Improving the Map and Shuffle Phases in Hadoop MapReduce

J. V. N. Lakshmi

Abstract Massive amounts of data are needed to be processed as analysis is becoming a challenging issue for network-centric applications in data management. Advanced tools are required for processing such data sets for analyzing. As a proficient analogous computing programming representation, MapReduce and Hadoop are employed for extensive data analysis applications. However, MapReduce still suffers with performance problems and MapReduce uses a shuffle phase as a featured element for logical I/O strategy. The map phase requires an improvement in its performance as this phase's output acts as an input to the next phase. Its result reveals the efficiency, so map phase needs some intermediate checkpoints which regularly monitor all the splits generated by intermediate phases. MapReduce model is designed in a way that there is a need to wait until all maps accomplish their given task. This acts as a barrier for effective resource utilization. This paper implements shuffle as a service component to decrease the overall execution time of jobs, monitor map phase by skew handling, and increase resource utilization in a cluster.

Keywords MapReduce · Hadoop · Shuffle · Big data
Data analytics · HDFS

1 Introduction

The objective of data analytics is scrutinizing, cleansing, renovating, and molding of the data for extracting functional information, portentous termination and sustaining choice making [1]. Data analysis has various sides and looming methods beneath diverse identities in special business, science and social science fields [2].

Big data is a meticulous technique of data analysis that focuses on analyzing huge data sets which materialize from various fields of intensive informatics data

J. V. N. Lakshmi (✉)
AIMS Institutes of Higher Education, Peenya, Bengaluru, Karnataka, India
e-mail: jlakshmi.research@gmail.com

S. C. Satapathy et al. (eds.), *Smart Computing and Informatics*, Smart Innovation, Systems and Technologies 77, https://doi.org/10.1007/978-981-10-5544-7_21

centers [3]. Big data typically comprises of data sets of massive volume beyond the skill of traditional software tools to analyze, handle, and process the data [4].

Procedures written in this practical way are mechanically parallelized and implemented on an immense cluster of commodity equipment [5, 6]. In program execution, runtime structures are concerned of the splits which are scheduled in handling many operations such as implementation across set of machines, managing failures, and handling inter-machine communications [7]. The crucial drawback is exhibited on Hadoop performance affecting the cluster.

The significant explanation of Hadoop is outlined as below:

(1) Distinct phases are leaped into a single task—the implementation of reduce function is CPU intensive and memory intensive as to segregate the map task data and produce the absolute outcome.
(2) Arbitrary requests from I/O effecting the shuffle phase—task tracker receives plenty of I/O reading requests. Each request will prompt plenty of I/O reading operations with different offset on the task tracker.

In this paper, an attempt is done to extricate shuffle phase from reduce task and instrument it as a standard resource provider. Integrate the shuffle service with sequential read policy and handling partitioning skew in reduce task to manage stragglers. Section 2 portrays the background and Hadoop MapReduce programming model, and Sect. 3 describes the problem statement. Section 4 discusses design process, and Sect. 5 analyzes on improvement in map phase. Section 6 involves the evaluation of algorithm, and Sect. 7 reviews the results. Finally, Sect. 8 concludes.

2 Background

The recent efforts from Hadoop MapReduce features are analyzed in improving performance are illustrated as follows:

(1) Map step, reduce step, the sort and merge step are included in Google MapReduce model implemented by Hungchih yang, Ali Dasdan et al.
(2) An architectural combination of MapReduce and database technologies resulted as HadoopDB is developed for analytical workloads.
(3) Hadoop MapReduce HDFS layer is replaced with concurrency optimized data storage layer which improves efficiency of data accessing concurrency, proposed by B Nicolae, G Antoniu et al.
(4) A pipeline architecture was proposed by N Conway, T Condie et al., which supports online streaming for many networking sites
(5) Resource manager and scheduler are alienated into separate components by YARN from Apache for solving the blockage of job tracker.

2.1 MapReduce

A data flow standard such as MapReduce is widely used for parallelizing the data on various applications [8]. This is a simple and open data flow programming model preferential when compared over usual high-level database approaches. This training model is used for processing large-scale datasets in computer clusters by exercising two function map ( ) and Reduce ( ). The functions Map ( ) and reduce ( ) are as follows:

$$\text{Map}(\text{K1},\ \text{V1}) \rightarrow \text{list}(\text{K2},\ \text{V2}) \quad \text{Reduce}(\text{K2},\ \text{list}(\text{V2})) \rightarrow \text{list}(\text{V2})$$

The Map ( ) functions uses key/value pair as input generating the intermediate key/value pairs. The generated intermediate key/value pairs are the input given to reduce function to produce final output [9].

