IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-3, Issue-3 E-ISSN: 2347-2693

Comparative Study on Speculative Execution Strategy to Improve MapReduce Performance

Rahul R. Ghule[1*] and Sachin N. Deshmukh[2]

Department of Computer Science & Information Technology
Dr. BabasahebAmbedkarMarathwada University, Aurangabad, India

www.ijcseonline.org

Received: Mar/02/2015 Revised: Mar/08/2015 Accepted: Mar/22/2015 Published: Mar/31/2015

***Abstract*—**MapReduce is widely used and popular programming model for huge amount of data processing. Hadoop is open source implementation of MapReduce framework. 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. Some tasks can be executed slowly due to internal or external reasons. Because of this slow tasks job execution time is prolonged which leads to degradation of Hadoop performance. To overcome this, current MapReduce framework launch speculative execution in which each slow tasks is backed up other node in order to reduce the job execution time. These slow tasks can be called as straggler tasks. However, current MapReduce speculative execution does not estimate the progress of the tasks properly which leads to identifying incorrect slow tasks. Also, they do not consider data skew among the tasks. This paper studies various speculative execution strategy like HAT (History based auto-tuning), Longest Approximate Time to End (LATE) and Maximum Cost Performance (MCP). These strategies overcome the drawbacks of default speculative execution to improve MapReduce performance.

***Keywords*—**MapReduce, Hadoop, Straggler, speculative execution

I. INTRODUCTION

MapReduce[3] is proposed by Google in 2004 and in less time it become popular programming and parallel computing framework to process huge amount of data. Hadoop [2] is popularly used as open source implementation of MapReduce. In MapReduce user has to specify map function which takes key value pair as input and produces intermediate key value pair as output. User also has to specify reduce function which is used to merge all intermediate value which has same intermediate key.

When MapReduce system starts executing a job then master divides the input job file into multiple map tasks. Then master schedules both map and reduce tasks to the different nodes in a cluster to achieve parallel processing. By this all the nodes in cluster executes the tasks which is assigned to them. MapReduce job is completed when all the data is processed completely. The execution time of a job is measured by the tasks which finished last [3]. This can be a serious problem in homogenous and heterogeneous environments. Though it is not that much harmful in homogenous environment where node process same amount of data in similar time. But this can be serious issue in heterogeneous environment as execution time can be maximized due to last finished tasks which have been affected by uneven size of data, dissimilar data type, capacities of computation and so on [7]. This can degrade MapReduceperformance. When a machine takes usually longer time to complete the tasks then it is called as straggler tasks which will increase the execution time of MapReduce job and degrade the cluster throughput[1].This problem of straggler machine is popularly handled by speculative execution strategy. In this MapReducestarts a backup task for slow task on fast node. The aim of backing up slow tasks on fast node is to improve its execution time so that backup tasks can finish earlier than original one. This will leads to decrease job execution time.

In default speculative mechanism,MapReduce schedulers does not detect straggler correctly due to wrongly calculated remaining time of all the tasks. This causes two problems. First, the incorrectly launched backup task for a false straggler tasks cannot improve job execution time. Second, these incorrect backup tasks will waste the system resources and finally it will lead to degrade the overall performance of MapReduce job [5].

The rest of the paper is organized as follows: Section II describes causes of stragglers. Section III discusses various speculative execution strategies. Section IV describes comparative analysis of the strategies.

II. CAUSES OF STRAGGLER

There can be internal and external causes of straggler tasks. As we know data can be homogenous and heterogeneous in nature. The heterogeneous environment can create resource competition amongst the tasks inMapReduce job execution. This can create straggler tasks. Also, multiple MapReducejobs can be executed on the single node. This also leads to resource competition and creates straggler tasks. Some external causes are like data skew present in input, faulty hardware, remote input and so on.

