Boosting the Performance of MapReduce by Better Resource Utilization in Cluster

Pooja Malikwade
M.E.(II year)
MMMCOE
Pune-52

S.B.Jadhav
Assistant Professor
MMMCOE
Pune-52

ABSTRACT
MapReduce implementations are being used for processing large data sets. MapReduce performs parallel computations to speed up the job processing. When performing parallel computations the skew that arises due large indivisible records or uneven distribution of data slows down the job execution process and lowers the cluster throughput. We provide a solution, by proposing an automatic system that handles skew which is compatible with MapReduce framework and is transparent to users. The proposed system makes use of idle resources in the cluster for skew handing. Task repartitioning method is implemented for the purpose of skew handling. The output order is maintained even after task repartitioning. The proposed system requires no extra input from the users and imposes minimum overhead in the absence of skew.

General Terms
Big Data, Hadoop

Keywords
Data skew, MapReduce, parallel database systems, performance gain, skew handling.

1. INTRODUCTION
Google proposed MapReduce framework in 2004 [1] and has been widely used for processing and analysing enormous amount of data. MapReduce is a powerful tool for parallel analysis. It has various implementations. The framework supports distributed processing of data, is tolerant to node failures in the cluster and is scalable. MapReduce provides an API for writing user defined operations. The users need to only write the map and reduce functions. The framework is responsible to run these functions in parallel in a cluster. MapReduce functionality is performed in five steps: a) preparing map() input b) running user defined map() code c) shuffling the intermediate map output, which will be input to the reducers d) running user defined reduce() code e) final output i.e. collecting and sorting all the reducer outputs. Figure1 shows the Mapreduce method.

Hadoop is an open source implementation of MapReduce. Hadoop is a software framework for storing and processing large amount of data. It has HDFS (Hadoop Distributed File System). This distributed file system is used to store data across various machines in a cluster. It automatically handles the hardware failures. It follows master slave architecture.

The case when load imbalance arises is known as skew. Load imbalance can occur during map or reduce phases. Skew is a well-known problem in the context of parallel database management systems. Presence of skew hinders the job execution rate i.e. it takes longer to execute the job and thus decreases the cluster throughput. MapReduce does not take care of skew efficiently.

Reasons for skew

i Large records: During the map tasks records are processed as key-value pairs. Some records may be more CPU intensive and may require more computations than other records. These large records further are indivisible which leads to skew. As the tasks processing these large records take longer time to finish.

ii Input dataset size: Some tasks take multiple dataset as a single input. Each dataset requires different processing times. Some dataset require larger processing times.

iii Reducer input: Intermediate output i.e. outputs of map tasks are distributed to the reducers using hash partitioning. This partitioning does not guarantee even data distribution to the reducers.

iv Large input: Large indivisible records processed by map tasks become input to one of the reducers, skew arises.

One way to handle skew is to adopt the approach of skew avoidance. This requires the users to write user defined operations in a manner which does not arise any skew [13]. But this approach is ineffective as it imposes extra burden on users and is applicable only to very few operations that satisfy certain properties. Another approach is to divide the job into very small fine-grained partitions and allocating to the machines as needed [14]. But this strategy imposes significant overhead due to frequent migration of data. Even though simple, skew avoidance along with above drawbacks also has the drawback that it does not always guarantee tasks without skew and also does not provide adequate resource utilization.

In this paper, we consider skew that arises from the characteristics of dataset. Speculative execution [1] technique used to handle skew in mapreduce is ineffective to handle the skew that arises from dataset. Because the technique does not improve any job execution time.

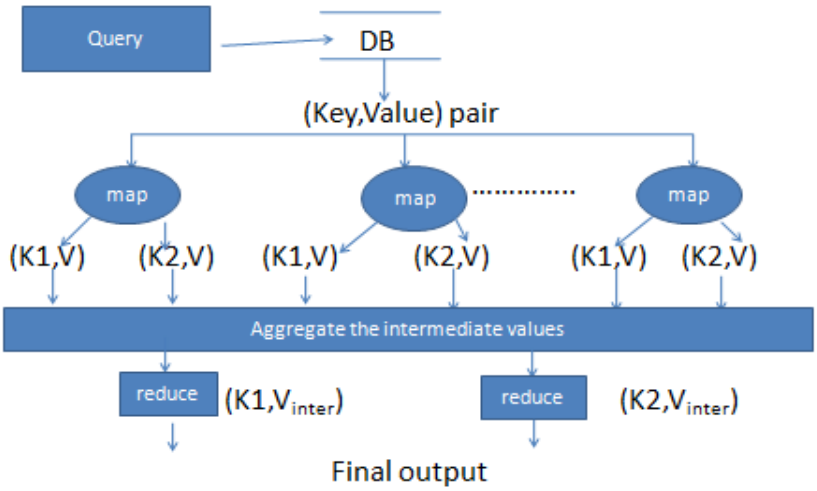


Figure 1: MapReduce Method

We propose an automatic skew handling system which has the following key features:

- Handles the skew that arises due the uneven distribution of data and also the skew that arises due to larger dataset that take longer time to process.
- Programmers need not provide any extra input.
- The system is transparent to the users.