2.2 Hadoop

Hadoop executes shuffle as a component of reduce task because of which there is high utilization of bandwidth in the cluster, resulting low usage of processor and unproductive performance [10, 11].

Hadoop distributed file system (HDFS) provides high throughput access to application data, resource allocation task in cluster and high unsystematic disk I/O requests are suitable for application that has large data sets [12].

From the Fig. 1, data in a Hadoop cluster is busted down into minor portions and circulated all through the collection, where a job tracker keeps track of jobs in both parent and child segments. The map and reduce functions can be implemented on

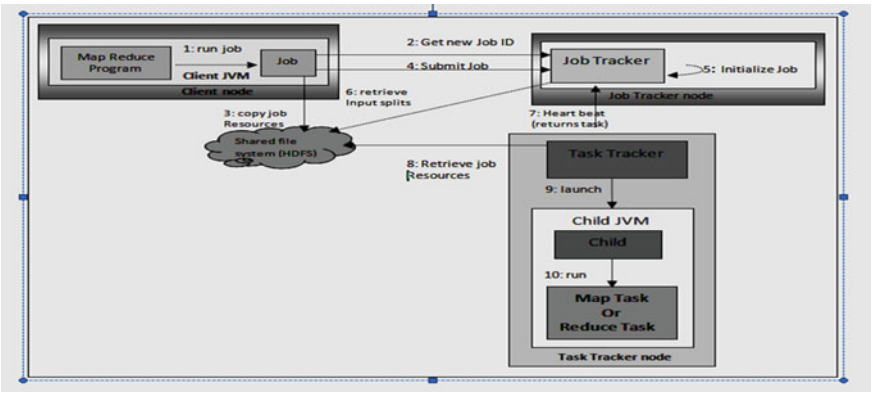

Fig. 1 MapReduce model

slighter subsets of your larger data sets, and this provide the scalability metrics that is needed for data processing [13].

3 Problem Statement

MapReduce Programming Model is very simple but as it processes, we come across many problems in map ( ) function. Map ( ) function is assigned with each split if one split cannot execute with any problem (or) if one split fails then we cannot compute the result of Map ( ) function. As combining the individual result of each map function is assigned as input to reduce function, Map function should perform in better way [14].

Two tasks associated with improved reduce phase are shuffle part and reduce part. The initial shuffle segment calls for transitional outcome from map phase. This necessitates more buffer area for various operations sorting and mapping to elicit output.

Numerous disk I/O requests from shuffle phase result in inefficient usage of resources. The above-specified reasons lead to cluster performance problems. The analysis shows an improvement in certain phases of Hadoop MapReduce specifically in terms of execution [15, 16].

4 Design Process

Shuffle and reduce as individual stages of tasks: Primarily remove copy and merge operations of shuffle from reduce as an entity splits.

4.1 Joining of Unusual Splits into Solitary Task

The shuffle phase fetches the transitional outcome from each and every map task where as the reduce function could not start its processing until shuffle phase releases the processed output data. This wastes the CPU resource time and decreases the network bandwidth.

4.2 Random I/O Request of Shuffle Task

Each map task needs to read facts from disk and transfer the response to defined reduce task instantly. This results in large amount of random disk I/O operations which in turn reduces the performance.

4.3 Design

Our features mainly involve the following stages to increase enhancement. Shuffle can process meager data improving the resource utilization efficiently within the same amount of time but the disk I/O request is progressively increased.

Fig. 2 describes the various stages of improved MapReduce architecture by implementing the technique of disjoint maps with skew in them. They are handled separately by the task tracker in slave node. The usage of generate function improves the shuffle phase and processes the data to reduce task.

Services from Shuffle: By implementing shuffle as service, resource utilization has been incremented because light weight common service relocates the on command for reduce task as a service [17].

Overcoming stragglers: To avoid blocking of slots, a skewed task is recognized and implemented. A skewed task is identified and accomplished them under similar task master. It detects partitioning skew before shuffling of data begins by monitoring data sizes produced by map and handles it by dynamically creating multiple reduce task per skewed partition [18].

Managing disk I/O requests in map phase: The I/O requests from different disk drives are processed within certain interval of time. These requests are sorted and grouped into a sequential list, forwarding them to respective output files of map tasks [19]. Task tracker reconstructs responses by reading data emerging from disk and sending their responses to reduce task in order.

