depends on two factors. Due to the extra overhead, the first one avoids synchronization between nodes. While the other minimizes the interface between the independent shuffle. The time required to store large amounts of data used to collect all shuffle data can be complicated by I / O bottlenecks. Because it poses a significant burden on the network infrastructure and limits high performance.

Hsu, Slagter, and Chung et al. [15] addressed the locality and loading aware through visualization. Virtualization provides a way to create logical subsets of resources from a physical machine from the resources point of view. From the user's point of view, the virtual environment behaves like a physical machine (PM).

Y. Zhou et al. [16] proposed to decouple the shuffle and restore tasks by iShuffle and convert them into a platform. iShuffle is implemented as a configurable plug-in implementation in Hadoop to improve multi-

user performance and reduce job completion time by up to 30.2% with optimized data shuffling.

Y. Wang et al. [17] designed and implemented a portable library called JVM-Bypass Shuffling (JBS) that could eliminate the overhead and limitations of the Java Virtual Machine (JVM), while reducing the execution time of Hadoop jobs by as much as 66.3 percent and reduce CPU utilization by 48.1%.

Zaharia et al. [18] built a spark system to solve the problem of disk I/O bottleneck caused during data shuffling and sorting the data sent to reduce task. Spark sort data in a preprocessing step by relaxing such constraint and computation can be pipelined before the reduce task to be accomplished in memory, to complete spark algorithm you must have a high memory to shuffle data to gain high performance.

P. Padala, K. Park, D. Presotto et al. [19] Design and implement a new multi-tenant called ShuffleWatcher that shapes and reduces Shuffle traffic resulting from data transfer in shuffling phase through overlapping between Map and shuffle phases of MapReduce. ShuffleWatcher improves cluster performance based on three techniques. The First One Network-Aware Shuffle Scheduling (NASS), The other two techniques Shuffle-aware Map Placement (SAMP) and Shuffle-aware Reduce placement (SARP). ShuffleWatcher improved the cluster throughput by 39-46% and job turn-around times by 27-32%.

Table 1. (Adapted from [9]) lists chronologically the research articles under consideration, together with the phase considered, the domain of application and the result for each research. We can realize that the research initially focused on fault tolerance [20] and was concerned with the Map phase. The focus was then directed to the Load-Balance [21] and Virtualization targeting the Shuffle phase. Research then approached both Map and Shuffle phases together, working on the improvement locality and data transformation. Pre-Processing was also worked upon for both Map/Shuffle, and Map/Reduce phases.

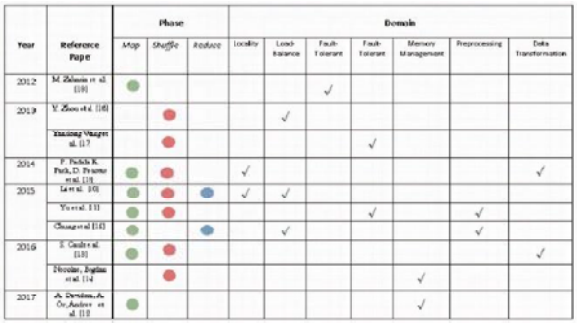


Table1: Analysis of Shuffling Techniques of MapReduce (Adapted from [9])

III. PROPOSED APPROACH

3.1. Query Similarity Measures

"There is a fundamental similarity to the definition of a cluster". Clustering a collection of documents/terms into groups within which similarity within a cluster is larger than across clusters has more than one purpose, e.g., Speeding up search and partitioning. There are several measures and techniques of similarity like Jaccard Coefficient, Euclidean distance, Cosine distances and more. The Cosine measure is better than the previous techniques since it is large when the vectors point in the same direction (i.e., are similar) [22]. In this paper, we use the Cosine similarity measure with term frequency (TF) and inverse document frequency (IDF) techniques to build (content-based similarity approach). We can summarize the similarity measures estimation in the following:

- Represent documents by TF-IDF vectors.
- Calculate the Cosine of the angle between document vectors.

