Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 6.017

IJCSMC, Vol. 7, Issue. 4, April 2018, pg.166 – 172

Performance Improvement Techniques for MapReduce - A Survey

Ebraheem M.Alhaddad[1], Fathy E. Eassa[2]
[1]King Abdulaziz University, Faculty of Computing and Information Technology
Computer Science Department, Jeddah, KSA
Email: alhaddad92@yahoo.com, ebraheem@kau.edu.sa
Telephone: (+966) 538384968
4163 - Abruq Ar Rughamah Unit No. 3, Jeddah 22262 – 7302, Kingdom of Saudi Arabia
[2]King Abdulaziz University, Faculty of Computing and Information Technology
Computer Science Department
Email: feassa@kau.edu.sa

Abstract— Enterprises these days acquire huge volumes of data from totally different sources and leverage this information by means that of data analysis to support effective decision- making and supply new practicality and services. The key demand of data analytics is scalability, merely thanks to the vast volume of information that requires to be extracted, processed, and analyzed in a very timely fashion. Arguably the foremost widespread framework for modern large-scale data analytics is MapReduce, in the main thanks to its salient features that embrace scalability, fault-tolerance, simple programming, and flexibility. However, despite its deserves, MapReduce has evident performance limitations in miscellaneous analytical tasks, and this has given rise to a major body of work that aim at up its potency, whereas maintaining its fascinating properties. This survey aims to review the state of the art in improving the performance of MapReduce.

Index Terms— Big Data, Data analysis, Cluster Computing, Data Management, Metadata, MapReduce

I. INTRODUCTION

Big data is attracting widespread interest of both information technology specialists in research and business communities. After reaching the zettabyte barrier in 2010 and exceeding 1.8 zettabytes in the generated- increasing in the five years by a factor of 9 [1]. The forecast for 2025, is that the internet will overcome the every living being brain capacity in the whole word. Although we have recognized the importance of big data now a days, researchers are still having different opinions and definitions. This is because of the different needs and approaches for each researcher, enterprises, data analysts and technical practitioners.

Doug Laney[2], an analyst at the META group (presently Gartner) is considered to be the one to offer the first

real big data definition, nevertheless the term "big data" had been used earlier. Although, the term big data didn't stated explicitly in the definition, the 3V,which is the main big data characteristics, are introduced for the first time. The Vs resemble volume, velocity and variety. The 3V definition will dominated the big data definition, presented by IT specialists such as Gartner, IBM and Microsoft, for the years to come.

In this report [3] McKinsey \& Company studied the potential value that big data can create for organizations and sectors of the economy and seeks to illustrate and quantify that value. The research discovered that data make a remarkable change in the world economy by improving businesses efficiency and productivity and support them in competing with the public sector which will eventually benefit consumers. For example, The use of big data can enhance the efficiency and quality of the heath care service in US by an estimated value of three hundred billion USD yearly. 2/3 of that in health care expense reduction.

With the tremendous generated and stored data by many sources, it vital for companies and organization to exploit new technologies in order to obtaining maximum value of big data. Volume, Variety and Velocity unique characteristics of big data that cut off the development of new technologies facilitate getting value from data. Big data management and analysis is at the core of the techniques has to be developed for the best benefit of big data value. In literature many frameworks developed for this purpose such as MapReduce, Spark, SCOOPY...etc. Despite the advantages provided by these systems to the fact that it still suffers from some defects and need to be treated. Some analysis operations may need to be performed only part of the data. Currently analysis tasks are performed on all data regardless of this fact. Thus developing methods to facilitate selective data access, which helps improve the performance of data analysis and management systems. After the map function complete processing the data, its output is send to the reduce cluster nodes. The transmission process degrade the overall system performance. In addition, Extracting metadata about the data to be saved may help in the classification of large amounts of data and thus save the most important data and neglect the data that is less important or redundant and does not add value to the data.

II. THE MAPREDUCE FRAMEWORK

In the era of Big Data, characterized by the unprecedented volume of data, the velocity of data generation, and the variety of the structure of data, support for large-scale data analytic considered a challenging task[4]. The developing interest in big data analysis systems has led to the advancement of original techniques from both the business and the scientific fields. Despite parallel database frameworks [5] provide some of these data analysis applications, they are costly, hard to manage and need to be adapted to fault-tolerance for long-processing queries [6] [7]. Different frameworks have been produced for the most part by business to address Big Data analysis, including Google's MapReduce[8], [9], Yahoos PNUTS [10], Microsoft SCOPE [11], Twitters Storm [12], LinkedIn's Kafka[13], and WalmartLabs Muppet[14]. Likewise, a few organizations, including Facebook [15] , both utilize and have subscribed to Apache Hadoop and its ecosystem.