Corresponding Author:*Rahul Ghule, rahulghule.r@gmail.com*
Department of Computer Science & Information Technology, Dr. Babasaheb Ambedkar Marathwada University, India

III. SPECULATIVE EXECUTION STRATEGY

There are various speculative execution strategy has been proposed to improve MapReduce performance. Their main focus is to shorten the job execution time. Some of the strategy is discussed below.

A. *Hadoop-Original*

The Hadoop-original is the default speculative execution in MapReduce. It is very simple. It begin speculative execution when map or reduce phase are about to complete [3]. Then it randomly selects some set of remaining tasks as backup tasks. But this has some pitfalls like whether randomly chosen tasks are really a straggler tasks. It does not consider that the backup node is fast or slow. Same speculative execution strategy is also used in Microsoft dryad. To overcome this default speculative execution strategy Hadoop made some changes. It launches speculative execution when all map and reduce tasks are assigned. It identifies straggler tasks on the basis of average progress rate. Whenever a task's progress is below the average progress rate then it classify that task as straggler task. But it can be misleading in heterogeneous environment as there is uneven data to process [2][5].

A. *Longest Approximate Time to End*

The Longest Approximate Time to End (LATE) algorithm is based on three major principle, they are- i) prioritize tasks to speculate, ii) select fast worker node on which slow task will be backed up iii) cap speculative to prevent thrashing. The LATE algorithm finds that default speculative execution in Hadoop can be misleading in heterogeneous environment and thus makes some improvement. LATE uses following parameters:

1. SlowNodeThreshold - This is the cap to avoid scheduling on slow nodes. The Scores for allsucceeded and in-progress tasks on the node are compared to this value.
2. SpecultiveCap - It is the cap on number of speculative tasks that can be running at once.SlowTaskThreshold- This is a progress rate threshold to determine if a task is slow enoughto be speculated upon. This prevents needless speculation when only fast tasks are running.Progress Rate of a task is given by (ProgressScore/ExecutionTime)
3. The time left parameter for a task is estimated based on the Progress Score provided byHadoop, as (1 – Progress_Score)/Progress_Rate.

LATE records the progress rate of a task and estimates their remaining time to identify slow tasks. The tasks having progress rate below the slowTaskThreshold cap are selected for backup process. Among those task which has longest remaining time is considered to be straggler tasks and given a high priority to back up on another worker node. A worker node can be classified as slow node when its performance score is below the slowNodeThreshold. LATE will never launch backup tasks on these nodes. In LATE,speculative cap is used to limit the number of backup tasks.

Algorithm 1: LATE

```
1.  a node N asks for a new task
2.  if number of running speculative tasks <SpeculativeCap then
3.  if nodes total progress <SlowNodeThreshold then
4.      ignore the request
5.  else
6.  rank currently running tasks that are not currently being speculated by estimated time left
7.  repeat
8.      select next task T from ranked list
9.      if progress rate of T <SlowTaskThreshold then
10.             Launch a copy of T on node N
11.             Exit
12.     end if
13. until while ranked list has tasks
14. end if
15. end if
```

The previous work shows that in a cluster with non-faulty nodes experiment LATE finished jobs 27% faster than Hadoop's native scheduler and 31% faster than no speculation. It also shows that SlowTaskThreshold (percentile of progress rate below which a task must lie to be considered for speculation) show that small threshold values harmfully limit the number of speculative tasks, values past 25%. It overcomes the problem of default speculative scheme but still it has some problems like input data skew. It also doesn't consider the fixed phase percentage of each phase in map and reduce. Such fixed percentage of phases will not be efficient to calculate progress rate [7].

