Optimization For Speculative Execution In Heterogeneous MapReduce Environment

HONG Qingqing^{a,*}, Fan QIAN^b, Lifeng ZHANG^a

^aKyushu Institute of Technology University, Fukuoka Sekisui-ku, Kita Kyushu-shi, Sendai-machi 1-1, Kitakyushu, 804-8550, JAPAN

^bCollege of Information Engineering, Yangzhou University, Yangzhou, China

*hongqingqing1221@gmail.com

Abstract

MapReduce proposed by Google is an open-source distributed computing framework for parallel processing of large-scale data. Speculative execution is a problem in the MapReduce. Because a job submitted by client cannot finish until all map and reduce tasks finished. Sometimes, because of the heterogeneous environment (the computing capabilities of each task-tracker is different), some task-trackers process tasks slower than others, we called them stragglers. In case of the stragglers, the tasks assigned will incur lower performance as well as entire execution time delay. The purpose of our research is to optimize the speculative execution through improving LATE⁽¹⁾ scheduler, we will solve the issue still exist in LATE, that the average progress score is unstable. To tackle this problem, we proposed that take process bandwidth along with the average progress rate of each phase into consideration.

We use two applications to simulate our proposal. One is word count and another is sort. The result showed that our proposal could decrease the time of speculative execution from 3% to 15%.

Keywords: MapReduce, Speculative execution, Heterogeneous.

1. Introduction

In Today's world, the internet service with millions users has become the most popular computer applications. Huge amounts of data make us apply the idea of parallel computing to the commercial cluster. Hadoop is one of more mature platforms in recent years. Its technology has been widely used in the Internet field, and it has been widely concerned in the field of research. Hadoop consists

of two parts: Hadoop Distributed File system (HDFS) and MapReduce⁽²⁾. HDFS is used to manage cluster's data, MapReduce is used to process parallel mission. In this case that, the parallel computing of large-scale data makes the capability of task scheduling more important in the MapReduce.

Hadoop solves task scheduling problems on the basis of a number of assumptions. Hadoop was originally designed for homogeneous environments means assume that all nodes have the same performance, and the task execution is performed at a constant speed. Meanwhile, the built-in backup task execution strategy does not consume any resources. But in real life, it's widely used in various heterogeneous environment now. Because of the heterogeneous nature of both machines and workloads, Hadoop performs poorly, and some severe performance degradation and resource inefficiency of the Hadoop scheduler is observed⁽¹⁾⁽³⁻⁵⁾.

In this work, we address the problem that how to improve task scheduling of Hadoop in heterogeneous environments. The basic principle is fundamental for a high performance and resource efficient heterogeneous Hadoop cluster. Our approach is to focus on LATE scheduling algorithm⁽¹⁾. Because LATE algorithm takes heterogeneous environments into consideration. However, LATE has poor performance due to its static fixed-weight based method.

To overcome the deficiency of LATE, we have developed LATE: take process bandwidth along with the average progress rate of each phase into considered.

The rest of the paper is organized as follows: chapter 2 background of the basic principle of MapReduce and our exist issue still remained in the previous research. Chapter 3 describes our research's proposal. Chapter 4 describes our evaluation and result.

2. Background And Related Work

2.1 MapReduce

MapReduce is an open-source distributed computing framework for parallel processing of big data proposed by Google⁽⁶⁾. MapReduce's framework consists of a master node called Job-tracker, slave node called Task-tracker and task scheduler (as Fig.1.a shown). The Job-tracker is responsible for scheduling all the tasks of a job, these tasks distributed in different nodes. The Job-tracker will monitor the node's situation of processing through heartbeat and re-run the failed tasks. The Task-trackers' duty is to process the tasks assigned by Job-tracker. When a job submitted by client, after Job-tracker received the submitted tasks and configuration information (the information includes status of task, task ID, start time of the task, finish time of the task and progress rate of the task), it will assign this information to Task-trackers. Meanwhile, Job-tracker will schedule the tasks based on scheduling algorithm and monitor the situation of data processing.

Users specify a Map function that processes a key/value pair to generate a set of intermediate key/value pairs and a Reduce function that merge all intermediate values associated with the same intermediate key. Finally, the last stage merge together all the key/value pairs (as Fig. 1.b shown⁽⁶⁾).