With the proposed system we expect to reduce the job execution time by 3x times in the presence of skew and incur very little overhead in the absence of skew.

The rest of paper is organized as follows. In section 2 we discuss the related work. In section 3 we show the implementation of proposed design. Finally, we show the performance evaluation in section 4.

2. RELATED WORK

MapReduce framework is used for many applications such as web indexing, log analysis etc.[2]. When a slow task is detected which is the cause for skew, that task is backed up on some other machine so that it can finish faster. This approach

is known as speculative execution. Speculative execution technique provides performance gains and hence is implemented in Hadoop [3] and Microsoft Dryad [4] to deal with skew.

Hadoop is an open source implementation of MapReduce. The speculative execution technique in Hadoop 0.20 implementation, detects a task as a skewed task when its progress rate is lesser as compared with average progress rate of all the tasks. But this strategy of skew detecting does not fit well in the heterogeneous environment. So a new strategy called LATE [5] is proposed. Hadoop-LATE selects a task based on the longest remaining progress time to backup. Hadoop-LATE and LATE assume that all tasks have same process types and same amount of input data. And also assume that progress rate of each task is stable. But these assumptions are not always true.

Miscrosoft Dryad [4] makes use of same speculative strategy as in [1] i.e., backing up the last few remaining slower map or reduce tasks. Mantri's [6] speculative strategy is based on estimating task's remaining time by identifying data left to be processed using process bandwidth.

The above strategies have certain drawbacks in identifying skewed tasks and the backup nodes. They assume that tasks have stable progress rates. But in reality, the progress rates fluctuate in the different phases of the map and reduce tasks. When choosing the backup nodes Hadoop and LATE do not consider whether the skewed tasks can finish faster on the backup node than on the original node.

Mantri [6] kills the slower task when the cluster is busy and when the cluster is idle duplicates the complete task and runs from the scratch. In [7] a new speculative execution strategy called Maximum Cost Performance is suggested which overcomes all the drawbacks of the above techniques. It uses both the progress rate and the process bandwidth to determine skewed task, uses a cost benefit model for backing up the tasks and also considers whether a skewed task can be executed faster on the backup nodes.

But all the above speculative execution strategies just provide with 2% to 25% of performance improvement according to [2]. This is because most of the skewed tasks are compute intensive, do not provide much performance gain. Skew handling techniques are classified into two classes: skew avoidance and skew detection and handling systems. In skew avoidance systems, skew is avoided in tasks before starting their executions. Skew avoidance has the advantage of no task repartitioning, but it cannot always guarantee complete skew avoidance i.e. there is presence of some skewed tasks and also resource utilization is limited. In [8] skew is considered only during the map tasks and is implemented in a simulator and not on a real cluster. The proposed system automatically detects and handles the skew during both map and reduce tasks.

3. SYSTEM DESIGN

Each map() and reduce() function invocations are independent of each other. The task with the longest remaining processing time is known to be skewed task and remaining unprocessed input bytes of that task at the time say 'i' (i.e. the time at which skew is detected) are repartitioned to run on idle nodes in a cluster.

3.1 Selecting skewed task for repartitioning

When a job is submitted to MapReduce framework, it is divided into 'N' number of independent tasks, which can be run on various nodes in a cluster for parallel execution. All 'N' tasks are first assigned to the slots of the nodes in the cluster. When on some node, a slot finishes the execution of assigned task to it that slot becomes idle, only then the skew detection method is invoked. Slave nodes estimate the progress of each task tprogress [9] and are periodically reported to the master node. For the estimation of tprogress each task has to keep track of total input bytes and records processed. The

Step 5.Repartition only the unprocessed input data of selected skewed task.

Step 6.If the selected skewed task is last task to be processed then repartition and reprocess whole task.

Step 7.The master node assigns the repartitioned data to the available slots including the slot of the node on which the task was originally allocated.