3.2. TF-IDF Algorithm

Several methods of calculating values have been developed, also known as term weights. One of the most well-known schemes is TF-IDF weighting [23]. The TF-IDF weighting scheme consists of two factors (1):

1. TF is the Term Frequency. A rationale for this factor is that the document containing a term t more frequently is generally more relevant to a query. i. e. The intention of the user. The more frequently the terms appear in the document, the higher the score. The square root of the term frequency is usually used as a damping function to limit the effects of many terms in a single document.

2. IDF is the Inverse Document Frequency, or the amount of information provided by its measurement term, i.e., Whether the term is common or rare in all documents, i. e. the higher the number of the term in different documents, the lower its score is. The score is a logarithmically scaled fraction of the total number of documents in the set D (N) to the numbers of documents containing the term (nt). If the term t is not in the set D, this will result in a zero divisor IDF factor. So, usually, add 1 to adjust the denominator. In general, the TF-IDF weight of item t for a single document d:

$$TF-IDF(t,d) = TF(t,d).IDF(t) = \sqrt{freq}(t,d).\log\left(\frac{N}{N_t+1}\right). \quad (1)$$

3.3. Cosine Similarity Algorithm

The individual weights of the term of each query can provide the ability to score and rank related queries [24]. The complex and widely used ranking function are to compare the deviation of the vector between two queries. In practice, it is easier to calculate the cosine of the angle between the vectors (2), not the angle itself:

$$SIMCOS(Q_1, Q_2) = \frac{Q_1 \cdot Q_2}{|Q_1||Q_2|} \quad (2)$$

The result is a non-negative similarity score between

0.0 and 1.0, where a similarity score of 1 means that query1 (Q1) and query2 (Q2) are identical.

3.4. Clustering Algorithm

Partitioning Clustering: This type of clustering algorithm involves object allocation. It considers k clusters and assigns each object to the cluster that suits it well. The number of clusters K needs to be specified by the user. Each cluster is represented by an object called the centroid the characteristics of which are specified according to the domain knowledge studied. The most famous data clustering technique is K-means [25]. Before being able to execute K-means and grouping the related text documents with each other in sets, the documents must be represented as mutually comparable vectors. To accomplish this task, the documents can be represented using the TF-IDF score, and we calculate the similarity between queries in that cluster, using the cosine similarity described previously [26].

3.5. ELRS Technique

Our approach is designed for enhancing the data locality and reducing the overhead of shuffling phase in Hadoop and is called (ELRS) technique Fig.2.

The ELRS is built in the first phase (pre-processing) by the combination of three algorithms. The first one called the TF/IDF algorithm. we are using the TF/IDF algorithm to assign weights to a set of SQL queries. The numbers of generated queries are 1200 queries and are divided into four groups ,150 ,300 ,600 ,1200 queries respectively. The numbers of generated queries are built in the TPC-benchmark. The number of tables of this benchmark is eight tables, five of them in this contribution. The TF/IDF is the most well-known methods that have been developed to evaluate term weight.

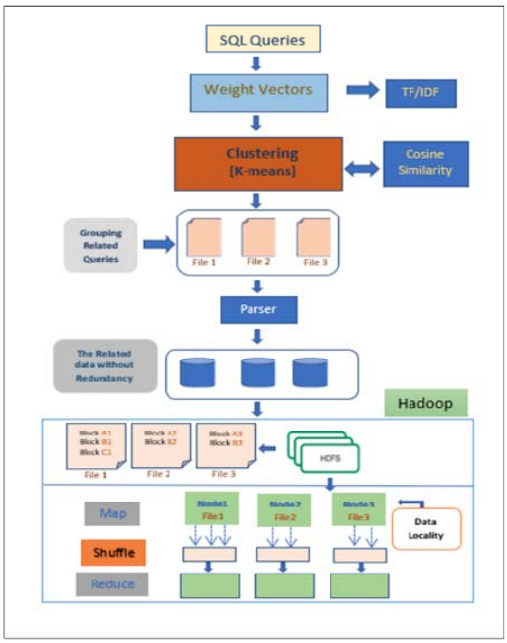


Fig.2. ELRS Technique

The second algorithm belongs to the measure of similarity. the cosine similarity algorithm is used to calculate the similarity between any two queries. Before this query are weighted by the TF/IDF algorithm. the last algorithm in the pre-processing phase is the K-means clustering algorithm. Applying this algorithm in the similarity of queries to grouping the related queries with each other. Just the related queries are grouped in the same files we are using the parser to eliminate the redundancy. the redundancy columns of queries are one of the most elements that impact the performance of ELRS. Before the redundancy is removed by the parser, the data is fetching to support the HDFS.