(a)MapReduce framework

(b) Execution process of MapReduce

Fig. 1. MapReduce

Ideally, in homogeneous environment each task should take approximately the same time to complete if it is assigned the same amount of data. But in real life, this

assumption will be broken down. In heterogeneous environment, there is no guarantee that data is evenly distributed among tasks or the same amount of input data will take the same time to process due to the computing capabilities of nodes may vary significantly.

For tackling this problem, MapReduce gives an mechanism called speculative execution.

2.2 Stragglers

A MapReduce processes ends when its all map and Reduce tasks are finished. Sometimes, a slow processing Task-tracker, called Straggler, is found. Meanwhile, the task running on the straggler will be called straggler task. This incurs lower performance as well as entire processing delay. There can be internal and external causes of straggler tasks. As we know data can be homogeneous and heterogeneous in nature. The heterogeneous environment can create resource competition among the tasks in MapReduce job execution. In this case that, straggler tasks will be occurred. Also, multiple MapReduce jobs 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⁽⁷⁾.

Fig. 2. Find a straggler

As Fig. 2. shown, for example, there two tasks running on two Task-trackers. After these tasks running 1 min, Task-tracker 1's progress score is 2/3, Task-tracker 2's progress score is 1/12. In this case that, we can identify that Task-tracker 2 is the straggler. And the task 2 will be identified as straggler tasks. So that, the straggler tasks will prolong the whole execution time and cost a lot of resources.

2.3 Speculative Execution

In order to avoid the stragglers, MapReduce employs the speculative execution mechanism to solve the execution time and computing resources. When MapReduce find that one task is running more significantly than other tasks, it will launch this slower task's copy with same data chunk on the fast node (here fast node means not the straggler). Commonly, each task has three copies on different

Task-trackers with same data chunk. As soon as one of these copies complete first, other copies will be discarded.

To select straggler tasks for speculative execution, MapReduce default scheduler monitor the progress of tasks using a Progress Score (PS) between 0 and 1. And PS_{avg} denotes the average progress score of one job. $PS[i]$ means i^{th} 's progress score. We will make an example to introduce how to launch speculative execution via MapReduce default mechanism. Suppose: one job has K number of tasks to be executed; there are totally N numbers of key/value pairs of one task to be processed and among these there are M of them being processed successfully. MapReduce gets PS according to Eq.1 and Eq.2 shown as follows and then launches the straggler tasks via Eq.3.

$$PS = \begin{cases} M/N, & \text{For Map Task} \\ \frac{1}{3} \times (K + \frac{M}{M}), & \text{For Reduce Task} \end{cases} \quad (1)$$

$$PS_{avg} = \sum_{i=1}^K \frac{PS[i]}{K} \quad (2)$$

For task T_i : $PS[i] < PS_{avg} - 20\%$ (3)

Here, it is assumed that a map task spends negligible time in the order stage (i.e., $M1=1$ and $M2=0$) and a reduce task has finished K stages and each stage takes the same amount of time (i.e., $R1=R2=R3=1/3$). If Eq.3. is satisfied, task T_i needs a backup task^[8].

But all the approaches introduced above are based on homogeneous environment, the default equation was assumed that all the nodes' computing capabilities are the same, the cost of launching a backup task can be ignored. In contrast, these assumptions will be broken down in heterogeneous environment. Scheduler launches backup tasks based on PS, since in heterogeneous environment a lower PS not necessarily means a longer remaining execution time.

2.4 Related Work

In order to tackle the deficiency mentioned above, In (1), researchers have proposed Longest Approximate To End (LATE) algorithm to overcome the pitfalls. LATE algorithm is designed to calculate estimated completion time of tasks and first speculate task which end farthest in the future (even if its progress score is faster than others). Because tasks ending last will hurt the response time most.

First, LATE sets several parameters described as follows:

(a) SlowNodeThreshold:

This is the cap to avoid scheduling on slow nodes. ((1) set SlowNodeThreshold as 25%)

(b) SlowTaskThreshold:

This is a progress rate threshold to determine if a task is slow enough to be speculated. ((1) set SlowTaskThreshold as 25%)

(c) SpeculativeCap:

It is the cap on number of speculative tasks that can be running at once. ((1) set SpeculativeCap as 10% of the number of tasks of a job).

