Improving MapReduce Performance through Process Migration

Rahul R. Ghule
Department of Computer Science & Information Technology,
Dr. BAM University, Aurangabad

Sachine N. Deshmukh
Department of Computer Science & Information Technology,
Dr. BAM University, Aurangabad

***Abstract*—MapReduce is widely used and popular programming model for huge amount of data processing. Hadoop is open source implementation of MapReduce framework. Hadoop MapReduce is used for large data processing. It computes large amount of data in less time. The Performance of Hadoop depends some of the metrics like job execution time and cluster throughput. In MapReduce, Job is divided into multiple map and reduce tasks. A node in Hadoop Cluster is supposed to perform multiple processes. Some process can be executed slowly due to internal or external reasons. Because of this slow process job execution time is prolonged which leads to degradation of Hadoop MapReduce's performance. To overcome this, various strategies has been proposed like speculative execution, scheduling etc. In Speculative execution, each slow task is backed up other node in order to reduce the job execution time. These slow tasks can be called as straggler tasks. However, current strategies do not take node's health in consideration. A node in cluster may be straggler rather than process. If a node becomes straggler then it will lead to poor performance of current MapReduce process. Our aim is to find performance of a node by calculating CPU load. If it is greater than threshold value then node is considered to be straggler node. So, current process on that node is backed up on other node for faster execution. Experiments results shows that our system improver MapReduce performance by 12.91%.**

Keywords: Hadoop, MapReduce, Speculative Execution, Straggler Machine

I. INTRODUCTION

In recent years the amount of data stored worldwide has been increasing very rapidly. Use of internet has been increased largely which generates enormous data. Web search engine and social networking site captures and analyze users action on their site to improve site design, detect spam and fraud. Facebook collects 15 Terabytes of data each day into its PetaByte-scale data warehouse [6]. Companies have petabytes or more data which contains valuable information for continued growth and success. However, the amount of data is too large to store and process in traditional relational databases. And hardware needed for conventional analysis is too costly. But this huge generated data will increase the demand of reliable backups and will decrease the hardware fault tolerance level. The potential gain of information from such huge amount of data i.e. BIG DATA is very large but that would not be beneficial if the hardware cost remain high.

Hadoop[12] has evolved as a distributed software platform for managing and transforming large quantities of data, and has grown to be one of the most popular tools to meet many of the above needs in a cost-effective manner. HDFS and Hadoop MapReduce framework are the integral part of Hadoop. It is master slave architecture. The master server is called as NameNode. Its job is to split file into blocks and distributes them across the cluster. Here, Slaves are called as DataNode whose job is to serve read/write operation from clients [8].

Mapreduce is proposed by Google in 2004 and is popular for parallel computing framework for large scale data processing. In Mapreduce job, master divides the files into multiple map and reduce task. Then it distributes these tasks among the different worker node for parallel processing. The input data is usually large and computation has to be distributed around hundreds of machine to finish in less amount of time [6].

But sometime tasks will take log time for completion. It will delay the total execution time. Such delayed tasks are called as straggler task. This straggler tasks degrade the overall performance of MapReduce jobs. This straggler task can be generated by external or internal reasons. It might be due to resource competition in MapReduce tasks or on machine or it can be due to faulty hardware. There different techniques to handle such problem like speculative execution. In Hadoop, it simply identifies a task as straggler when it falls behind the average progress rate of all tasks by a fixed gap [8]. LATE (longest approximate time to execute) keeps the track of progress rate and estimates its remaining time. After computation, LATE will select the straggler task that having longest remaining time. Mantri has also developed strategy to deal with straggler problem. It uses the tasks process bandwidth to calculate remaining time of the task [5]. MCP which is proposed by Qi Chen uses both progress rate and process bandwidth to indentify straggler tasks. MCP uses exponentially weighted moving average algorithm to calculate process speed. Whenever it detects a straggler tasks, it will backs it up on different worker node to execute faster [1]. Unlike these strategies, our system considers the straggler machine. It identifies the health of machine. Whenever CPU usage is above threshold value, current MapReduce process is backed up on other worker node to execute faster.

The rest of this paper is organised as follows: Section II describes basic MapReduce mechanism, causes of straggler and related work. Section III describes our design and performance evaluation.

II. BACKGROUND

A. *Mapreduce Mechanism:*

In general, MapReduce has three basic stages namely map stage, shuffle stage and reduce stage.

Map Stage- In this mapper's job is to process the input data to extract key-value pair. The input data can be a file or directory and is stored in the Hadoop file system (HDFS). The input file is processed line by line by the mapper function. Then it operates on the data and creates multiple small chunks of data. Then it extracts key-value pair from the input data.

