Volume 4, Issue 3, March 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Enhanced Speculative Execution Mechanism to Improve Mapreduce Performance with Effective Resource Utilization

T. Mallem Aswanth*, Prof. C . Malathy
Computer Science and Engineering,
SRM University, India

Abstract*— MapReduce is an effective programming model for large scale data processing. Hadoop is an open source implementation of MapReduce framework. Hadoop's performance depends on the rate at which task is completed which implicitly assumes that all the nodes are homogeneous in which tasks make progress linearly, and uses these assumptions to decide when to speculatively re-execute tasks that appear to be progressing slowly. In practice, these homogeneity assumptions do not always hold. The MapReduce performance is impacted by the nodes on which tasks take long time to finish. Speculative execution is an approach for dealing with such problems by backing up those slow tasks on alternative nodes. In this paper we propose an enhanced speculative execution mechanism that improves the performance of the MapReduce during execution of the task. Using this mechanism we calculate the task completion time on nodes that were running currently and determine which task to be backed up based on the load of the node using enhanced speculative execution mechanism.

Keywords*— MapReduce, Hadoop, speculative execution mechanism, homogeneity assumptions, MapReduce performance.

I. INTRODUCTION

MapReduce is proposed by Google in 2004 and has become a popular parallel computing framework for large-scale data processing since then. In a typical MapReduce job, the master divides the input files into multiple map tasks, and then schedules both map tasks and reduce tasks to name nodes in a cluster to achieve parallel processing [2]. When a machine takes an unusually long time to complete a task it will delay the job execution time (the time from job initialized to job completed) and degrade the cluster throughput (the number of jobs completed per second in the cluster) significantly. This problem is handled by speculative execution. In a MapReduce cluster, after a job is submitted a master divides the input files into multiple map tasks and then schedules both the map tasks and the reduce tasks to the nodes. A node runs tasks on its task slots and keeps updating the tasks' progress to the master. Map tasks extract key-value pairs from the input, transfer them to some user defined map function and combine function, and finally generate the map outputs. After that the reduce tasks copy their input from each map task, merge these input to a single ordered (key, value list) pair and transfer this to some user defined reduce function, and finally generate the result for the job.

1.1MapReduce working and Architecture

1. Hadoop breaks the input data into multiple data items by new lines and runs the map function once for each data item, giving the item as the input for the function. When executed, the map function outputs one or more key-value pairs as shown in fig. 1.
2. Hadoop collects all the key-value pairs generated from the map function, sorts them by the key, and groups together the values with the same key.
3. For each distinct key, Hadoop runs the reduce function once while passing the key and list of values for that key as input.
4. The reduce function may output one or more key-value pairs, and Hadoop writes them to a file as the final result.