Based on these three parameters, the LATE algorithm works as follows⁽¹⁾:

- If a node asks for a new task and there are fewer than SpeculativeCap speculative tasks running:
 - 1) Ignored the request if the node's total progress is below SlowNodeThreshold.
 - 2) Rank currently running tasks that are not currently being speculated by estimated time left.
 - 3) Launch a copy of the highest-ranked task with progress rate below SlowTaskThreshold.

Besides (1) sets these three parameters, the main idea of LATE is to estimate the remaining time of tasks via using progress score provided by Hadoop. The equations are described as below:

$$\text{Progressrate} = \frac{\text{ProgressScore}}{t} \quad (4)$$

$$\text{Estimated Time Left} = \frac{1 - \text{ProgressScore}}{\text{Progressrate}} \quad (5)$$

Where t in Eq.1. represents for the amount of time the task has been running for. These equations are based on the assumptions that tasks make progress at a roughly constant rate.

We illustrate with an example to describe the Eq.4. and Eq.5. vividly. As figure 3 shown, there are four tasks running on three Task-trackers. The performance of these three Task-trackers are different as shown in figure 3. After these four tasks running 2 minutes on these Task-trackers, the progress score of Task-tracker 2 is 66%, and Task-tracker 3's progress score is 5.3%. After two minutes, task 2 and task 4 still haven't finished. So based on the Eq.4. and Eq.5., we can estimate the time left of task 2 is one minute, the task 4s' is 1.8 minute. In this case that, we can identify that task 3 is the straggler task due to its estimated time is the largest. And then the backup task of 3 will be launched in Task-track 1. And the improvement that after speculative execution, the time that LATE has solved. The same as Hadoop default scheduler, LATE also need to wait until a task has run for 1 minute before evaluating it for

speculation.

Fig.3. How to select the straggler task

The LATE has several advantages. First, it is robust to node heterogeneity, because it will relaunch only the slowest tasks, and only a small number of tasks. LATE also caps the number of speculative tasks to limit contention for shared resources. In contrast, Hadoop's default scheduler has a fixed threshold, beyond which all tasks that are slow enough have an equal chance of being launched. Second, LATE considered node heterogeneity when deciding where to run speculative tasks. In contrast, Hadoop's default scheduler assumes that any node that finishes a task and asks for a new one for is likely to be a fast node. Finally, by focusing on estimated time left rather than progress rate, LATE speculatively executes only tasks that will improve job response time, rather than any slow tasks⁽¹⁾.

3. Pitfalls In The Previous Work

As we mentioned above, speculative execution still has many issues for different reasons. Many researchers have proposed different approaches to solve it. Mainly they focused on reducing the stragglers, choosing appropriate nodes for speculative tasks and data locality for straggler tasks. In section 2.4, we have introduced LATE algorithm proposed by (1). Although LATE has overcome the deficiencies of default approach of speculative execution but still has problem exist in calculating estimated remaining time. The progress score of a Task-tracker does not increase continuously from beginning to end. Progress score sometimes shows fast increase, but may show slow increase in other occasions, since most of the MapReduce clusters are shared by users have launched some applications. Estimated time left calculated in the former case is shorter than the real value of time to complete and is longer in the latter case.

For example, suppose a task is running at the speed of a percent per second for b seconds when suddenly the speed falls to a/5 percent per second due to resource competition. The average process speed will fall down to a/2 percent per second only after 5/3 b seconds. As a result, the remaining

time estimated according to the progress score will be much shorter the actual value.

Fig.4. Process speed fall down dramatically

In this case that, the LATE proposed by (1) will take a long time to identify a straggler task by using the progress score or even fail to identify straggler tasks.

4. Design And Implementation

Our proposal is proposed to tackle the problem exist in LATE we mentioned in the previous chapter. Many researches focuses on speculative execution mainly two areas:

1. Reducing the stragglers;
2. Choosing appropriate task-trackers for straggler tasks and data locality for straggler tasks;

As far as we concerned, the key method is to optimize the speculative execution through increasing the precision of identifying stragglers and reducing the cost of launching straggler tasks.