B. *History Based Auto-tuning Strategy*

The History-based auto-tuning (HAT) calculates the progress rate accurately and it incorporates historical information recorded on each node anddetects straggler tasks dynamically. In previous strategy like LATE, only slow nodes is classified irrespective of 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 read the historical information from local 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. The runtime algorithm of HAT is as follows,

Algorithm 2: Runtime algorithm of HAT

Input: Key/Value pair
Output: Statistical value

1. Every worker nodes reads historical information and tunes parameters using the history-based auto-tuning strategy
2. Every worker node computes progress scores of all the running tasks using the progress monitoring algorithm
3. HAT processes the tasks and detects slow tasks using the straggler detecting algorithm
4. HAT detects slow nodes which can be map slow node or reduce slow node using the slow nodedetecting algorithm
5. HAT launches backup tasks on appropriate worker nodes using the backup task launching algorithm
6. HAT collects the results and historical information is updated on every node.

Algorithm 3: Straggler Tasks Detecting algorithm

1. While (the job is still running) {
2. Every worker calculates progress rate of every tasks that are running on it.
3. HAT calculates the average progress rate of all the running tasks.
4. Every worker determines slow tasks.
5. Every worker gives the list of slow tasks running on it.
6. HAT calculates the remaining time for all the slow tasks according to (5) and orders the tasks in descending order according to the remaining time.
7. HAT calculates the up-bound of the number of straggler tasks, Strag_UB.
8. If (the number of slow tasks ≤ Strag_UB)
9. All the slow tasks are detected as straggler tasks.
10. Else if (the number of slow tasks >Strag_UB)
11. HAT selects Strag_UB slow tasks with the longest remaining time as slow tasks.
12. HAT inserts all the straggler tasks into straggler map/reduce task list.
13. usleep(100000); } //straggler tasks are detected after every 100ms

HAT can launch backup tasks for reduce straggler tasks on map slow nodes and map straggler tasks on reduce slow nodes. The previous work shows that HAT can get up to 37% of performance gain over Hadoop and 16% performance gain over LATE scheduler. HAT cannot address data locality problem when launching backup tasks [16].

C. Maximum Cost Performance (MCP)

Qui Chen and Cheng Liu have proposed a new smart speculative execution strategy called as Maximum Cost Performance (MCP) to improve MapReduce performance. It mainly focuses on decreasing job execution time of MapReduce job and increase cluster throughput. This strategy aims to identify straggler tasks correctly and estimating total remaining time accurately. On the basis of the total remaining time backup tasks is selected and it is backed up on other node for faster execution. It also evaluates performance score of backup node so that backup task can be backed up on good node. Backing up a straggler tasks will eventually leads to maximum cost performance.
A task will be backed up on the basis of following condition

- It must have been executed for certain amount of time
- Both progress rate and process bandwidth must be low in the current phase
- Profit of backing up a straggler task is higher than not backing it up.
- Predicted remaining time of slow tasks on backup node is less than he estimated remaining time on original node.
- It has longest remaining time than all other tasks.

1) Selecting Backup task:

MCP can predict the process speed of all tasks and calculate their total remaining time on the basis of which detection of straggler tasks has been done.

a) Determine Process Speed:

Thisstrategyusesexponentially weighted moving average to predict the process speed of all running tasks. It is as follows

$$Z(t) = \propto * Y(t) + (1-\propto) * Z(t-1), 0<\propto<=1 \qquad (1)$$

Here, Z(t) = estimated process speed
Y(t) = observed process speed
t= time
$\propto$ is trade-off between stability and responsiveness [4].

To accurately estimate process speed, itwill not start calculating process speed as soon as execution starts. It allows the tasks execute for some time then it start estimating process speed.

b) Estimating Remaining Time and Identify Straggler Tasks:

Previous strategies used to identify straggler tasks on the basis of average progress rate alone or some uses process bandwidth alone. But alone it is not sufficient to identify straggler tasks. For example, if tasks havelarge data to process it tend to have low progress rate although its bandwidth is normal. This leads to misjudgement. Hence MCP uses both progress rate and process bandwidth to identify straggler tasks.