In the second phase; HDFS does not provide any means to co-locate related data on the same set of nodes. The simple edit in HDFS is saved the related blocks of the same files in the same node to achieve the best data locality.

In this thesis, the performance gain is due to the elimination of data shuffling, and the reduce phase of MapReduce through enhanced the localization of data.

IV. EXPERIMENTATIONS

4.1. The TPC Benchmark (TPC-H)

To illustrate this approach, we used the Transaction Processing Performance Council (TPC) Benchmark [27]. The TPC Benchmark (TPC-H) is a decision support benchmark that consists of a suite of business-oriented ad-hoc queries and concurrent data modifications. The components of the TPC-H database are defined consisting of eight separate tables. As experimental data, the TPC-H benchmark has been chosen since it is scalable to different data set sizes and therefore provides a good basis for comparisons of execution times over differently scaled data sets.

4.2. Test Environments

For our test environment, we evaluate our approach in the MapReduce framework by using six nodes. Each node with Intel Core i5 2.4GHz processor, 4 GB of memory, 500 GB SATA disk and operating system Ubuntu 16. 4 Linux with Apache Hadoop release 2.9.0. Other test environment factors such as I/O Local Disk and, I/O Network factors, are not in our research focus. We focus only on the runtime for algorithms to complete the pre-partitioning operation with different data in size and different algorithms in MapReduce jobs. We use the TPC-H benchmark dataset to compare our implementation with original Hadoop. We use five tables customers, orders, lineitem, supplier and partsupp. The size of these tables from 200 million records up to 800 million records. We generate 1200 queries to implement our algorithm. And we implement the techniques

proposed earlier in this paper. We have different two experiments:

Experiment 1: we discuss the results of the pre-processing phase including weight vectors algorithm and similarity measures together with the clustering algorithm.

Experiment 2: we compare our new algorithm ELRS with Hadoop native with increasing data size.

4.3. Experiment 1

We present the pre-processing phase by building the clusters of queries for each group using K-means Clustering with TF/IDF and cosine similarity [25]. Table 2. shows the Query representation in the pre-processing phase and also illustrates the details for distribution of clusters for every group.

Test Sets	No. of Queries	Clustering		Parsing Execution Time to Remove Redundancy (Sec)	Input Size of HDFS
		NO. of Clustering	Run time (Sec)		
Group 1	150	3	36.03	13.3	4.2 GB
Group 2	300	3	148.46	29.2	8.6 GB
Group 3	600	5	244	63	16.9 GB
Group 4	1200	6	306	129	33.8 GB

Table 2. Query Representation in the Preprocessing Phase

In this phase, we handle the collection of SQL queries with different data sizes and partition them into clusters where each cluster contains the related queries.

4.3.1 Query Parser

The parser is accountable for syntactic checking of a query. The task of the parser is to take text written in SQL statement and transfer it into a parse tree that is a tree whose nodes equal to either fields or syntactic categories. Fields are elements such as keywords, names of attributes or relations, constants, parentheses, operators, etc. Syntactic categories are named for the families of query subpart those all play a similar role in a query. For example, the syntactic category simple-select will be used to represent any simple SELECT query in common select-from. The main goal of the parser is to remove the data redundancy from the queries as described in Fig.2, and the runtime of the query parser as described in Table .2.

4.3.2Data Prefetching

The main goal of our contribution is to achieve the data locality to improve the performance of MapReduce. After the parser is finished a data prefetching is started, we can fetch the data and pass it to HDFS to distribute the blocks of data to the numbers of nodes and set the related data with each other in the same node. It is proved that the proposal of this thesis reduces data transmission overhead and achieve the localization of data.

4.4. Experiment 2

4.4.1 Data Locality

We measure data locality by the total amount of data stored locally. When data is local, access is fast. Data locality is, therefore, a reflection of performance for MapReduce. In the first part of the last phase, we measure the data localization for each group of queries. We used the groups of the related data performed in the pre-processing phase to complete the experiments of our approach. When the numbers of queries are 150, the data locality is enhanced by 32.3 %, When the numbers of queries are 300, the data locality is enhanced by 28.2 %, When the numbers of queries 600, the data locality is enhanced by 25.1 %, and when the numbers of queries are 1200, the data locality is enhanced by 22.8 % as shown in Fig.3.