4.1 Strategy Description

Our approach is to select straggler tasks accurately and promptly and launch the straggler tasks in the idle processors. To ensure fairness, we assign task slots in the order the job submitted. Our proposal gives the new task a higher priority than straggler tasks just like other speculative strategies. It means that our proposal will not start straggler tasks until all new tasks have been assigned. Our proposal chooses the straggler tasks from the candidates based on the tasks' estimation time left which calculated by taking the process bandwidth along with the average progress rate of each phase into considered not only considered the progress score like LATE scheduler. Then, we also calculate the backup time of straggler tasks and compared with the time that not backup. Because when the cluster does speculative execution, the straggler tasks and original tasks will occupy two slots, in this case that it will cost many resources.

4.2 Take the process bandwidth with the average progress rate of each phase into considered

LATE scheduler we have mentioned identifies a task as straggler task when its progress rate is lower than the average progress rate by a fixed threshold. But only use the progress rate alone is in sufficient to identify straggler tasks. For example, large tasks which have more data to process may have a lower progress rate even though their processing bandwidth is normal. However, only use the process bandwidth is also not sufficient. Because of the impact of the constant task start up time, small tasks which have less data to process will be misjudged by the process bandwidth alone. So in our approach, we take the process bandwidth alone with the progress rate into considered, in this case that we can avoid the misjudgment and increase the precision.

Moreover, from the Fig.5. and Fig.6. shown as below we can see that a task 's progress rate may vary greatly across different phase.

Fig.5. Map task process

Fig.6. Reduce task process

These two figures show the time cost in the Map and combine phases is significantly different from the percentage of that configured in Hadoop for typical Word count, Grep, Sort. This gap will make mutation of tasks' progress rate happen.

The Fig.7. shows the observed curve that some researchers used 30 machines to do the sort application and counted the average rate of reduce tasks at each time point. We can find the in the copy phase of Reduce task, the average progress rate is almost steady. But when the sort phase started, because of the data in reduce process is local

data, the sort phase will complete fast. So the progress rate of sort phase increase significantly, the entire reduce tasks' progress rate will be increased. In the reduce stage, because the output of reduce tasks will be written in HDFS which is a distributed file system in Hadoop, large data should be transferred through network. In case that, it will limit the progress rate of reduce stage. However, the average of each phase s steady. So we will use this per-phase process speed to identify straggler tasks.

Because when a task is running in some phase current phase (cp), the remaining time left in cp is estimated by the factor of the remain data and the process bandwidth in cp. But the remaining time of the following phases indicated by fp is difficult to calculate since the task has not entered those phases yet. Therefore, we use the phase average score to estimate the remaining time of a phase indicated by est_time_p . The average progress score is the average progress score of tasks that have entered the phase. For those phases that no task has entered, we do not calculate their remaining time which is fair to all tasks. We will use $factor_d$ to adjust est_time_p because tasks may process different large data. The $factor_d$ represented by the ratio of input size of this task to the average input size of all tasks. So the formula of estimating the remaining time of tasks as follows:

$$rem_time = rem_time_{cp} + rem_time_{fp} \quad (6)$$

$$rem_time = \frac{rem_data_{cp}}{bandwidth_{cp}} + \sum_{f \in fp} est_time_p \times factor_d \quad (7)$$

$$factor_d = \frac{data_{input}}{data_{avg}} \quad (8)$$

4.3 Calculate the backup time of speculative tasks and compare with the time that does not backup

Speculative execution has not only benefits, but also it may cost a lot. In a Hadoop cluster, task slot is the cost of speculative execution. Because the benefit is that shortening of the job's execution time. We use a cost_benefit model to analyze the trade-off. In this model, the cost is represented as the time that the computing resources are occupied indicated by $slot_number \times time$, while the benefit is represented as the time saved by speculative execution.

To estimate the backup time of a speculative task, we use the sum of est_time_p for each phase in this task as an estimation. So the formula for calculating the backup time is as follows:

$$Backup_time = \sum_p est_time_p \times factor_d \quad (9)$$

Two slots will be occupied for backing up a task because the original and the backup need to keep running until either completes. So in this case that, save one slot is indicated by $rem_time - backup_time$. But in contrast, not backing up speculative tasks will cost just one slot rem_time and benefit nothing. The benefit of these two cases (backing up the straggler tasks or not) is the slot time that can be saved. So the formula to calculate the benefit of these two cases are shown as follows:

$$benefit_{backup} = \alpha \times (rem_time - backup_time) - \beta \times 2 \times backup_time \quad (10)$$

$$benefit_{not_backup} = \alpha \times 0 - \beta \times rem_time \quad (11)$$

In these two formulas, rem_time and $back_time$ we have mentioned in the previous, α and β are the weight of benefit of and cost respectively. Then we will compare with the benefit of backing up the straggler tasks or not. If the benefit of backing up this task out weights that of not backing it up, we will consider this task slow enough and select it as a backup candidate.