MapReduce has turned into the most prevalent framework for big data storing and analysis of big data mining by cluster of machines, mainly as a result of its simplicity[4]. MapReduce [8] is a model which is presented by Google for programming high performance clusters to perform big data analysis in one pass. The MapReduce is designed in a way that it can scale up to a cluster of hundreds an even thousands of data nodes in a fault-tolerance manner. One benefit of this system is its dependence on a basic and capable programming model. Furthermore, it detaches the application designer from all the complicated and subtle elements of running a distributed program, for example, issues on data location, job scheduling and fault-tolerance [16] [7].

The MapReduce framework is presented as an easy and intense programming model that empowers simple development of scalable distributed applications to process big data running on high performance clusters. One of the benefits of this approach is that it detach job developers from the lower level implementation details such as data localization, job scheduling and fault-tolerance. The design of this framework follow the divide and conquer model. The model comprises of two main functions specifically Map and Reduce. In the Map function, the input data is sub partitioned into many small chunks, each chunk is fed to a Map function, processed and create average key/value output. This key/value data is rearranged in a process called shuffling using merge-sorted algorithms and then send to the cluster nodes running the Reduce function. In turn, the Reduce function process this all intermediate output related to the same key/value pair and produce the final output. It should be noted that the shuffle phase is much more time consuming than the Map and Reduce tasks [4]. Figure 1 depicts a pseudo-code for a MapReduce analysis job; the job is counting words frequency in an input document data. In this example,

the map function generate each word as a key and 1 as a value for the key/value pair for each word in the input data, while the Reduce function count the value of all key/value pairs for the same key(word).

```
map(String key, String value):
 // key: document name
 // value: document contents
for each word w in value:
EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
      result += ParseInt(v);
    Emit(AsString(result)); Figure
```

Figure 1. Mapreduce Job Example.

Figure 2 illustrate the MapReduce flow of execution. The MapReduce job execution workflow is as follows:

- The input data for the MapReduce is split into almost even M splits, and the program is cloned into many instances so that each instance can run in a cluster data node.
- The program instances are classified into: one master and the rest are considered as workers. The master then assign the job for the workers. The workers are intern classified into M map tasks and R reduce tasks. The master run in the cluster master nodes, while each worker run in a data node.
- The M data splits stored at each data node are processed by the map function. Each map function process the M split and produce an intermediate file containing the corresponding key/value. This file is temporarily buffered at local memory.
- The buffered files are then partitioned and stored to the local disk. The location of these files are sent to the master node, which intern pass these locations to the reduce.
- After the map functions complete processing a certain percentage of the data, the master worker notify the reduce workers to start processing. In order to each reduce worker process files containing key/value data for the same key; a shuffling process sort the map output files by the value of the key. The reduce workers receive the location of the sorted files from master and then transfer the data to the reduce cluster nodes.
- The reduce function processes the data and start to fill the final output file. When all map and reduce functions finish processing, the location of the final output file location is sent to the master node.
- The master worker send the final output file to the user program.

Periodically, the master worker ping all map workers. If any worker don't respond in a certain amount of time, it is considered as a fail worker and the node containing a replica of the fail worker is assigned the job. This is necessary because a map worker store intermediate data in its local file system and when it fails the data is no longer accessible. On the other hand, a failed reduce worker isn't rescheduled because its output is stored in the global file system. When a map worker finish processing successfully, it return to idle state and therefore reading to receive a new job.

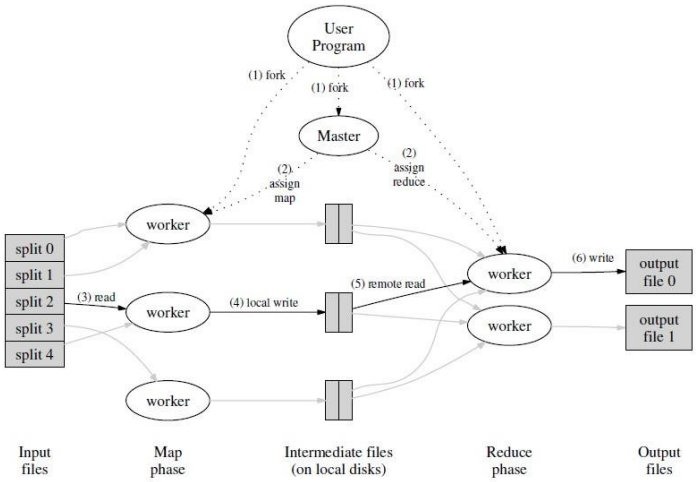

Figure 2. MapReduce Framework[8], [9].

