

Enhancing Dataset Processing in Hadoop YARN Performance for Big Data Applications

Ahmed Abdulhakim Al-Absi, Dae-Ki Kang and Myong-Jong Kim

Abstract In Hadoop MapReduce distributed file system, as the input dataset files get loaded and split to every worker, workers start to do the required computation according to user logic. This process is done in parallel using all nodes in the cluster and computes output results. However, the contention of resources between the map and reduce stages cause significant delays in execution time, especially due to the memory IO overheads. This is undesired because the task execution in the Hadoop MapReduce induces an overhead in considering redundant data in case of imprecise applications which increases the execution time. Thus, in this paper we present our approach to optimize local worker memory management mechanism to reduce the presence of null schedule slots. Efficient utilization of slots leads to reduce execution times. The local memory management mechanism adopted enables efficient parallel execution and reduced memory overheads. The approach effectively reduced the MapReduce computation time which minimizes the budget for application execution in the cloud.

Keywords Dataset · Hadoop YARN · MapReduce · Big data · Cloud computing

A.A. Al-Absi (✉) · D.-K. Kang

Division of Computer and Information Engineering, Dongseo University, Busan, Korea
e-mail: absiahmed@gmail.com; ahmed_absi2005@yahoo.com

D.-K. Kang

e-mail: dkkang@dongseo.ac.kr

M.-J. Kim

School of Business, Pusan National University, 63 Beon-gil 2, Busandaehag-ro, Geumjeong-gu, 609-735 Busan, Korea
e-mail: mjongkim@pusan.ac.kr

1 Introduction

With the increase in population and beneficiary of internet services, data size is getting increased day by day where 100s of quadrillion of data files are there in cloud available in unstructured nature [1]. On the other hand the application of data-insensitive applications does need certain optimum approach to manage these data files and retrieve the data even without mammoth task and complexity. Various applications like IaaS and PaaS do need the applications which can be effective for providing optimum data access in real time operations. Taking into account of these all circumstances, now days, a number of research works being going on, and Hadoop was one of the significant outcomes that is being used extensively for cloud frameworks.

Hadoop [2] is open-source software in the form of a highly scalable and fault tolerant distributed system which plays a very significant role in data storage and its processing. This framework Hadoop encompasses two dominant parts, first Hadoop distributed file system (HDFS) while second refers for MapReduce. HDFS is the mechanism to classify data on nodes or clusters and provide an interface for data management between users, tasks trackers and nodes or machines. The space where there is certain optimization scope is MapReduce.

According to [3] the main issue with MapReduce framework is its batch-processing oriented nature, which means stateless mapper followed by a stateless reducer, that are executed by a batch job scheduler. This paradigm makes repeated querying of datasets difficult and imposes limitations. Moreover, the process of mapping and converting to intermediate combiner is a time consuming and need to be optimized. In cloud context and due to the cloud heterogeneous behavior existing between the central servers and storing disks, there must be something parallel architecture that could enhance the processing speed and data retrieval rate in MapReduce [4].

A number of researches like in [5, 6] have been done for optimization for MapReduce framework, but still there exists a huge scope for further optimization with diverse cloud platform to come up with the optimum cloud computing model. Taking into consideration of these factors, here in this paper, we introduce a parallel execution model based on MapReduce framework. The major contributions of the parallelized model are in the local worker memory management mechanism and the optimization technique adopted to reduce the presence of null schedule slots. The worker nodes i.e. the Map workers and Reduce workers operate based on the slots assigned. Efficient utilization of slots leads to reduced execution times. The local memory management mechanism adopted enables efficient parallel execution and memory overhead reduction. As in Hadoop our approach also considers the data in chunks. The Map Workers in MapReduce framework perform computations on the chunks of data. In our work, these chunks of data are further split to enable parallel execution in the Map Worker nodes.

2 Hadoop Scenario and Proposed Solution

2.1 Motivation

In MapReduce programming model, Hadoop runs MapReduce files in the form of (Key, Value). Converting these input text files into form of (Key, value) requires passing the values from the input split to mapper by one more pre-defined interface called Record Reader [7]. The key of the split file is associated with each line by its byte offset in which the Record Reader is invoked repeatedly on the input until the entire InputSplit file is completed whereas each invocation of the RecordReader leads to another call to the map() method of the Mapper and store the intermediate result into the combiner [7].