After iterating through all the running tasks, we will get a set of backup candidate. The candidate that has the longest remaining time will be backed up finally.

5. Evaluation and result analysis

5.1 Evaluation Environment

In this chapter, we will evaluate the performance of our proposal under heterogeneous MapReduce environment. And compare with LATE we mentioned in the related work.

Firstly, we set up a heterogeneous environment: making MapReduce framework runs on nodes with diverse hardware, such as CPU and the memory shown as table 1. The MapReduce framework we built is composed of five nodes shown as Fig.12., where, master node named Job-tracker, which is equipped with 8GB of memory and Intel core i7 6700HQ 2.6GHz two core processors. Other nodes are worker nodes named Task-tracker 1,2,3,4 are responsible for executing Map and Reduce tasks. The parameter of each Task-tracker shown as table 1. In order to make the computing capability variances between each nodes can be bigger, we execute background program of a read-write file to increase CPU load intentionally in Task-tracker 2.

Table 1. The configuration of MapReduce cluster

Node	CPU	Memory
<u>Jobtracker</u>	Intel corei7 6700HQ 2.6GHz	8GB
<u>Tasktracker 1</u>	Intel corei5 6300HQ 2.3GHz	4GB
<u>Tasktracker 2</u>	Intel coreM5 6Y54 1.1GHz	2GB
<u>Tasktracker 3</u>	Intel corei7 4720HQ 2.6GHz	8GB
<u>Tasktracker 4</u>	Intel corei3 4030U 1.9GHz	4GB

Fig.7. The architecture of our model

5.2 Evaluation Parameter Setting

All nodes in my experiment run on Ubuntu 10.10 Sever 64bit, JDK 1.6 and Hadoop 0.20, programming language is Java. We assume that every worker node has four Map slots and four Reduce slots, in other words, there are four Map tasks and Reduce tasks in one worker node. And other parameter in MapReduce is default.

We will use Wordcount which is a program to divide the sentences and to calculate the number word appearance as type of Jobs compare to the sort which is mainly for sorting files, merging sorted files, and checking files to see if they are sorted. And the input data is created by RandomTextWriter according to Zipf's law. And In order to improve the accuracy of evaluation results, we will execute the WordCount and sort repeatedly with different input data size, and each size we execute five times and obtain the average value.

Because we need compare the result of our proposal with LATE, we need set SpeculativeCap, SlowNodeThreshold and SlowTaskThreshold adapted by [1] as follows:

$$mapreduce.job.speculative.speculativecap = 0.1$$

$$mapreduce.job.speculative.slownodethreshold = 0.25$$

$$mapreduce.job.speculative.slowtaskthreshold = 0.25$$

Each Tasktracker only has four map tasks:

$$mapred.tasktracker.map.tasks.maximum = 4$$

We will use Wordcount which is a program to divide the sentences and to calculate the number word appearance as type of Jobs compare to the sort which is mainly for sorting files, merging sorted files, and checking files to see if they are sorted.

5.3 Measure Metrics

We compare our proposal with LATE to show the improvement by using the job execution time and the precision of identifying the stragglers as measure metrics. These two metrics are introduced as follows: (1). Job execution time: The average elapsed time per job, to evaluate the overall performance of MapReduce; (2). The precision of identifying the stragglers: We will locate the stragglers and calculate the precision. Meanwhile, we need compare the time of identifying the stragglers between these two approaches.

5.4 Evaluation Result & Results Analysis

Fig.8. and Fig.9. are our final evaluation results. In these figures, X axis is the input data size and Y axis is the execution time of tasks in each node. Firstly, we evaluate the execution time of tasks in each node. As these two figures showed, we used six groups of input data with different sizes that is 1 GB,2 GB,3 GB,4 GB,5 GB,6 GB. The figure 13 is the result of using word count as our experiment's application. And the figure 14 is the result of using sort as our experiment's application. Both of these two results, we can see that with the input data size increased, the execution time of our proposal and LATE scheduler are increased. And the we can the input data size larger, our proposal's improvement shows more significant.