As we know map and reduce phases are subdividedi.e.

map is divided in map and combine whereas reduce is divided in copy, sort and reduce. It use EWMA algorithm in each of thesesub phases to predict process speed in each of these phases. This can be useful in accurately identifyingstraggler tasks. Meanwhile, estimating remaining time of tasks in each phase is done.

A tasks remaining time is calculated by sum of remaining time in each phase. When a task is running ina phase say *cp*(i.e. current phase), then its remaining time in *cp* is calculated by remaining data left to process and process bandwidth in *cp*. It isneeded to calculate remaining time of the same tasks in other phases say *fp*(i.e. following phase). Hence to estimate remaining time in *fp* it uses the average of process speed in each phase. It takes average of the process speed of all the tasks which have entered in *fp*. It does not calculate average process speed of the phase in which no tasks has entered yet [1].

$$rem_{time} = rem_{time_{cp}} + rem\_time_{fp} \quad (2)$$

$$rem_{time} = \frac{rem\_time_{cp}}{bandwidth_{cp}} + \sum_{p\ in\ fp} est_{time_p} * factor_d \quad (3)$$

$$factor = \frac{data_{input}}{data_{avg}}$$

To estimate backup time of a straggler tasks, ituses the sum of the estimated time for each phase of the tasks given by est_time_p. It is as follows,

$$backup\_time = \sum_p est\_time_p * factor_d \quad (4)$$

This estimated backup time is compared with the calculated remaining time of straggler task.

c) Maximizing Cost Performance of a cluster

The cost of speculative execution in tasks can be represented by occupied slots. The aim is to maximise the cost performance of cluster computing resources and shortening the job execution time. The cost can be taken as the time for which computing resources are occupied i.e. *slot_num * time* and benefit is denoted by time saved by the speculative execution in the task. Backing up a task takes two slots (one for original and one for backup tasks) and can save one slot *rem_time – backup_time* whereas not backing up a task will take only one slot *rem_time* and gain nothing [1]. This is defined as follows,

$$profit_{backup} = \propto * (rem_{time} - backup_{time}) - \beta * 2 * backup\_time \quad (5)$$

$$profit_{notbackup} = \propto * 0 - \beta * rem\_time \quad (6)$$

If profit of backing up straggler tasks is more then it will back it up on other node else original task is executed.

2) Selecting proper backup node

To gain better performance, straggler task must be backed up on fast backup node. If straggler task is backed upon slow backup node then it won't get any gain by backing it up. To classify, MCPcan maintained health of each node. Ithas assigned a performance score to each node and on the basis of this score node status of its health is obtained, whether it is fast or slow. For that ituses some appropriate parameter to measure the performance score of the nodes.

It keeps the track of the process bandwidth of the tasks which are executed on each node. Hencethe performance score of the nodes can be calculated. It considers data locality of the tasks. It has taken the locality of data in account whether it is data local or non-local because data local task are three times faster than non-local tasks. Before assigning a node to straggler tasks it must check its locality so that it can perform faster. If slow tasks executes faster on backup node then only it is backed up on worker nodes.

The previous workshows that MCP finishes job 37% faster than the Hadoop-original and 19% faster than the LATE. MCP improves cluster throughput by 32% over Hadoop-original and 15% over LATE [1].

IV. COMPARATIVE ANALYSIS

The challenge is to improve MapReduce performance by shortening job execution time and maximizing cluster throughput. Various Speculative strategies are developed till now. The default speculative execution strategy randomly selects the task and backs it up on other worker node. But it has some drawbacks in identifying slow tasks and choosing worker node for backup tasks. It assumes that task makes progress at stable rate but this assumption breaks down for various reasons. First, map and reduce is subdivided in different sub phases. These sub phases are assigned fixed progress of a task which makes monitoring process difficult. Second, reduce phase launched asynchronously before all map tasks complete. Thiscauses variation in progress rate of tasks. To overcome such drawbacks LATE has been developed. It keeps the track of progress rate of each task and which is used to correctly identify straggler tasks. The task which has progress rate below threshold value will be considered for backup task. But it does not consider whether backup task finish earlier on worker node. HAT strategy is developed which can maintain history of each task. All the historical information is used to detect straggler tasks accurately. It classifies the slow nodes into map slow nodes or reduce slow node which is not present in earlier strategy. This helps in backing up tasks properly. MCP addresses the issue of input data skew, fixed percentage of each phases of map and reduce efficiently. It maintains performance score of worker node for backup task. It uses EWMA prediction algorithm to estimate progress speed of a tasks. While