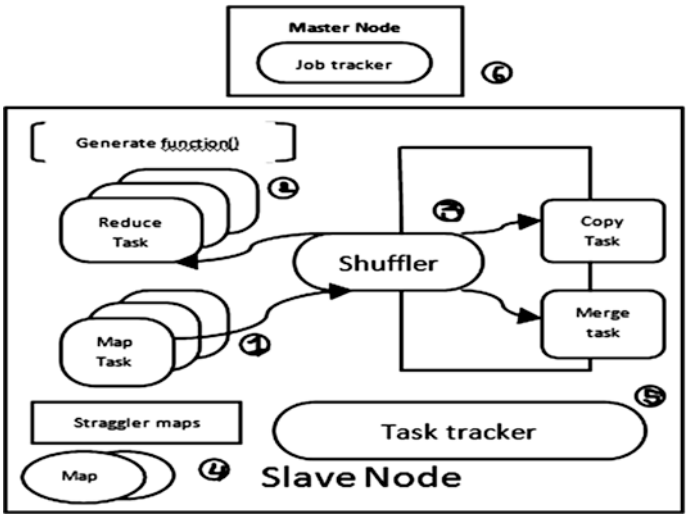

Fig. 2 Improved shuffle phase and handling straggler

5 Improving the Map Phase

From the problem statement discussed, there is a need to improve the map ( ). The function needs some checkpoints which monitor the function regularly and try to solve when a split intimates an interrupt.

Algorithm: Improving Shuffle service and avoiding stragglers in Map task.

1. Map phase runs the map task
   //each map task does the job
   //assigned by the Job Tracker
2. if(collision=true)
3. Generate(); //Straggler detection and removal and partition the task with skew
   //collision is an interrupt due to which map task cannot give an output
   // if there is a collision it calls generate method
4. Map() /* After the map processing the task tracker initialize the shuffle task launching the services*/
5. End if
6. Regular Check_points()
   // to verify proper execution
7. // Managing Disk I/O request by processing
   Read_ratio, read_time, file_size, read_total methods
8. Map task output is transferred to Shuffle phase
9. Shuffle sorts and merges the generated output
   st= dr/bw
   st*: shuffle time for reduce task*
   dr*: data to be shuffle per reduce*
   bw*: band width between nodes.*
10. This transfers the results to Reduce()

The generate function monitors the map phase at regular checkpoint and views the status of each map split. These checkpoints are arranged dynamically and access the needs of the splits. Distributed storage structure shares information among different tasks. The above algorithm specifies the design in Fig. 2, and various phases of handling the stragglers and handling the resources efficiently are elaborated.

The map jobs are scheduled in a queue, and reduce jobs use priority queue structure. In this way, interpretation of result from the map ( ) results intermediate key/value pairs [20]. These pairs are given as an input to the reduce function, and after interpretation, we generate the final output. The generate function is also used even in reduce phase. Dynamically arranged checkpoints monitor the reduce phase and split into smaller splits when an interrupt occurs. It finally combines all the split's output for obtaining the final result.

6 Evaluation

Presenting the performance and resource utilization of MapReduce jobs by implementing the shuffle service can be analyzed below:

6.1 Simulation Experiments

Intricate methods, routine calls, resource requirements, etiquette, and exchanges in the Hadoop cluster influence the ratio of disk read/write operations. Because of these random requests, there is a decrease in the regular reading ratio on disk and by which there is an increase in reduce task time.

6.2 MapReduce Job Experiment

Pi estimator utilizes more of CPU computations so it is CPU-intensive task where as word count and TeraSort are resource oriented. The resources are memory and band width.

6.3 Straggler Handling

The techniques are employed in distributing the reducers with even number of map outputs in parallel, ensuring there are no skews.

6.4 Settings

The configuration of our Hadoop with 0.24 version requires 12-node cluster among one is a master and remaining ten are slaves. Every node in a cluster uses core processor organizes 2 GHZs 4 GB of Ram and 500 GB disk drive.

Read total: Sum of read operations per each reduce task.
File size: Volume of information produced by each map task in core state.
Read time: Mean time between transfer requests and to obtain data for process.
Read ratio: Average ratio of read total to read time.
Local read total: Word count of reduce task with comparison of job time.
Reduce_skew: Stragglers count in reduce task.

7 Results

Reading performance is verified with an improved fetch phase with varied file sizes such as 128, 256, and 512 MB than the earlier fetch task. If sequential strategy on read operation is applied, then mean increase in read ratio is 94.17%, and if concurrent strategy is applied, then the ratio is 62.81%.

Figure 3 shows the data-read in local mode by varying in their speed of accessing. The data read is measured as 128, 256 and 512 MB per second avoiding stranglers. Drawing the result from word count showing the diminishing time utilization of reduce phase from 8.94% to 6.32%.

The reduce phase utilization of resources and the word count are low as the data from map phase has to release the entire output. Best word count is observed in Fig. 4 as it visualizes the read and write ratios on disk. After the necessary modifications done for the shuffle phase, the graph illustrates the improvement by showing 7% increase in resource utilization.