1) *Shuffle stage*- In this stage, output of mapper is taken which is in key-value form from all the working nodes. Then shuffling and sorting of key-value pair is done and is advances to Reducer.

2) *Reduce Stage*- The Mapper's intermediate output is provided to this stage. Reducer copies those output, sort it and is given to reducer to create new set of final output.

In a MapReduce cluster, when job is submitted, a master divides the input file in different blocks according o hadoop policy. Then it schedules both the map and reduce task to multiple working node called as worker node. A worker node on which task is running keeps updating the task's progress by periodic heartbeat. Map tasks creates key-value pair from input file and then it is transferred to some user define map function and combiner. After that intermediate output has been created which is provided to multiple reducers. Reducer copies those intermediate output from mapper and then merge all key-value pairs in a stream. This stream is then transferred to user define reduce function. Then finally, output from all reducer is combined into a single file [9].

B. *Causes of Straggler:*

A task is said to be straggler when it is taking more time to execute. These tasks degrade the total execution time a job. There can be internal and external factors due to which straggler tasks are generated. In real world, MapReduce cluster process multiple tasks running on the single worker node. This thing can lead to resource competition among the tasks. This will create slow tasks. The straggler tasks can be generated due to heterogeneous data, input data skew or resources competition at the co hosted application. They can also be generated when machine becomes slow i.e. CPU load has been increased which will lead to poor performance of machine. This can create Straggler tasks. The following table 1 shows the causes of straggler process/machine.

TABLE I. Causes of Straggler Machine

External factors	**Internal factors**
▪ Defective hardware ▪ Heterogeneous input ▪ Remote source being slow	▪ Resource competition due to other running tasks ▪ Increased CPU load

C. *Related work:*

There are various strategies developed till date to improve performance of MapReduce. In Google's MapReduce, the identification of straggler tasks starts when map and reduce stage are about to complete. It arbitrarily selects the tasks from the set of remaining tasks and mark that task as straggler tasks. Then that tasks will be backed up on ideal worker node. But this strategy doesn't consider that the task which they are selecting as straggler task is really a straggler task? A task can process more data which in real may not be slow but can be detected slow by default strategy in Google [13].

Hadoop-original improves this mechanism by keeping track of progress rate. When there is no map or reduce tasks to assign, it starts executing. Whenever tasks progress is less than average progress rate then it is considered to be as straggler tasks and then it is backed up on other ideal node. But it can be confusing in heterogeneous environment.[8] However, LATE improves the previous strategy which considers progress rate of task and calculate the total remaining time of tasks. A threshold is used to identify straggler task called as *slowTaskThreshold.* Whenever progress rate is less than the *slowTaskThreshold* is taken as straggler task. And then this straggler tasks are moved to execute on another worker node. Here *slowTaskThreshold* is calculated by standard deviation which can be misleading to identify straggler tasks [5].

Another strategy Maximum Cost Performance (MCP) proposed by Qi Chen considers both progress rate and process bandwidth to select slow tasks. It uses EWMA algorithm to predict process speed and calculate remaining time of tasks. It does not consider the average progress rate. It also focuses on maximizing cluster throughput. Whenever MCP identifies a slow task, it marks it to backup on ideal worker node. But it first checks whether backing up tasks will be beneficial or not. If it is beneficial then and only then it will be back up on ideal worker node [1].

Zhe Wang and Dong proposed workload aware strategy to improve the performance of MapReduce. They analyze the runtime history data from Job Tracker. It will send waiting jobs to characteristic estimation which will separates the job in two queues. Its aim is to divide the job dynamically into two type based on historical data named as CPU intensive and IO intensive [4].

The History-based auto-tuning (HAT) estimates the progress rate accurately and it incorporates historical information recorded on each node and detects straggler tasks dynamically. In previous strategy like LATE, only slow nodes is classified without considering their type. But in HAT slow nodes are classified into map slow nodes and reduce slow nodes. When HAT starts MapReduce job execution, each worker node reads the historical information from local worker node and set them as the default values of the parameters. The historical information contains the values of map and reduce tasks. On the basis of dynamic tuned values of map and tasks, HAT can compute progress score of the tasks accurately. It identifies the slow nodes according to average progress rate of map tasks and reduce tasks on each node. If there are any slow tasks are present, HAT launches backup tasks [3].

In Mantri [7], main focus is on saving cluster computing resources. To start execution of this strategy, it won't wait for the complete assignment of map and reduce task. It starts backing up outliers whenever they encountered. If backup tasks have high probability to finish earlier then Mantri will kill original task. To estimate task's remaining time, Mantri will use process bandwidth not progress rate. But this cannot assure that back up tasks will complete earlier than original one.