backingup a slow task,it considers data locality. MCP and HAT improves LATE strategy efficiently.

V. CONCLUSION

To improve MapReduce performance is currently a big task to do. Various techniques like scheduling of tasks and speculative execution strategy are under research. This paper discussed different strategies of speculative execution to improve MapReduce performance. LATE has made considerable changes in Hadoop-original. MCP is better than LATE as itovercomes pitfalls of LATE strategy. HAT uses historical information of a task which shows considerable improvement over LATE. The main aim of all these strategies is to shorten the job execution time and increase the cluster throughput. This will be helpful in Big Data applications.

ACKNOWLEDGMENT

The author would like to thanks the university authorities and Department of Computer Science and Information Technology, Dr. Babasaheb Ambedkar Marathwada University, Aurangabad for providing the infrastructure to carry out the work. This work is committed by university commission.

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] Apache hadoop, http://hadoop.apache.org/, Accessed on 26 December 2015

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," Comm. ACM, vol. 51, pp. 107-113, Jan. **2008**.

[4] Exponential Weighted Moving Average, http://en.wikipedia.org/wiki1, Accessed on 7 January 2015

[5] MapReduce. [Online] Available: http://www.ibm.com, Accessed on 15January 2015

[6] 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**.

[7] 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**.

[8] Zhe Wang, Zhengdong Zhu, Pengfei Zheng, Qiang Liu, Xiaoshe Dong, "New Scheduler Strategy for Heterogeneous Workload-aware in Hadoop," 8th Annual ChinaGrid Conference, **2013**.

[9] Huanle Xu, Wing Cheong Lau, "Optimization for Speculative Execution of Multiple Jobs in a MapReduce-like Cluster," 8th Annual ChinaGrid Conference, **2013**.

[10] Xuelian Lin, Chunming Hu, Richong Zhang, Chengzhang Wang, "Modeling the Performance of MapReduce under Resource Contentions and Task Failures," Cloud Computing Technology and Science (CloudCom), IEEE 5th International Conference on (Vol 1), December **2013.**

[11] Tao Gu, Chuang Zuo, Qun Liao, Yulu Yangand Tao Li, "Improving MapReduce Performance by Data Prefetching in Heterogeneous or Shared Environments", International Journal of Grid and Distributed ComputingVol.6, No.5, **2013**.

[12] 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," Proc. Ninth USENIX Conf. Operating Systems Design and Implementation (OSDI), **2010**.

[13] Y. Kwon, M. Balazinska, and B. Howe, "A Study of Skew in Mapreduce Applications," Proc. Fifth Open Cirrus Summit, **2011**.

[14] Open Stack Cloud Operating System, http://www.openstack.org/, Accessed on 13 February 2015.

[15] Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2/,Accessed on 28 January 2015

[16] 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**.

AUTHOR PROFILE

Rahul R Ghule doing M. Tech (Computer Science & Engineering) from Department of Computer Science and Information Technology, Dr. Babasaheb Ambedkar Marathwada University, Aurangabad-431001, Maharashtra, India.
Rahulghule.r@gmail.com

Assistant Professor Dr. Sachin N. Deshmukh Department of Computer Science and Information Technology, Dr. Babasaheb Ambedkar Marathwada University, Aurangabad-431001, Maharashtra, India.
sndeshmukh@hotmail.com