The main issue is that the contention of resources between the map and reduce stages causes significant delays in execution time, especially due to the memory IO overheads. As the Hadoop map stage is initially completed, the reduce task is performed. This kind of a serial execution mode can hinder execution performance. MapReduce Map processes are initiated sequentially as splits the dataset into small chunks. The map and reduce workers perform their tasks in their pre assigned slots. The presence of ideal or unutilized slots affects system performance. The task execution in the Hadoop MapReduce induces an overhead in considering redundant data in case of data-intensive applications which increases the execution time. This is undesired for a number of reasons because it requires more storage and consumes more computation time. Therefore, the next section addresses this issue by a solution to optimize map computation time and improves the storage efficiently.

2.2 Proposed Solution

In Hadoop MapReduce, the contention of resources between the map and reduce stages cause significant delays in execution time, especially due to the memory IO overheads. This is undesired because it requires more storage and computation time. Thus, to enhance performance of MapReduce and resource utilization, we present our approach to optimize MapReduce performance by data reduction technique. In our approach, the map and the reduce slots assignments are optimized to minimize the occurrence of null slots or unoccupied slots. The Map Workers receive the chunk data from the cloud storage and store the data in the local cache memory. The received data in MapReduce is further split into parallel data blocks in our approach. Figure 1 illustrates our parallel data model in Hadoop MapReduce.

As is noted, each data block is executed in a parallel fashion. In addition to the parallel execution, redundant and previously computed data is eliminated from the local memory to achieve reduction in execution time and storage overheads. It is observed that the local memory available with the Map Workers reduces in size as the execution time increases. The reduction in the local memory is dependent on the

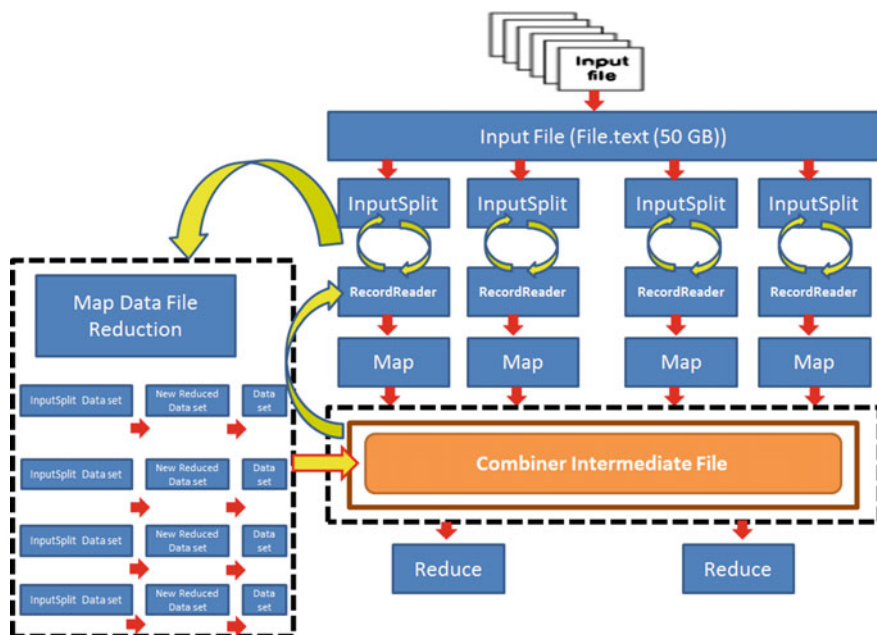


Fig. 1 The proposed data reduction in Hadoop MapReduce

actual computation function. The Map Workers store the intermediate results in the cloud storage. The Reduce workers utilize the intermediate data to perform the reduce tasks. For example, in wordcount, suppose we have a file contains A to Z, if worker-1 has done the computation for “A”, the file in which the next worker will get is the file with “A” already removed. Workers will do their individual work and as soon as one worker finishes its job it goes to the next level in parallel processing. The words are eliminated and stored in central store. Therefore, the loading and computation process will also be expedited in our approach. Figure 2 presents our data reduction in Hadoop MapReduce.