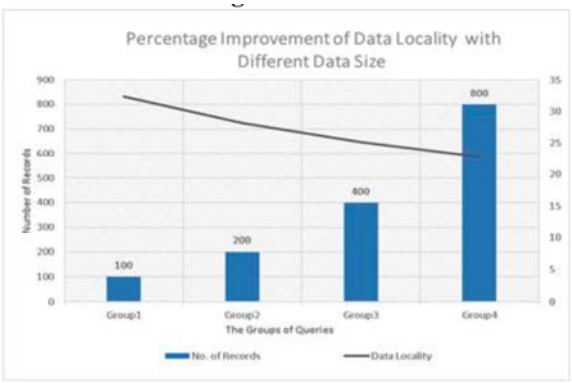


Fig.3. Data Locality

The average Experimental results show an improvement in MapReduce performance, improving data locality by around 27.2 %.

4.4.1 Query Performance

The goal of ELRS is to reduce the execution time for the query search to improve the performance data analysis jobs in MapReduce. In ELRS, we use average execution time for samples of queries to evaluate MapReduce jobs performance. The localization of related data in the same nods is an important factor to indicate enhance of the MapReduce performance. In the following part, we present the execution time performance of our technique compared with native Hadoop.

In Fig.4, compares the execution time of four queries samples by using ELRS and native Hadoop in around 100 million records of TPC-H benchmark. From reading the result presents the ELRS has outperforms compared with native Hadoop by around 46.7 %.

In Fig.5, compares the execution time of four queries samples by using ELRS and native Hadoop in around 200 million records. The result presents the ELRS has outperforms compared with native Hadoop by around 40.5 %.

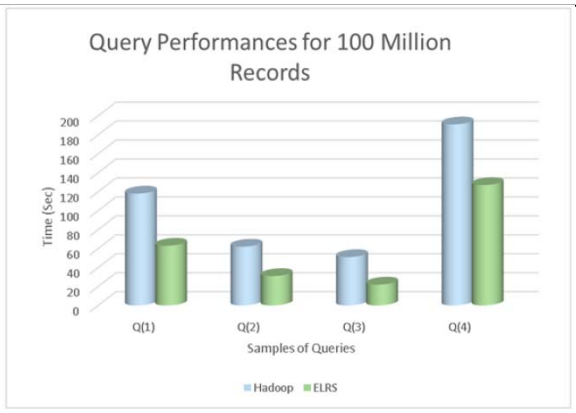


Fig.4. Query Performances for 100 Million Records

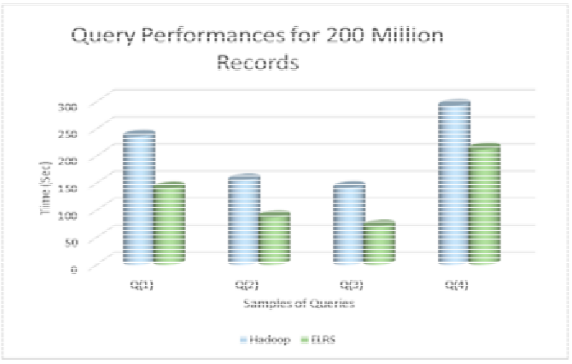


Fig.5. Query Performances for 200 Million Records

Fig.6 compares the execution time of four queries samples by using ELRS and native Hadoop in around 400 million records. From reading the result presents the ELRS has outperforms compared with native Hadoop by around 36.4 %.



Fig.6. Query Performances for 400 Million Records

Fig.7 compares the execution time of four queries samples by using ELRS and native Hadoop in around 800 million records. The result presents the ELRS has outperforms compared with native Hadoop by around 33.9 %.

We after reviewing the results of the improvement achieved in the shuffling phase of the research papers mentioned earlier. Compared with them, we find that our results are limited to them as an average and in the future, we seek to increase the rate of improvement after overcoming the challenges and provide a working environment with higher specifications than the current.