3.2 Algorithm

//Start skew detection only when a node becomes free in the cluster

Input: R=set of running tasks

W= set of unscheduled waiting tasks

Initializations: flag=false

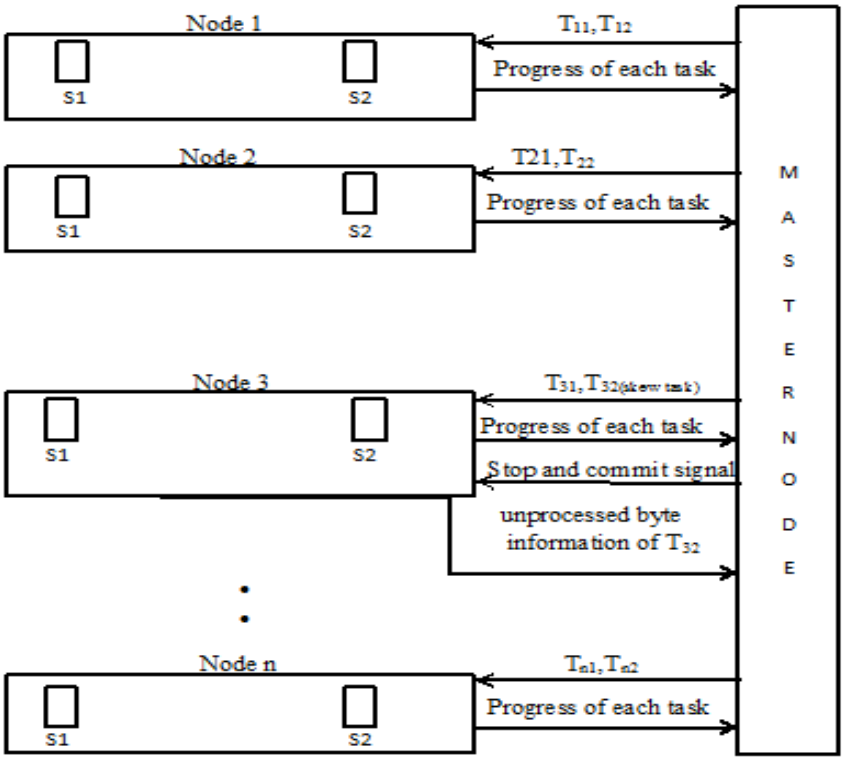


Figure 2: System Architecture

most skewed task i.e. task with the longest remaining processing time is selected and checked if its remaining processing time is greater than twice the repartitioning overhead i.e.

$t_{progress}$ >2O

where O is repartitioning overhead

Repartitioning overhead is set to 20s and if the selected task's remaining processing time is greater than twice the overhead then it is repartitioned. Repartitioning is performed on remaining unprocessed bytes. Only one skewed task is chosen at-a-time to avoid frequent repartitioning of the tasks.

3.3 Handling skew

Steps taken to handle skew are:

Step 1.Slection of a task for repartitioning

Step 2.Master node signals stop and commit to the slave node on which the selected skewed task runs

Step 3.In case the task is impossible to stop, master chooses next longest remaining time task, else goto step 4.

Step 4.Calculate the remaining input data to be processed

task=null

if W≠ϕ then

task=choosenexttask(W)

else if flag=false then

task=identifymax_time_remain(task)

if task ≠ϕ and time_remain(task)>2.O then

{ // O is overhead= 20s

stop(task)

flag=true}// skew detected

// After the skewed task is detected

orgtasktime=T_{org} from T

processedB=task till time t_i

taskUB=orgtasktime-processedB

taskUB={t_1,t_2,….t_n}// Repartition of task

execute taskUB on n+1 nodes

combine the output of taskUB

return to identify skew again if any node free.

4. PERFORMANCE EVALUATION

4.1 Setup

Real data set is used as an input to MapReduce framework. One node in the cluster acts as a master node and others as slave nodes. Nodes are connected in LAN. Each node is configured to run at most two tasks. Word count application is taken for testing purpose as it is a compute intensive application.

4.2 Manual generation of skew

Skew is generated manually during the execution of tasks. Some tasks are given larger documents or two times more input data to process. Those tasks then take longer time to execute which can further be detected as skewed tasks.

4.3 Expected gain in performance

The proposed system is expected to execute the job faster by 3x times in the presence of skew.