III. METHODOLOGY

A. *Introduction:*

As we know Hadoop is used to process large amount of data. Each node in Hadoop cluster has multiple operations to be executed. It may happen that due to lots of load, worker node may behave inappropriately. It may become straggler. This leads to slower performance of process on that node. This degrades the performance of overall cluster. When large amount of data has to be processed then it will become a point of concern.

As seen in section 2.3, there are various techniques which deal with straggler task/process problem. But all these method does not deal with machine's health. None of them consider wellness of the machine. It may happen that given process may not be straggler process, but machine on which it is going to execute may be straggler (slow). Then previously introduced techniques fails in such case. Proposed system design mainly focuses on performance of a machine. It calculates CPU usage of system which indicates the performance of the machine. It also considers the memory usage of a machine. By considering these factors proposed system will decide whether a machine is straggler or not. If machine is straggler then it will back up the task/process to the ideal worker node in the cluster. This technique will improve the MapReduce performance as it overcomes the problem of straggler machine. The aim of this work is to overcome straggler machine problem and improve Mapreduce Performance.

B. *Data Flow Diagram:*

Following figure 1 illustrates the data flow diagram of our system.

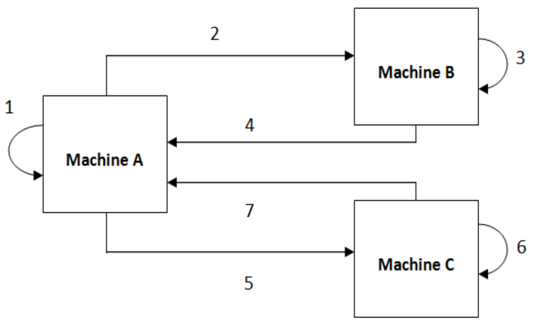

Figure 1: Data Flow Diagram

Here, Cluster of three machines has shown. It is describe as follows,

1. Machine A keep on checking its CPU usage. It has several MapReduce job to perform. Whenever its CPU usage is beyond the threshold value which is 65%, it will be considered as straggler machine. It indicates that machine will going to perform poorly. But if it is not straggler then its current MapReduce process will be executed there only.
2. When Machine A is found to be straggler, its current MapReduce process is migrated on Machine B. That process will be suspended on Machine A.
3. Machine B estimates its CPU usage. If it is below the threshold value it will starts executing Machine A's process. But if Machine B is also straggler then it will not execute Machine A's process.
4. If machine B found to be faster than Machine A then, Machibne A's MapReduce process will be executed on Machine B and it will provide the results back to Machine A.
5. For next process on Machine A, if it is found to be straggler machine. Then its current MapReduce process is migrated on Machine C. That process will be suspended on Machine A.
6. This step is same as Step 4 where Machine C estimates its CPU usage. If it is below the threshold value it will starts executing Machine A's process. But if Machine B is also straggler then it will not execute Machine A's process.
7. If Machine C found to be faster than Machine A then, Machine A's process will be executed on Machine C and it will provide the results back to Machine A.

C. *Algorithm:*

A node can have n number of tasks to perform. While executing each task system will calculate CPU_Usage of that node. If CPU_Usage is greater than α then we can say node is performing poorly and become straggler machine. Then that current process is backed up on another node to perform faster execution. Beside this, node will continue to perform next process. Meanwhile results of backed up process will be fetched from backup node. Thus in this way, it will give faster execution of MapReduce. The algorithm is as follows,

Algorithm 1:

```
1.  START
2.  for n number of processes {
3.          Calculate CPU_usage();
4.          if (CPU_usage > α )
5.                  Backup the current process on ideal worker
                    node in cluster;
6.                  getResults() from backup node;
7.          else
8.          Excute the current process on the same machine;
9.          getResults();
10. } // end for
11. END
```

Here, α = 65% based on experiments.[10]

The formulas mentioned below are used to calculate CPU usage of a machine.

$$CPU_{Usage} = \frac{totalUsedCPUTime}{totalAvailableCPUtime} \times 100$$

$$totalAvailableCPUtime = CPU_{count} \times (end - elapsedStartTime)$$

$$totalUsedCPUTime = currentCPUtime - CPU\_StartTime$$

Where,
CPU_Count = number of available processors

elapsedStartTime = It is the time taken from the startt of computer

CPU_StartTime = total CPU time of current thread in nanoseconds

Here, CPU usage is nothing but the CPU time measured in percentage. CPU time tells us about the amount of time CPU was busy in processing instruction from computer program or operating system [10]. This CPU time is use to assess the overall busyness of machine. If CPU usage is greater than α then machine tends to lag which leads to insufficient processing power. Thus processes on that machine also lags and degrade the performance [11]. In this way our system backed up MapReduce process on other node when its native machine tends to lag.