3 Performance Evaluation

3.1 Experiment Setup

In order to implement our system to optimize Hadoop MapReduce performance by data reduction, we considered Microsoft Azure cloud platform for our developed and optimized Hadoop implementation. Here we implement Microsoft Azure HDInsight technology for creation of virtual machines.

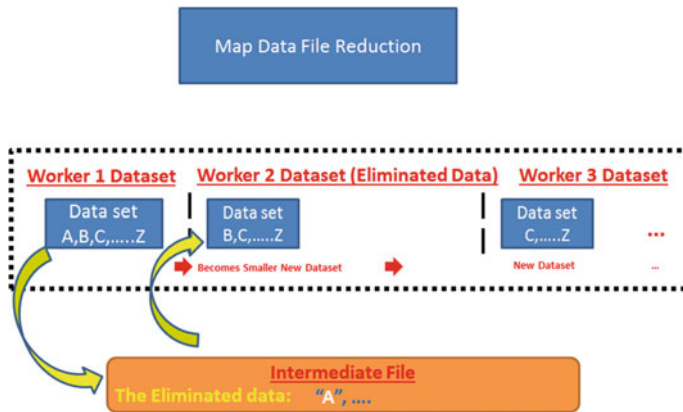


Fig. 2 The proposed data reduction process

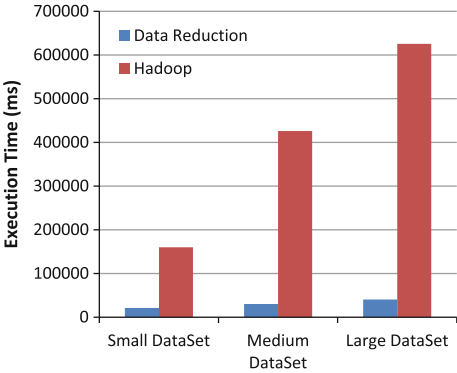
Wikipedia dataset [8] were downloaded as text files for our work. These text files converted into 10 small chunk files, where each file was just over 25 MB. Each Map task processes one of these chunk files. This resulted in a total dataset size of approximately 250, which was copied to the local storage as well as to Azure blob storage service. We used three datasets for Wordcount application benchmark (Small S 100 MB, Medium M 200 MB, Large L 250 MB). The datasets used to test MapReduce loaded input files execution times issue on the wordcount example jobs that come with the Hadoop distribution. We used wordcount application as a benchmark for our experiment and we installed our work on the ongoing development of Apache Hadoop YARN (Yet Another Resource Negotiator) version 2.4.0.2.1.3.0.

3.2 Evaluation Result

This section presents the evaluation results for our work in enhancing MapReduce using the data reduction. The experiment has been executed for the purpose of evaluating the Map phase aiming to observe the Hadoop map phase execution time before and after applying data reduction approach. We sequentially have executed several requests for the word count application and on each request, we have checked the execution time of map's total time spent by all maps in occupied slots in milliseconds (ms). Figure 3 represents the execution time of our work in Microsoft Azure for wordcount with 4 workers when we vary the input dataset size.

From Fig. 3, it is clear that the execution time increases as the size of input data scales. Furthermore, Hadoop acquires poor performance compared to our work with data reduction. It is because MapReduce's performance is affected due to the contention of resources between the map and reduce stages which cause significant

Fig. 3 Workers over all execution time performance comparison of “wordcount” benchmark between Hadoop and data reduction approach in different datasets



delays in execution time, especially due to the memory IO overheads which affect I/O performance and require more storage. Note that our approach has effectively reduced the MapReduce computation time by considering the input data computation in a parallel fashion. The local memory management aids in memory overhead reductions. The optimization strategy to reduce the occurrences of unutilized slots improves the execution efficiency.