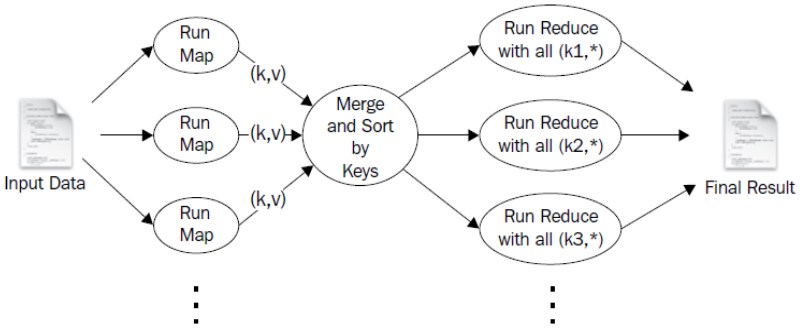

Fig. 1 MapReduce Architecture

© 2014, IJARCSSE All Rights Reserved

II. Related work and Motivation

Several speculative execution strategies have been proposed in the past. Google only begins to do backups when a Map or Reduce operations are close to completion and shows that speculative execution can decrease the job execution time . Speculative execution is also implemented in Hadoop and Microsoft Dryad to improve the MapReduce performance in the cluster. The speculative in Dryad is similar to the one used in Google. However, Hadoop which is a widely used open-source implementation of Mapreduce presents a new strategy called LATE (*Longest Approximate Time to End*) in its Hadoop-0.21 implementation. It monitors the progress rate of the tasks and estimates their remaining time. Tasks with their progress rate below *slowTaskTherehold* are chosen as backup candidates[3] and the one with the longest remaining time is given the highest priority. Later, Microsoft Mantri proposes a new speculative execution strategy for Dryad. Mantri estimates for each task the remaining time to complete that task, *trem*, and predicts the duration of a new copy of the task, *tnew*. Once a slot is idle, Mantri then decides whether to do backup or not based on the statistics of *trem* and *tnew*. In Mantri a duplicate is scheduled if P(*trem* > 2 _ *tnew*) > δ is satisfied. By default, δ = .25. Hence, Mantri[9] schedules a duplicate only if the total resource consumption decreases. Moreover, Mantri allows killing a task if it really takes a long time to finish the remaining work. Mantri cares more about saving the cluster computing resource and less about reducing the job execution time for users. Both Hadoop-LATE and LATE[5] use the average progress rate to select slow tasks and estimate their remaining time. They consider tasks of the same type (map or reduce) process same amount of input data and Progress rate is stable during a task's lifetime. But tasks do not always process the same amount of data and may experience several types of data skew in MapReduce[6]. When the input data has some big records that cannot be divided, the map tasks that process those records will process more data. Partitioning the intermediate data generated by the map tasks unevenly will also lead to the partition skew among the reduce tasks, typically when the distribution of keys in the input data set is skewed[8] this reduces the MapReduce performance. Since most of the MapReduce clusters are shared by users, the performance of worker nodes may degrade just because other users have launched some applications, this can cause some tasks to slow down dramatically. Fig. 2 shows how process speed falls down dramatically. Unfortunately, it will take a long time for them to be identified as stragglers by using the average process speed[10]. For example, suppose a task is running at the speed of *a*% per second for *b* seconds when suddenly the speed falls to *a*/5% per second due to resource competition (shown in Fig 2). The average process speed will fall down to *a*/2% per second only after 5/3 *b* seconds. As a result, the remaining time estimated according to the average process speed will be much shorter than the actual value. Therefore, using the average progress speed cannot identify the straggler tasks in time and can even lead to some misjudgements. LATE and Hadoop-LATE use a threshold (e.g. *slowNode-Threshold*) to identity the straggler nodes. LATE uses the sum of progress of all the completed and running tasks on a worker node to represent the performance score of the node, while Hadoop-LATE uses the average progress rate of all the completed tasks on the node. They both consider a worker node as slow when the performance score of the node is less than the average performance score of all nodes by a threshold, and will never launch any speculative task on this slow node. However, some worker nodes may do more time-consuming tasks and get lower performance score unfairly. For example, they may do more tasks with a larger amount of data to process or they may do more non-local map tasks. As a result, such worker nodes are considered to be slow by mistake. Our proposed enhanced speculative execution mechanism goes one step further by reducing the number of useless speculative tasks. We have established some analytical models to obtain a good speculative execution strategy by utilizing the distribution information of tasks' finish time.