III. MAPREDUCE ADVANTAGES

Key advantages of the MapReduce framework include:

- **Simplicity:** programming jobs to run using MapReduce is simple. An experience with parallel systems or under- standing of the system infrastructure is not required.
- **Fault-tolerance:** Executing big data analysis jobs is usu- ally done using computer clusters consisting of thousands of data nodes. In an environment such as this, defects in some of these data nodes are expected to occur. MapReduce can deal with such problems so that no loss of results or interruption of work.
- **Flexibility:** MapReduce does not require data to be organized in a specific format. Thus, the data of any structure can be analyzed using MapReduce directly and without any prior data preprocessing operations.
- **Scalability:** MapReduce can scale to clusters of PCs.

IV. MAPREDUCE WEAKNESSES

Despite its evident merits, MapReduce often fails to exhibit acceptable performance for various processing tasks. We have identified a list of issues related to large-scale data processing in MapReduce that significantly impact its efficiency.

A. Data Access:

Data Access is a critical task for efficient and high performance of execution of a job. For cretin types of big data analytic tasks, access of input data of only some data nods or even accessing a selected data in those data nods is needed. MapReduce implement a brute force access for all input data at every job execution, which degrades the overall system performance.

B. High Communication Cost:

The output of the Map task is send to the Reduce nodes. The size of the transmitted data depend on the type of input data and type of analytic job. As the size of data increase, drain the network throughput and decrease overall system performance.

C. Redundant and wasteful processing:

In MapReduce, jobs the use the same input data and perform similar sub tasks are not allowed to share results, resulting in redundant data processing and wasting processing power.

D. Re-computation:

In parallel database systems, reuse of previous query results is achieved by query result materialization. Results of queries that are expected to run more often are materialized on disks. On the other hand the MapReduce framework implement fault-tolerance by enforcing a check point mechanism. The buffers of the map worker periodically save intermediate data into local disk. Thus, in a case of failure using these check points are used to resume the work. However, after the MapReduce finish the jobs successfully, the results are discarder. For a new job that need par or all of the result of a previously completed job is not supported by the MapReduce. Therefore, a new job need to rerun the previous job a gain to get the results. • Lack of early termination: In some analysis task no need to process all the data to get the final result. For instant, a query may find the final result after processing 60% of the data. Unfortunately, MapReduce Unfortunately the design does not support stopping data processing when the result is obtained before all data is processed. This leads to wasting time and resources.

E. Lack of iteration:

Recursive and iterative computation queries are common in data analysis tasks, such as PageRank algorithms, clustering, social network analysis. . . etc. However, in order to such iterative tasks in MapReduce framework, the programmer has to write a MapReduce job for each iteration and coordinate the execution sequence to get final result. More importantly, a significant performance penalty is paid, since data must be reloaded and reprocessed in each iteration. Supporting empirical analysis task: In some cases, as in the analysis of scientific

data, the analysis task itself isn't identified. In other words, we may need to do more than one empirical analysis task and implement each one and observe the results accordingly. The task is adjusted until we reach the final task. In such cases it is useful to run these empirical tasks on a part of the data and when the link to the final version is implemented using the full data.

F. Load balancing:

In order to improve performance and reduce the time required to process data, Parallel data management systems apply methods to divide and distribute data on processing units in order to distribute the processing burden on these units. This distribution process must be done carefully to achieve the desired goal. Unless the data is distributed to the processing units equally, the unit with the largest data will tack a loner time to finish, therefor it will represent a bottleneck to the whole system and we will lose the advantage for which the work was distributed in the first place. Data partitioning schemes that are not data-aware, such as MapReduce data partitioning schemes causing the occurrence unbalanced data distribution. More importantly, the size of the data affects processing time but is not the only factor. The nature of the data itself affects the time of execution. Complex data for example takes longer to process. Therefore, the development of intelligent mechanisms for the distribution of equal voltage, ensuring the best performance of the system, between the elements of treatment and take into account all the elements that affect the completion of tasks.