Fig.8. The execution time of using word count

Fig.9. The execution time of using sort

Both these two results, our proposal can decrease the job execution time from 3%~ 15%. From the figure 15, we can see the improvement of sort is more significant than word count. Word count does not gain significant improvement, cause the nature of word count, intermediate data and the final output of word count are very small. Word count executes in the map stage, is cpu intensive. But Sort writes a large amount intermediate data and final output through the network and to the disk The reduce tasks in sort jobs do much more work than its in map stage, the map stage makes up a very small part of the total execution time.

Fig.10. Comparison of improvement ratio between sort and word count

The result of second measure metrics is shown as table 2. We monitor the situation of each task and count the failed tasks through the Hadoop's Web management. From the web, we can check the number of tasks (Map tasks and Reduce tasks), the progress of tasks and failed tasks. From the table 3 counted by the web management, we can see that when we use word count as example, the precision of identifying stragglers by our proposal is 39.25%, and by LATE scheduler is 20.3%. So this table shows our proposal identifies the stragglers takes more accurately than LATE scheduler. From the result, our proposal can improve the precision in identifying stragglers by 19% compared to the result of LATE scheduler.

Table 2. The precision of identifying the stragglers

Proposal	Precision of identifying stragglers
Our proposal	39.25%
LATE scheduler	20.3%

5.5 Drawbacks of our proposal

From our results of evaluation, we can see that our proposal has a better performance than Zaharia's LATE scheduler. But there is still a problem exist in our proposal. We didn't take data locality into considered. In this case that, when launch the speculative tasks with no local data, the cost of bandwidth and I/O disk, CPU will be increased.

6. Conclusion & Future Work

In this paper, to tackle the issue exist in (1), we presented a new approach for speculative execution to solve the problem, which takes the process bandwidth along with the average progress rate of each phase into consideration. Meanwhile, we will calculate the backup time of speculative tasks and compare with the time that non-backup.

In the experiment, we firstly set up a heterogeneous environment by different computer with various hardware and then compared the results of same job executed in MapReduce with our proposal and Zaharia's one. The evaluation results show that our proposal additionally considered the different computing capability of each node, abd comparing with Zahria's one, our proposal can shorten entire job execution time in case of MapReduce framework running on heterogeneous environment about 3%~15%. And through our two results of simulating two different kinds of applications, we can see that our proposal have a better performance during processing I/O intensive application.

Evaluation results are verified that the theoretical analysis of our proposal could tackle the issue. We didn't take data locality into consideration. In this case that, when to launch the speculative tasks with no local data, the cost of bandwidth and I/O disk, CPU will be increased.

References

(1) M.Zaharia, A.Konwinski, A. d.Joseph, R. Katz and

I.Stoica, : "Improving MapReduce Performance in Heterogeneous Environment", in Proc. of 8th USENIX Symposium on Operating System Design Implementation, 2008.

- (2) Vaquero L M, Rodero-Merino L, Caceres J, et al. A break in the cloud: Towards a Cloud Definition[J]. ACM SIGCOMM Computer Communication Review, 2009, 39 (1):50-55.
- (3) G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective Straggler Mitigation: Attack of the Clones," in the 10th USENIX Symposium on Networked Systems Design and Implementation SDI '13),2013.
- (4) Q. Zhang, M. F. Zhani, R. Boutaba, and . L. Hellerstein, "HARMONY: Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," in Inte ational Conference on Distributed Computing Systems (ICDCS), 2013.
- (5) F. Ahmad, S. T. Chakradhar, A Raghunathan, and T. N. Vijaykwnar, "Tarazu: optimizing MapReduce on heterogeneous clusters," inASPLOS , New York, NY, USA, 2012,pp. 61--74.
- (6) J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107– 113, Jan. 2008
- (7) Rahul R. Ghule and Sachin N. Deshmukh, "Comparative Study on Speculative Execution Strategy to Improve MapReduce Performance," *International Journal of Computer Sciences and Engineering*, vol. 3, Issue 3, pp. 196-200, 2015.
- (8) Xiaoyu Sun, Chen HE and Ying Lu, "ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm," 2012 IEEE 18th International Conference on Parallel and Distributed System, pp. 148-255, 2012.