Fig.7. Query Performances for 800 Million Records

V. CONCLUSIONS

In this paper, we interduce a new idea and implementation of our new algorithm, called ELRS algorithm; we focused on an approach to improve the performance of MapReduce through achieved the locality of data, which improve locality of data leads to eliminating the network overhead in the shuffling phase for the MapReduce

The proposal of this strategy depends on two phases, the first one by pre-partitioning data based on Query-based similarity through TF-IDF and Cosine similarity algorithms and grouping the related queries with each other by K-means clustering algorithm. The second phase we have supported HDFS with the related data that was processed in the phase one and control where are stored it to co-locate the related data files in the same nodes and achieved the data localization.

The experimental results obtained by the implementation of ELRS technique showed, improve data locality by around 27.2 % and a reduction in the execution time of Hadoop jobs by around 40.1%.

FUTURE WORK

Suggest for future work is by applying the semantic analysis of SQL queries. On the other hands, we will try to use the GPU instead of the CPU because it is high-power and may help save reasonable time in the processing process[28]. We can enhance this work in the future to the speeding up of search queries and reduce the execution time of MapReduce jobs in big data [29,30].

REFERENCES

[1] Sahal, R., Mohamed Helmy Khafagy, Omara, F.A.,"Comparative study of multi-query optimization techniques using shared predicate-based for big data ",International Journal of Grid and Distributed Computing