8 Conclusion

MapReduce programming model requires improvement in map phase as well as in shuffle phase. Though it is simple, but while implementation some complications are observed at map phase. If one map fails, it cannot compute the output as the result of map phase is an output for reduce phase.

The reduce phase adds a scheduler for every node. So, by using generate function which dynamically monitors the reduce phase will solve the basic problem in map phase. Cluster resources are well utilized efficiently when data is huge for processing transitional information then shuffle is determined as a service with minute amounts of time.

Hadoop MapReduce uses word count and TeraSort which acts as an added advantage for performance enhancement with different data structures. Resource deployment, absolute time usage are perfection features observed in skew handling technique.

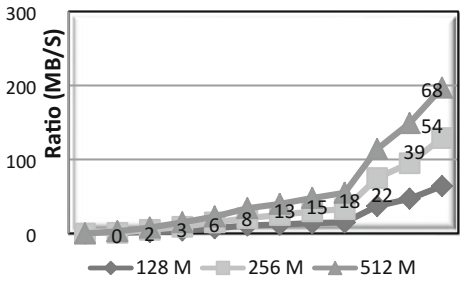

Fig. 3 Read total (MB) local read

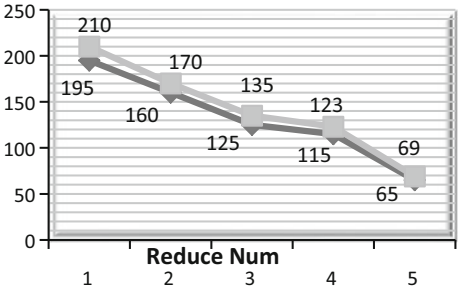

Fig. 4 Word count comparison of reduce task

References

1. Arulmurugan, A., Srinivasan, R.: Enhanced task scheduling scheme for Hadoop MapReduce systems. In: IJETCSE, May 2015
2. Dimitris, F., Ioannis, M.: Scheduling MapReduce Jobs and Data Shuffle on Unrelated Process. MIT, Cambridge (2015)
3. Pavloet, A.: A comparison of approaches to large-scale data analysis. In: Proceedings of ACM SIGMOD, vol. 5, pp. 367–378 (2009)
4. Yandong, W., Yu, W., Que, X.: Virtual shuffling for efficient data movement in MapReduce. In: IEEE Transitions on Computers Conference, June 2015
5. Luiz, A.B., Jeffrey, D., Holzle, U.: Web search for a planet: the Google cluster architecture. IEEE Micro **23**(2), 22–28 (2003)
6. Huston, L., Wickremesinghe, R., SatyaNarayana, M.: Storage architecture for early discard in interactive search. In: FAST Conference Proceedings (2004)
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters in Google, Inc OSDI (2004).
8. Lakshmi, J.V.N., Ananthi, S.: A theoretical model for big data analytics using machine learning algorithms. In: ICACCI Conference, Delhi, October 2015
9. Kwon, Y.C., Howe, B.: A study of skew in Map Reduce application. In: International Conference, USA (2014)
10. Alan, F.G., Olga, N., Shubham, C., Pradeep, K., Shravan, M.N.: Building a high level dataflow system on top of MapReduce: the pig experience. In: IEEE Conference (2009)
11. Yanfei, G., Jia, R., Xiaobo, Z: IShuffle—improving Hadoop performance with shuffle-on-write. In: USENIX ICAC, USA (2013)
12. Abouzeid, A., Bajda, P., Abadi, D.J., Rasin, A., et al.: HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. PVLDB **2**(1), 922–933 (2009)
13. Ananthi, S., Lakshmi, J.V.N.: A study on Hadoop architecture for big data analytics. In: Delhi Conference ICETSCET, September 2014
14. Herodotos, H., Lim, H., Luo, G.: StarFish—a self tuning system for Big Data Analytics, CIDR, USA (2011)
15. Ronnie, C., et al.: SCOPE: easy and efficient parallel processing of massive data sets. In: Proceedings of VLDB (2008)
16. Ashish, T., Joy deep Sen, S.: HIVE—a warehousing solution over a MapReduce framework. In: VLDB (2009)
17. Li, J., Ye, Y.: Improving the shuffle of Hadoop MapReduce. In: Proceedings of IEEE ICCCTS (2013)
18. Li, J., Yue, Y., Lin, X.: Improving the shuffle of Hadoop MapReduce. In: IEEE ICCCTS, Beijing, China (2013)

19. Prateek, D., Sriram, K., Janakiram, D.: Chisel: resource savvy approach for handling skew in MapReduce application. In: IEEE Conference on Cloud Computing, vol. 35, pp. 45–56 (2013)
20. Dean, J., Ghemawat, S.: MapReduce: a flexible data processing tool. ACM Commun. **53**, 72–77 (2010)