G. Supporting real-time or interactive processing:

MapReduce is big data analysis framework characterized as a batch processing highly fault-tolerant system. The design of MapReduce so that these features can be achieved on the other hand makes the design unable to support other systems such as real-time processing systems. For instance, in order to achieve high fault tolerance, MapReduce enforce Map functions periodically to writ intermediate results into local disks; Reduce workers transfer big amount of data from Map workers. Unfortunately, the performance of MapReduce degrades because of these mechanisms.

V. MAPREDUCE IMPROVEMENTS

In this section, an overview is provided of various methods and techniques present in the existing literature for improving the performance of MapReduce.

- **Indexing:** Hadoop++[17],HAIL[18] are two indexing system proposed to resolve the data access limitation of MapReduce, An another approach to resolve data access problem is by using data layouts. A survey of data layouts approaches is in [19].A column file approach, where data is partitioned vertically and then grouped based on correlated columns, is proposed in Llama [20] and Cheetah[21]. A combination between vertical and horizontal partitioning of data is introduced in RCFile[22].
- **Intentional data placement:** CoHadoop [23] colocates and copartitions data on nodes intentionally, so that related data are stored on the same node. To achieve colocation, CoHadoop extends HDFS with a file-level property (locator), and files with the same locator are placed on the same set of DataNodes.
- **Avoiding redundant processing:** MRShare [24] is a sharing framework that identifies different queries (jobs) that share portions of identical work. Such queries do not need to be recomputed each time from scratch. ReStore[25] is a system that manages the storage and reuse of intermediate results produced by workflows of MapReduce jobs. It is implemented as an extension to the Pig dataflow system. ReStore maintains the output results of MapReduce jobs in order to identify reuse opportunities by future jobs.
- **Early termination:** EARL [26] aims to provide early results for analytical queries in MapReduce, without processing the entire input data. EARL utilizes uniform sampling and works iteratively to compute larger samples, until a given accuracy level is reached, estimated by means of bootstrapping. RanKloud [27] has been pro- posed for top-k retrieval in the cloud. RanKloud computes statistics (at runtime) during scanning of records and uses these statistics to compute a threshold (the lowest score for top-k results) for early termination.
- **Incremental processing:** REX [28] is a parallel query processing platform that also supports recursive queries expressed in extended SQL, but focuses mainly on incremental refinement of results. An iterative

data flow system is presented in [29], where the key novelty is support for incremental iterations. Such iterations are typically encountered in algorithms that entail sparse computational dependencies, where the result of each iteration differs only slightly from the previous result. Differential dataflow[30] is a system proposed for improving the performance of incremental processing in a parallel data flow context. It relies on differential computation to update the state of computation when its inputs change, but uses a partially ordered set of versions, in contrast to a totally ordered sequence of versions used by traditional incremental computation.

VI. CONCLUSION

MapReduce is the most widely spread and famous big data analysis system. This is due to its significant features that include scalability, fault-tolerance, simplicity, and flexibility. Still, several of its shortcomings hint that MapReduce is not perfect for every large-scale analytical task. In this work we tried to emphasize on the work to overcome these shortages.

REFERENCES

[1] J. Gantz and D. Reinsel, "Extracting Value from Chaos State of the Universe: An Executive Summary," *IDC iView*, no. June, pp. 1–12, 2011. [Online]. Available: http://idcdocserv.com/1142

[2] D. Laney, "3d data management: Controlling data volume, velocity and variety," *Gartner.*, 2001.

[3] J. Manyika, M. Chui, B. Brown, and J. Bughin, "Big data: The next frontier for innovation, competition, and productivity," no. June, 2011. [On- line]. Available: http://www.citeulike.org/group/18242/article/9341321

[4] C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in MapReduce," *VLDB Journal*, vol. 23, no. 3, pp. 355–380, 2014.

[5] D. DeWitt and J. Gray, "Parallel database systems: the future of high performance database systems," *Communications of the ACM*, vol. 35, no. 6, pp. 85–98, june 1992. [Online]. Available: http://portal.acm.org/citation.cfm?doid=129888.129894

[6] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," *Proceedings of the 35th SIGMOD international conference on Management of data*, pp. 165–178, 2009.

[7] S. Sakr, A. Liu, and A. G. Fayoumi, "The Family of MapReduce and Large-Scale Data Processing Systems," *ACM Computing Surveys*.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[9] B. Y. J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," Communications of the ACM, vol. 53, pp. 72–77, 2010.