9(5), pp. 229-240,2016.
[2] Sahal, R., Mohamed Helmy Khafagy, Omara, F.A.,"Exploiting coarse-grained reused-based opportunities in Big Data multi-query optimization", Journal of Computational Science, 26, pp. 432-452, 2018
[3] Shanoda, M.S., Senbel, S.A., Mohamed Helmy Khafagy,"JOMR: Multi-join optimizer technique to enhance Map-R
[4] educe job", 9th International Conference on Informatics and Systems, INFOS 2014, 7036682, pp. PDC80-PDC87, 2014
[5] G. Salton & M.J. McGill, Introduction to modern information retrieval, McGraw-Hill New York, 1983. G. Salton and M. McGill, editors. Introduction to Modern Information Retrieval. McGraw-Hill, 1983.
[6] Abdel Azez, H.S.H., Mohamed Helmy Khafagy, Omara, F.A.," Optimizing Join in HIVE Star Schema Using Key/Facts Indexing", IETE Technical Review (Institution of Electronics and Telecommunication Engineers, India),35(2), pp. 132-144,2018.
[7] Mahmoud, H., Hegazy, A., Mohamed Helmy Khafagy. "An approach for big data security based on Hadoop distributed file system", Proceedings of 2018 International Conference on Innovative Trends in Computer Engineering, ITCE 2018, 2018-March, pp. 109-114,2018.
[8] Hsu, C.-H., K. D. Slagter, and Y.-C. Chung. 2015. "Locality and Loading Aware Virtual Machine Mapping Techniques for Optimizing Communications in MapReduce Applications." Future Generation Computer Systems 53: 43–54.
[9] Zhang, Hao, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang. "In-memory big data management and processing: A survey." IEEE Transactions on Knowledge and Data Engineering 27, no. 7 (2015): 1920-1948.
[10] El-douh, A. A., Elkadi, Mohamed Helmy Khafagy query. (2017). SAUSSUREA (ISSN: 0373-2525), 2017 Volume 4 (2): PP 42-49 RESEARCH ARTICLE / REVIEW ARTICLE Comparative Study of Improving Shuffling Techniques in MapReduce Volume 4 (2): PP 42-49 RESEARCH ARTICLE / REVIEW ARTICLE, 4(2), 42–49.
[11] J. Li, J. Wu, X. Yang, and S. Zhong, "Optimizing MapReduce Based on Locality of K-V Pairs and Overlap between Shuffle and Local Reduce," 2015 44th Int. Conf. Parallel Process., pp. 939–948, 2015.
[12] W. Yu, Y. Wang, X. Que, and C. Xu, "Virtual shuffling for efficient data movement in MapReduce," IEEE Trans. Comput., vol. 64, no. 2, pp. 556–568, 2015.
[13] Davidson, A., & Or, A. (n.d.). Optimizing Shuffle Performance in Spark. Retrieved from http://www.cs.berkeley.edu/~kubitron/courses/cs262a-F13/projects/reports/project16_report.
[14] Gault S, Perez C. Dynamic Scheduling of MapReduce Shuffle under Bandwidth Constraints[J]. Lecture Notes in Computer Science, 2014.
[15] B. Nicolae, C. Costa, C. Misale, K. Katrinis, and Y. Park, "Towards Memory-Optimized Data Shuffling Patterns for Big Data Analytics," Proc. - 2016 16th IEEE/ACM Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2016, pp. 409–412, 2016.
[16] Hsu, C.-H., K. D. Slagter, and Y.-C. Chung. "Locality and Loading Aware Virtual Machine Mapping Techniques for Optimizing Communications in MapReduce Applications." 2015. Future Generation Computer Systems 53: 43–54.
[17] Y. G. and J. R. and X. Zhou, "iShuffle: Improving Hadoop Performance with Shuffle-on-Write," 10th Int. Conf. Auton. Comput. ICAC'13, San Jose, CA, USA, June 26-28, 2013, pp. 107–117, 2013.
[18] Y. Wang, "JVM-Bypass for Efficient Hadoop Shuffling," 2013.
[19] M. Zaharia et al., "Resilient Distributed Datasets A Fault-Tolerant Abstraction for In-Memory Cluster Computing Motivation."
[20] Faraz Ahmad, Srimat T. Chakradhar, Anand Raghunathan , T. N. Vijaykumar, ShuffleWatcher: Shuffleaware scheduling in multi-tenant MapReduce clusters, Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference, June 19-20, 2014, Philadelphia, PA.
[21] Kaseb, M.R., Mohamed Helmy Khafagy, Ali, I.A., Saad, E.S.M." Redundant independent files (RIF): A technique for reducing storage and resources in big data replication ",Advances in Intelligent Systems and Computing,745, pp. 182-193, 2018
[22] Sarhan, E., Ghalwash, A., Mohamed Helmy Khafagy." Queue weighting load-balancing technique for database replication in dynamic content web sites ", Proceedings of the 9th WSEAS International Conference on Applied Computer Science, ACS '09, pp. 50-55,2009
[23] Bordag, S. (2008). A Comparison of Co-occurrence and Similarity Measures as Simulations of Context pages 52–63. Springer Berlin Heidelberg, Berlin, Heidelberg.
[24] White, H. D. (2017). Pennants for Garfield: bibliometrics and document retrieval. Scientometrics, 114(2), 1–22. Doi:10.1007/s11192-017-2610-9.
[25] Shoaib, M., Daud, A., and Khiyal, M. (2014). An improved similarity measure for text documents. J. Basic Appl. Sci. Res, 4(6):215–223.
[26] Zahir, J., Qadi, A. E. L., & Aboutajdine, D. (2015). Access Plan Recommendation Using SQL Queries Similarity Access Plan Recommendation Ap-, 14, 638–645.
[27] Marwa Hussien Mohamed, Mohamed Helmy Khafagy, Mohamed Hasan Ibrahim,"Recommender Systems Challenges and Solutions Survey," 2019 International Conference on Innovative Trends in Computer Engineering (ITCE'2019), Aswan, Egypt, 2-4 February 2019,pp.149-155, DOI: 10.1109/ITCE.2019.8646645.
[28] www.tpc.org 2018.
[29] Marwa Hussien Mohamed and Mohamed Helmy Khafagy. "Hash semi cascade join for joining multi-way MapReduce." 2015 SAI Intelligent Systems Conference (IntelliSys)(2015): pp.355-361. 29.
[30] Marwa Hussien Mohamed, Mohamed Helmy Khafagy, Mohamed Hasan Ibrahim . "Hash Semi Join Map Reduce to Join Billion Records in a Reasonable Time." Indian Journal of Science and Technology, vol. 11, no. 18, Jan. 2018, pp. 1–9., doi:10.17485/ijst/2018/v11i18/119112.
[31] Marwa Hussien Mohamed, Mohamed Helmy Khafagy, Mohamed Hasan Ibrahim . "From Two -Way to Multi-Way: A Comparative Study for Map-Reduce Join Algorithms." WSEAS Transactions on Communications, Volume 17, 2018, pp. 129-14

★★★