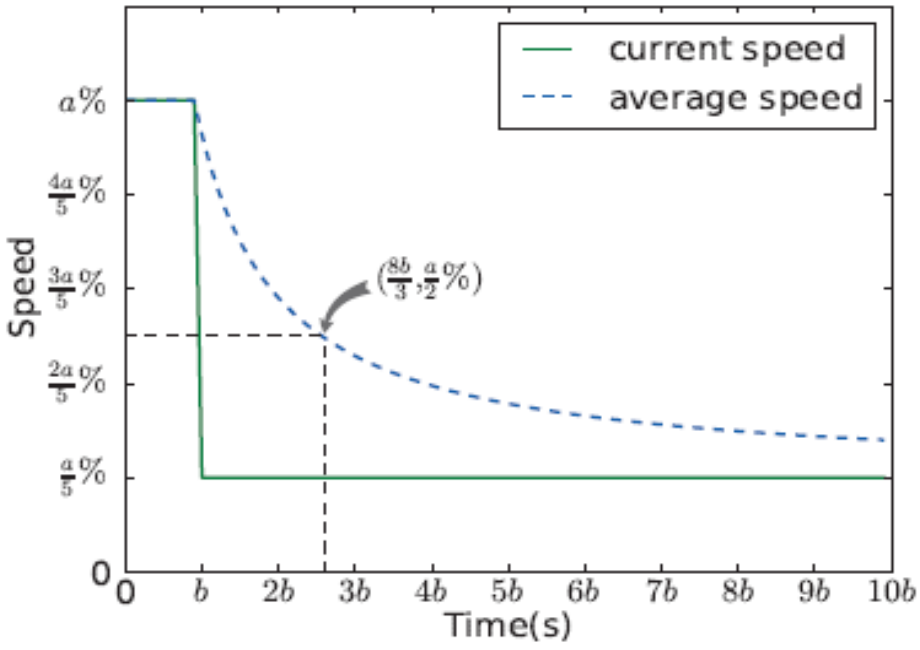

Fig. 2. Process speed falls down dramatically

© 2014, IJARCSSE All Rights Reserved

III. Preliminary Methods

In a homogeneous MapReduce cluster, assume t_i be the duration of task i (the interval of time between the task initialized and task completed) each job consists of multiple map tasks and reduce tasks. Consider this cluster consisted of M identical worker nodes and each node has one map slot and one reduce slot. We use the default FIFO scheduler to schedule the jobs that come into the cluster [5]. Let us assume that this cluster is homogeneous so all the worker nodes are identical. If a task i runs t_i time on a worker node, then it will consume $c * t_i$ amount of resource on this node where c is a constant number[9] there will be a start up time for each task we do not consider it. Fig. 3 shows the reduce task remaining time predicted by multiple methods. We propose a new speculative execution strategy named enhanced speculative execution (ESE) mechanism for effective resource utilization by which job execution time will be shortened and cluster throughput will increase. ESE aims at selecting straggler tasks accurately and backing them up on proper worker nodes. To ensure fairness, we assign task slots in the order the jobs are submitted just like other speculative execution strategies, ESE gives new tasks a higher priority than backup tasks. In other words, ESE will not start backing up straggler map/reduce tasks until all new map/reduce tasks of this job have been assigned. ESE chooses backup candidates based on a prompt prediction of the tasks' process speed and an accurate estimation of their remaining time. Then, these backup candidates will be selectively backed up on proper worker nodes to achieve effective resource utilization.

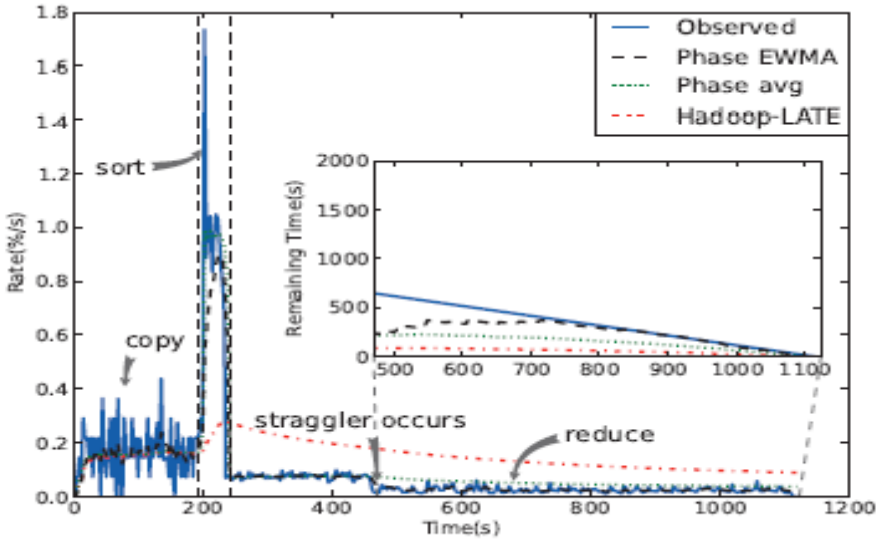

fig. 3. Reduce task remaining time predicted by multiple methods

3.1 Estimating task remaining time

In this phase the remaining time to complete the task is estimated by the sum of the remaining time left in each phase. When a task is running in some phase c_p (i.e., the current phase), the remaining time left in c_p is estimated by the factors of the remain data and the process bandwidth in c_p[4]. However, the remaining time of the following phases f_p is difficult to calculate since the task has not entered those phases yet. Therefore, we use the phase average process speed to estimate the remaining time of a phase $est.time_p$. The phase average process speed is the average process speed of tasks that have entered the phase. For those phases that no task has entered, we do not calculate their remaining time. Since tasks may process different amount of data, it is adjusted to $est\ time_p$ by $factor_d$, which represents the ratio of the input size of this task to the average input size of all tasks[10]. Now we can estimate the remaining time of tasks as follows:

$$rem\_time = rem\_time_{cp} + rem\_time_{fp} = \frac{rem.data_{cp}}{bandwidth_{cp}} + \sum_{p\ in\ fp} est.time_p * factor_d \quad (1)$$

$$factor_d = \frac{data.input}{data.avg} \quad (2)$$

To estimate the backup time of a slow task, we use the sum of $est.time_p$ [10] for each phase in this task as an estimation. Therefore, we can calculate the backup time as follows:

$$backup\_time = \sum_p est\_time_p * factor_d \quad (3)$$

when reduce tasks that start later are running in the copy phase, their process speed can be fast at the beginning and drop later[10]. This will cause process speed fluctuation and impact the precision of time estimate. To avoid such impact, we estimate the remaining time of the copy phase as the time to copy all the completed map task outputs. We calculate the remaining time of the copy phase using the following equation:

$$rem\_time_{copy} = \frac{finish.percent_{map} - finish.percent_{copy}}{process.speed_{copy}} \quad (4)$$