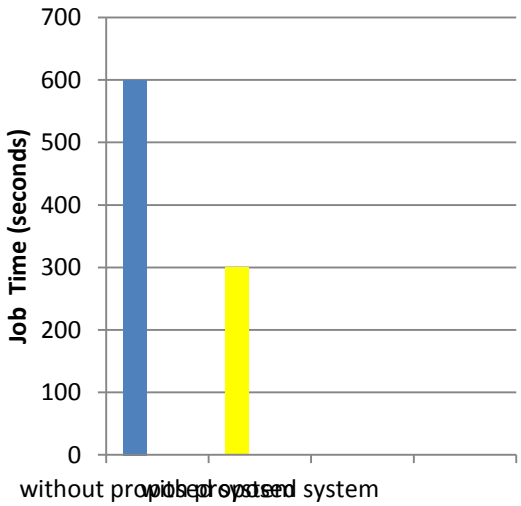


Figure 3: Expected performance gain in job execution in the presence of skew

5. RESULT SET

The following are the result sets with the detection and execution of skewed task on idle node. Figure 9 shows the performance gain when skewed task is detected and run on idle nodes and thereby improving the performance of MapReduce.

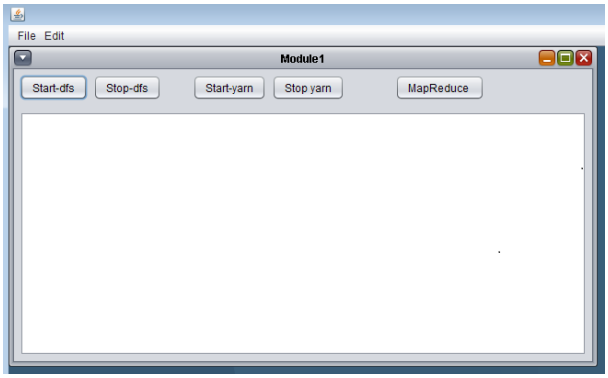


Figure 4: GUI to start MapReduce job

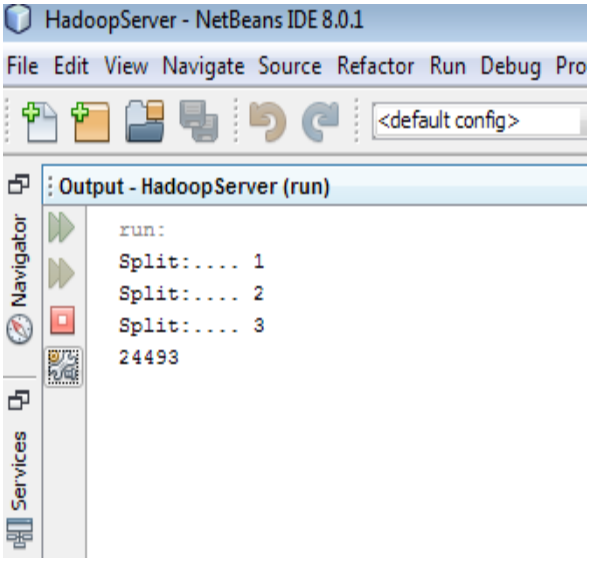


Figure 5: Master divides the task

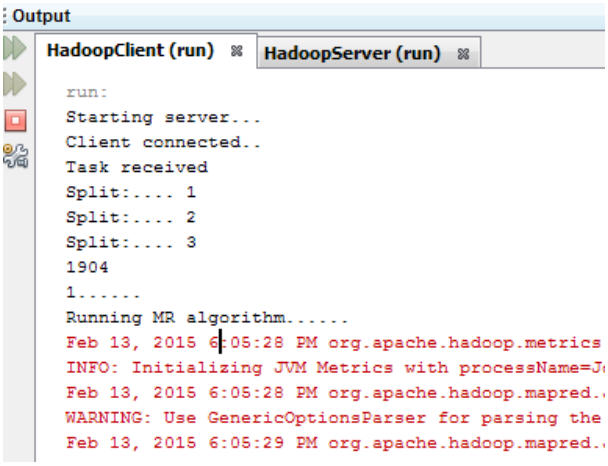


Figure 6: Task received on client side

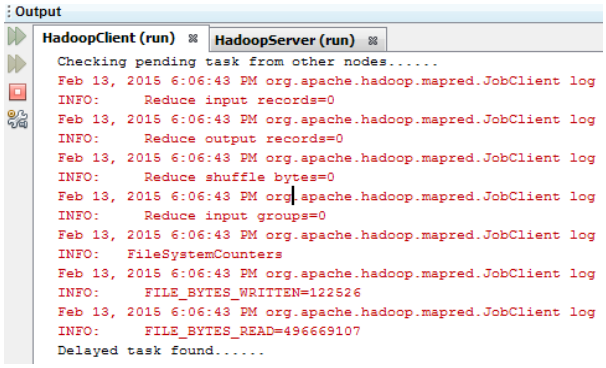


Figure 7: Idle node checks for skewed task on other nodes

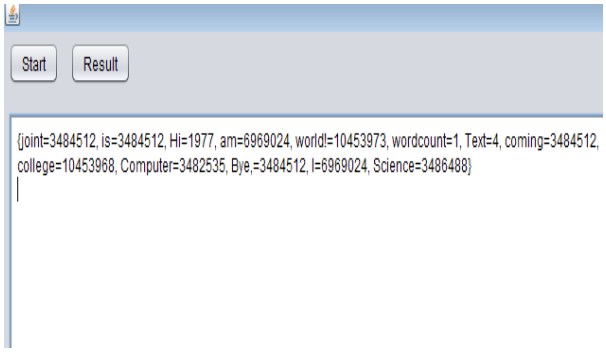
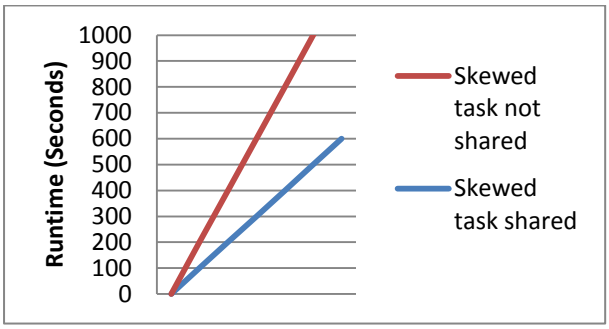


Figure 8: Final result on master



Word Count Application

Figure 9: Time taken is lesser when skewed task shared

6. CONCLUSION

Mapreduce is a parallel programming paradigm. The job execution rate slows down on MapReduce in the presence of skew. With the proposed system we can boost up the job execution of MapReduce framework. The proposed system is an automated system for detecting and handling skew and is compatible with all the systems that implement MapReduce type job execution. It requires no extra input from the users. Job execution is monitored periodically and load is re-balanced as the slots of the nodes become available. The order of output is preserved even when a skewed task is repartitioned and is also transparent to programmers i.e. programmers feel that they are working on MapReduce framework. We expect to have a 3x times faster job execution for compute intensive MapReduce jobs. In the absence of skew the system will incur little to no overhead. Further we also maximize the resource utilization of the cluster. Overall, the proposed system provides consistent job execution in the presence of skew and provides an enhanced performance gain.

Future work involves reducing the overhead incurred when skewed task is detected. Which can further improve the performance of MapReduce and also the throughput of cluster.

7. REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008.

[2] K. Ren, Y. Kwon, M. Balazinska, and B. Howe, "Hadoops adolescence: A comparative workload analysis from three research clusters," in *Proceedings of IEEE 8th International* Conference on e-Business Engineering, ser. ICEBE'2011, 2011.

[3] "Apache hadoop, http://hadoop.apache.org/."

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proc.of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07, 2007.

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. of the 8th USENIX conference on Operating systems design and implementation*, ser. OSDI'08, 2008.

[6] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proc. of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10, 2010.

[7] Q. Chen, C. Liu, and Z. Xiao, "Improving mapreduce performance using smart speculative execution strategy," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, p. 1, 2013.

[8] Z. Guo, M. Pierce, G. Fox, and M. Zhou, "Automatic task re-organization in mapreduce," in *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, ser. CLUSTER '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 335–343.

[9] K. Morton, A. Friesen, M. Balazinska, and D. Grossman. Estimating the progress of MapReduce pipelines. In Proc. of the 26nd ICDE Conf., Mar. 2010.

[10] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "Scope: easy and efficient parallel processing of massive data sets," *Proc. VLDB Endow.*, vol. 1, pp. 1265–1276, August 2008.

[11] X. Pan, J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Ganesha: blackbox diagnosis of mapreduce systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, pp. 8–13, January 2010.

[12] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-reducemerge: simplified relational data processing on large clusters," in *Proc. of the 2007 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '07, 2007.

[13] M. C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. Bioinformatics, 25(11):1363{1369, June 2009.

[14] M. Shah, J. Hellerstein, and E. Brewer. Highly-available, fault-tolerant, parallel dataows. In Proc. of the SIGMOD Conf., June 2004.