[10] A. Silberstein, A. Silberstein, B. F. Cooper, B. F. Cooper, U. Srivastava, U. Srivastava, E. Vee, E. Vee, R. Yerneni, R. Yerneni, R. Ramakrishnan, and R. Ramakrishnan, "PNUTS: Yahoo!'s Hosted Data Serving Platform," Proceedings of PVLDB 2008, vol. 1, no. 2, pp. 1277—-1288, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1376616.1376693

[11] R. Chaiken, B. Jenkins, and P. Larson, "SCOPE: easy and efficient parallel processing of massive data sets," Proceedings of the VLDB Endowment, vol. 1, no. 2, pp. 1265–1276, 2008.

[12] J. Leibiusky, G. Eisbruch, and D. Simonassi, Getting Started with Storm. O'Reilly, 2012.

[13] K. Goodhope, J. Koshy, and J. Kreps, "Building LinkedIn's Real-time Activity Data Pipeline." IEEE Data Eng, pp. 1–13, 2012.

[14] W. Lam, L. Liu, S. Prasad, A. Rajaraman, Z. Vacheri, and A. Doan, "Muppet: {MapReduce-style} Processing of Fast Data," Proc. {VLDB} Endow., vol. 5, no. 12, pp. 1814–1825, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2367502.2367520

[15] A. S. Aiyer, M. Bautin, G. J. Chen, P. Damania, P. Khemani, K. Muthukkaruppan, K. Ranganathan, N. Spiegelberg, L. Tang, and M. Vaidya, "Storage Infrastructure Behind Facebook Messages: Using HBase at Scale." IEEE Data Eng. Bull., vol. 35, no. 2, pp. 4–13, 2012.

[16] D. A. Patterson, "The Data Center is the Computer," Communications of the ACM, vol. 51, no. 1, p. 105, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1327452.1327491

[17] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: Making a Yellow

Elephant Run Like a Cheetah (Without It Even Noticing)," in Proceedings of the VLDB Endowment, vol. 3, no. 1-2, 2010, pp. 515–529.

[18] J. Dittrich, J.-A. Quia e Ruiz, S. Richter, S. Schuh, A. Jindal, O. Schad, J. A. Q. Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad, "Only aggressive elephants are fast elephants," in PVLDB: Proceedings of the VLDB Endowment, vol. 5, no. 11, 2012, pp. 1591–1602.

[19] J. Dittrich and J.-a. Quian, "Efficient Big Data Processing in Hadoop MapReduce," in Proceedings of the VLDB Endowment, vol. 5, no. 12, 2012, pp. 2014–2015.

[20] Y. Lin, D. Agrawal, C. Chen, B. C. Ooi, and S. Wu, "Llama: Leveraging Columnar Storage for Scalable Join Processing in the MapReduce Framework," Proceedings of the 2011 international conference on Management of data - SIGMOD '11, p. 961, 2011. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1989323.1989424

[21] S. Chen, "Cheetah: A high performance, custom data warehouse on top of mapreduce," in Proceedings of the VLDB Endowment, 2010, pp. 1459–1468.

[22] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu, "RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems," in Proceedings - International Conference on Data Engineering, 2011, pp. 1199–1208.

[23] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: flexible data placement and its exploitation in Hadoop," in Proc. VLDB Endow., vol. 4, no. 9, 2011, pp. 575–585.

[24] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas, "MR- Share: sharing across multiple queries in MapReduce," Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 494–505, 2010.

[25] A. Aboulnaga and D. I. Elghandour, "ReStore: Reusing results of MapReduce jobs," Vldb, vol. 5, no. 2012, p. AESNP3, 2012.

[26] C. Laptev, Nikolay and Zeng, Kai and Zaniolo, "Early Accurate Results for Advanced Analytics on MapReduce," Proc. VLDB Endow., no. 2150- 8097, pp. 1028–1039, 2012.

[27] L.-s. M. Retrieval, "RanKloud : Multimedia Data Server Clusters," System, pp. 64–77, 2011.

[28] S. R. Mihaylov and Z. G. Ives, "REX : Recursive , Delta-Based Data- Centric Computation," in Proc. VLDB Endow. (PVLDB), 2012, pp. 1280–1291.

[29] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, "Spinning Fast Iterative Data Flows," vol. 5, no. 11, pp. 1268–1279, 2012. [Online].
Available: http://arxiv.org/abs/1208.0088

[30] F. D. McSherry, D. G. Murray, R. Isaacs, and M. Isard, "Differential Dataflow," in Biennial Conference on Innovative Data Systems Research (CIDR), 2012. [Online]. vol. 46, no. 1, pp. 1–44, 2013.