© 2014, IJARCSSE All Rights Reserved

In the equation above, the *process speed copy* is estimated by weighted moving average(WMA). The equation is reasonable because the process percentage of the copy phase in reduce tasks is limited by the percentage of completed map tasks. Simply backing up a task that has the longest remaining time is not proper, as we can always get a task that has the longest remaining time. To make sure the task is slow enough, we consider whether backing up this task can effectively save cluster computing resources.

3.2 Selecting a proper node to backup the task

When the cluster is heavily loaded, i.e., the total number of slots is smaller than the number of tasks in job *J* then a speculative execution mechanism[1] has to be implemented to improve the performance of MapReduce frame work. When $M < N$(M is the number of slots available, N is the number of tasks present in a job) which indicates that at the beginning we cannot launch all the tasks of job *J* in the cluster. We extend the Microsoft's scheme and propose the *Enhanced Speculative Execution* (ESE) algorithm. The corresponding pseudo code is given in Algorithm 1.

Algorithm 1. ESE algorithm

1: if nodes are available do
2: else
3: if tasks are waiting for nodes then
4: if there exists some tasks that are running currently which satisfy the condition
$t_{rem} > \sigma$ _ E[t_{new}] then
5: choose the task with longest remaining time to be backup in this node.
6: else
7: Backup the waiting task that has the largest data to read.
8: end if
9: end if

in the above algorithm t_{rem} refers to the remaining time to complete a particular task on a node and E[t_{new}] refers to the estimated time to complete the slow task on an alternative node.

IV. Conclusion and Future work

As some large-scale clusters exhibit inefficient resource utilization. We propose resource stealing to improve resource utilization by aggressively harnessing the portion of unutilized resources. Speculative execution in Hadoop was observed to be inefficient, which is caused by the excessive runs of useless speculative tasks. Our proposed Enhanced Speculative Execution manages speculative tasks and expected to improve the efficiency. Finally, various scheduling algorithms, which take into consideration the real-time network performance [7], are proposed to alleviate the performance degradation caused by network heterogeneity. In addition, we will further improve the proposed heterogeneity-aware scheduling heuristics of map and reduce tasks to make them more efficient and robust. As for data replication in HDFS, we will study how to calculate the best propagation path to minimize the replication time and thus improve the data locality of MapReduce jobs. Both storage and computation will be considered simultaneously to maximize the potential performance improvement.

Acknowledgment

The authors would like to acknowledge with gratitude and appreciation, the support provided by the staff of Computer Science lab. This work is supported by SRM University, Chennai, India.

References

[1] J. Dean and S. Ghemawat, "*Mapreduce: simplified data processing on large clusters,*" *Commun. ACM*, vol. 51, pp. 107–113, January 2008.
[2] "Apache hadoop, http://hadoop.apache.org/."
[3] 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.
[4] B. Ucar, C. Aykanat, K. Kaya, and M. Ikinci, "*Task assignment in heterogeneous computing systems*," *J. Parallel Distrib. Comput.*, vol. 66, pp. 32–46, January 2006.
[5] "Apache hadoop 2.0, http://hadoop.apache.org/docs/r2.0.0-alpha/."
[6] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "*Dryad: distributed data-parallel programs from sequential building blocks,*" in *Proceeding of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07, 2007.
[7] Xiaoyu Sun, Chen He, Ying Lu, "*ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm,*" in *the 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 2012 IEEE.
[8] J.-A. Quiane-Ruiz, C. Pinkel, J. Schad, and J. Dittrich, "*Rafting mapreduce: Fast recovery on the raft,*" in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, april 2011.
[9] Wikipedia: Apache Hadoop. http://en.wikipedia.org/wiki/Hadoop.
[10] Qi Chen, Cheng Liu, and Zhen Xiao, "*Improving MapReduce Performance Using Smart Speculative Execution Strategy,*" to appear in *IEEE Transactions on Computers* (TC).

© 2014, IJARCSSE All Rights Reserved