A Load-Aware Scheduler for MapReduce Framework in Heterogeneous Cloud Environments

Hsin-Han You, Chun-Chung Yang and Jiun-Long Huang
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, ROC
E-mail: hhyou@cs.nctu.edu.tw, chunchung@cs.nctu.edu.tw, jlhuang@cs.nctu.edu.tw

ABSTRACT

MapReduce is becoming a popular programming model for large-scale data processing in cloud computing environments. Hadoop MapReduce is the most popular open-source implementation of MapReduce framework. Hadoop MapReduce comes with a pluggable task scheduler interface as well as a default FIFO job scheduler. The default Hadoop scheduler only considers the homogeneous environments, and thus does not perform well in heterogenous environments. Although being proposed to schedule tasks/jobs in heterogenous environments, the LATE scheduler does not consider the phenomenon of dynamic loading which is common in practice. In view of this, we propose a new scheduler named Load-Aware scheduler, abbreviated as the LA scheduler, to address the problem resulting from the phenomenon of dynamic loading, thus being able to improve the overall performance of Hadoop clusters. Experimental results show that the LA scheduler is able to reduce up to 20% in average response time by avoiding unnecessary speculative tasks.

Keywords: Cloud computing, Dynamic loading, Hadoop, MapReduce, Scheduling, Speculative execution

1. INTRODUCTION

Data are becoming larger and larger everyday and in every field. Google claimed that in 2008, they obtain 20 petabytes of raw data everyday [6], while Facebook claimed that they have processed over 12 terabytes of compressed raw data per day [10]. This phenomenon happens not only in Web applications. For example, NYSE (New York Stock Exchange) generates 1 terabyte of data per day and LHC (Large Hadron Collider) in Geneva produces 15 petabytes of data every year [12]. These data are huge, hard to be stored, maintained, and even processed. Therefore, Google proposed MapReduce [6] which is a distributed programming framework that provides failure recovery, scalability and job monitoring, to address such problems.

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

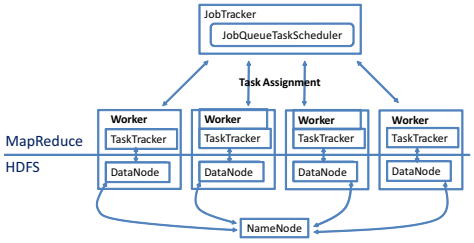


Figure 1: System architecture of Hadoop

Since being published in [5], MapReduce framework has been widely discussed and currently is the most popular framework for large-scale data processing. Some researches leverage MapReduce to obtain scalability and performance enhancement for their applications. [4] discusses machine learning algorithms on MapReduce framework. [8] points out that data mining algorithms need a good framework for large-scale data processing and implements a co-clustering algorithm on MapReduce framework. Some papers such as [3] and [11] treat MapReduce as a framework for data warehouse or data storage. There are also researches that implement MapReduce framework on special hardware like GPUs [7] or multi-core chips [9].

In MapReduce framework, every MapReduce job has to implement two functions: map and reduce [6]. Given the input data, MapReduce framework will automatically partition the input data into several splits, and these splits are then processed by different instances of map function. An instance of map function is called a *mapper* while an instance of reduce function is called a *reducer*. Each mapper first parses the corresponding input data split into key/value pairs. Then, for each key/value pair, the mapper outputs some or no intermediate key/value pair(s). All intermediate key/value pairs will be sorted so that the values corresponding to the same key will be aggregated together and processed by the same reducer. Each reducer will iterate over its value set, and output some or no key/value pair(s) as the result.

Since Hadoop MapReduce is the most popular open source implementation of MapReduce framework, we use the terminology of the Hadoop community in the rest of this paper. Figure 1 shows the architecture of Hadoop MapReduce. As shown in Figure 1, Hadoop MapReduce consists

of one master node called *JobTracker* and some slave nodes called *TaskTrackers*. Users submit MapReduce jobs to the JobTracker, and the JobTracker will split each job into several tasks[1] and assign these tasks to the TaskTrackers. Similar to most distributed systems, how tasks are scheduled will greatly influence the performance of Hadoop clusters. Hadoop MapReduce provides a default scheduler that schedules jobs in a FIFO manner as well as a TaskScheduler interface for developers to design their own schedulers. With the open TaskScheduler interface, both Facebook and Yahoo! have implemented and contributed their schedulers, named FairScheduler [2] and CapacityScheduler [1], respectively.