Figure 4 reports the wordcount execution with data reduction compared to traditional Hadoop as we scale the number of workers in small dataset. In this graph, the less the execution time the higher the performance.

We can find it clear from Fig. 3 that the reduction of the execution times had a big impact on datasets inputsplit execution performance. The reduction approach was able to successfully complete execution of the Map before the Hadoop with the scalability of workers in different input file size small, medium and large datasets.

From the experiment results, we conclude that the data input loading in MapReduce is an important factor in terms of I/O performance on cloud environment. The contention of resources between the map and reduce stages cause significant delays in execution time, especially due to the memory IO overheads. In

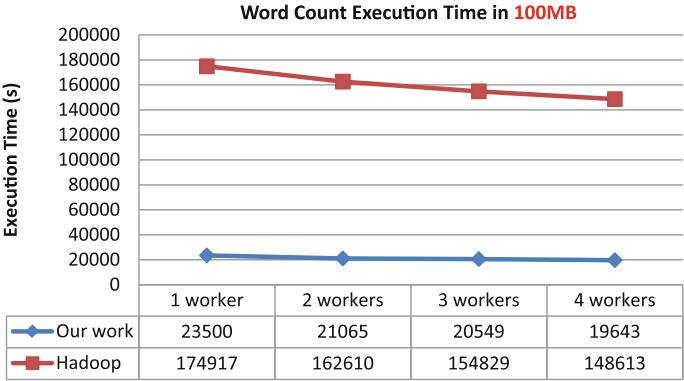


Fig. 4 Scale Wordcount Execution Time for small dataset input file

our approach, The Map Workers receive the chunk data from the cloud storage and store the data in the local cache memory. The received data is further split into parallel data blocks and therefore our work approach outperforms Hadoop by a factor of 7.44 in wordcount application. Our approach has provided faster computation time with less storage, which minimizes the budget for application execution in the cloud.

4 Conclusion and Future Work

In Hadoop based cloud computing systems, the contention of resources between the map and reduce stages cause significant delays in execution time, especially due to the memory IO overheads. This is undesired because the task execution in the Hadoop MapReduce induces an overhead in considering redundant data in case of imprecise applications which increases the execution time. Thus, this paper presented our approach to optimize local worker memory management mechanism to reduce the presence of null schedule slots. The approach effectively reduced the MapReduce computation time which minimizes the budget for application execution in cloud.

For future work, we are planning to analyze and compare the performance of our work in other high performance computing application like gene sequencing, sequence matching, and page ranking.

Acknowledgment This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. NRF-2013R1A1A2013401).

References

1. Gupta, P.K.: Introduction to Analytics and Big Data/Hadoop. Implementing Information Infrastructure Summit (IIIS). Marina Mandarin, Singapore, 30 May 2013. http://issuu.com/fairfaxbm/docs/cws_jul-aug2013/17
2. <http://hadoop.apache.org/>
3. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. *Commun. ACM* **51**(1), 107–113 (2008)
4. Kolb, L., Thor, A., Rahm, E.: Load balancing for mapreduce-based entity resolution. In: 2012 IEEE 28th International Conference on Data Engineering (ICDE), pp. 618–629 (2012)
5. Luo, Y., Guo, Z., Sun, Y., Plale, B., Qiu, J., Li, W.: A hierarchical framework for cross-domain MapReduce execution. In: Proceedings of ECMLS, pp. 15–22 (2011)
6. Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., Stoica, I.: Improving mapreduce performance in heterogeneous environments. In: OSDI. USENIX, pp. 29–42 (2008)
7. <https://developer.yahoo.com/hadoop/tutorial/module4.html>
8. Thottethodi, M., Ahmad, F., Lee, S., Vijaykumar, T.N.: Puma: Purdue mapreduce benchmarks suite. Technical Report, Purdue University (2012)

Advanced Multimedia and Ubiquitous Engineering

Future Information Technology Volume 2

Park, J.J.; Chao, H.-C.; Arabnia, H.R.; Yen, N.Y. (Eds.)

2016, XXVI, 513 p. 233 illus., Hardcover

ISBN: 978-3-662-47894-3