D. *Performance Evaluation:*

The results of proposed method are taken on two machine of which cluster has been formed. Both machines have Intel® Core™2 Duo CPU T6400 @ 2.00GHz × 2 processor with 2GB RAM and operating system of Ubuntu 14.04. Hadoop 1.2.0 and openJDK 7 is used in both the machines. We have executed word count, IP address extraction and Log File analyzer as benchmark program for performance evaluation. These programs have been evaluated on dataset of books and log file.

Following Figure shows bar chart for the performance evaluation. On X-axis we have benchmark programs, whereas Y-axis represents total execution time of process on machines in seconds. Machine A is considered to be straggler machine and machine B is non-straggler machine.

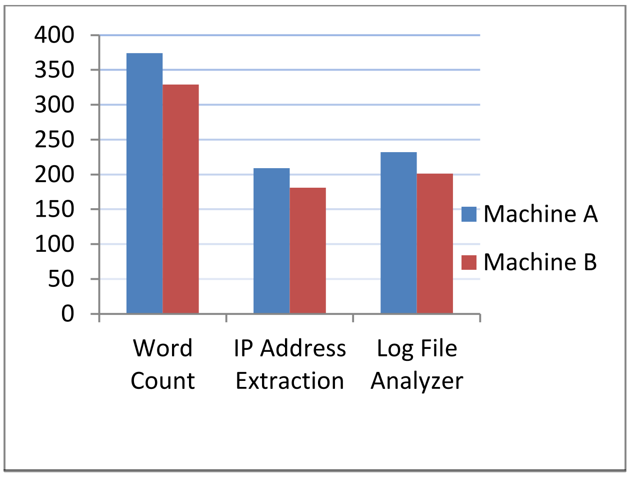

Figure 2 Performance Evaluation

This shows that process executing on non-straggler machine executes faster. Word Count, IP Address extraction and Log File Analyzer gives 12.03%, 13.39 % and 13.33% improvement respectively if they are backed up on non-straggler machine. So collectively taking average of improvement, we can say our design improve MapReduce performance by 12.91%.

IV. CONCLUSION

In this paper, we studied the basic mechanism of MapReduce and previous strategies to improve Mapreduce performance. Unlike previous strategies, our system identifies the straggler machine. We considered machine to be straggler when its CPU usage is greater than 65%. If a machine is found to be straggler then its current process/ task is backed up on other machine to execute faster. Our experimental results show that backing up task on other node will improve MapReduce performance by 12.91 %.

REFERENCES

[1] Qi Chen, Cheng Liu and Zhen Xiao, "Improving MapReduce performance using smart speculative execution strategy", IEEE Transaction on Computers VOL 63, NO. 4, APRIL 2014.
[2] IDC study. (2014) EMC. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf.
[3] Quan Chen, MinyiGuo, Qianni Deng, Long Zheng, Song Guo, Yao Shen, "HAT: History-based auto- tunningMapReduce in heterogenous environments" Springer Science+Business media, LLC, 2011.
[4] Zhe wang, Zhengdong Zhu, Pengfei Zheng, Quiang Liu, Xiaoshe Dong, "A new schedular strategy for heterogenous workload-aware in hadoop", 8th Annual China Conference, 2013.
[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, 2008.
[6] A. Thusoo et al. "Hive - a warehousing solution over a map-reduce framework", PVLDB, 2(2):16261629, 2009.
[7] G. Ananthanarayana, S. Kandula, A. Greenberg, I. Stocia, Y. Lu, B.Saha, and E. Harris, "Reining in the Outliers in Mapreduce Clusters Using Mantri" Proc. Inth USENIX Conf. Operating System Design and implementation, (OSDI '10), 2010.
[8] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," Comm. ACM, vol. 51, pp. 107-113, Jan. 2008
[9] MapReduce. (2013) [Online]. Available: www.ibm.com/software/data/infosphere/hadoop/map-reduce/.
[10] CPU Time. (2014) [Online]. Available: https://en.wikipedia.org/wiki/CPU_time.
[11] Understanding load Averages. (2009) [Online]. Available: http://blog.scoutapp.com/articles-/2009/07/31/understanding-load-averages.
[12] Apache hadoop. (2014) [Online]. Available: http://hadoop.apache.org.
[13] Huanle Xu, Wing Cheong Lau, "Optimization for Speculative Execution of Multiple Jobs in a MapReduce", Annual ChinaGrid Conference, 2013