In order to facilitate fault tolerance, when a TaskTracker performs badly or crashes, the JobTracker will re-assign the current tasks on the failed TaskTracker to other TaskTrackers. Such technique is called *speculative execution* and this type of tasks is called *speculative tasks*. In addition to facilitating fault tolerance, speculative execution is also able to reduce job response time [6]. It is reported in [6] that speculative execution can result in about 44% reduction in job response time. However, previous study [13] has shown that the default speculative task scheduler in Hadoop is designed for homogeneous environments (i.e., each TaskTracker is of equal capability) and does not perform well in heterogeneous environments (i.e., each TaskTracker is of different capability). As a result, the authors of [13] proposed a speculative task scheduler, called the LATE scheduler, to choose more suitable tasks to be speculatively executed. The experimental result shows that using the LATE scheduler is able to significantly shorten the job response time in heterogeneous environments [13].

However, we observe that, in the LATE scheduler, a MapReduce job might launch many unnecessary[2] speculative tasks in the last wave of tasks due to the following reasons.[3]

- *The LATE scheduler is designed to minimize the response time of first job in the job queue, and will prolong the response time of the other jobs in the job queue.*
- *The LATE scheduler uses the past information to estimate the time to finish of tasks and is not suitable for environments with dynamic loading.*

In view of this, we propose a scheduler, called the Load-Aware scheduler (abbreviated as the LA scheduler), for heterogeneous environments with dynamic loading. The LA scheduler consists of two modules: the data collection module and the task assignment module. The data collection module gathers the system-level information of all TaskTrackers periodically while the task assignment module makes scheduling decisions according to the TaskTrackers' information collected by the data collection module. To facilitate task scheduling on heterogeneous environments with dynamic loading, a task response time estimation method based on the collected system-level information is also proposed. We implement the LA scheduler into Hadoop for performance evaluation. Experimental results show that the LA scheduler is able to reduce up to 20% in average response time by avoiding unnecessary speculative tasks.

[1]A mapper or a reducer is called a *task*.
[2]A speculative task is called *necessary* if launching the speculative task will shorten the response time of the corresponding job.
[3]For better readability, the details are given in Section 3.

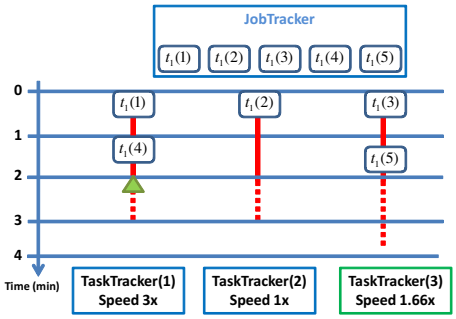


Figure 2: Consideration of node heterogeneity

The rest of this paper is organized as follows: Section 2 introduces some related work on scheduling in MapReduce framework. Section 3 describes the design of our speculative task scheduler. Some implementation details are also given. Several experiments are conducted to measure the performance of our scheduler and the experimental results are given in Section 4. Finally, we make a conclusion in Section 5.

2. RELATED WORK

2.1 Default Speculative Task Scheduler of Hadoop

The default Hadoop scheduler for speculative tasks assumes that each TaskTracker is of the same capability. Tasks of the same job should finish almost in the same time. When the progress of a task is below the average progress of other tasks of the same job, the JobTracker will suspect that the TaskTracker is encountering some problem and will assign a speculative task for this task on another available TaskTracker.

2.2 LATE Scheduler

Zaharia et. al. points out in [13] that the default speculative task scheduler of Hadoop does not perform well in heterogeneous environments (i.e., TaskTrackers are of different capability). Since the default speculative task scheduler of Hadoop assumes that TaskTrackers are homogeneous and tasks tend to finish in waves, schedulers built on this assumption will result in some bad choices in choosing which tasks to be speculatively executed.

We use the example in Figure 2 to illustrate how the default scheduler speculates the wrong task. Suppose that we have a job with 5 tasks and a cluster with 3 TaskTrackers. TaskTrackers are of different processing speed: 3X, 1X and 1.66X. Let $t_i(j)$ be the j-th task of job i. At first, task $t_1(1)$, $t_1(2)$ and $t_1(3)$ are assigned to TaskTrackers and are estimated to finish in 1, 3 and 1.8 minutes. After one minute, TaskTracker 1 finishes task $t_1(1)$ and task $t_1(4)$ is assigned to TaskTracker 1. 0.8 minute later, TaskTracker 3 finishes task $t_1(3)$ and task $t_1(5)$ is assigned to TaskTracker 3. 0.2

minute later, TaskTracker 1 finishes task $t_1(4)$, so it has an empty working slot. Task $t_1(2)$ and task $t_1(5)$ are both candidates to be executed speculatively. It is obvious that launching a speculative task for task $t_1(5)$ can make job 1 finish within 3 minutes while launching speculative task for task $t_1(2)$ will not help at all. In the above case, without considering the different speed of TaskTrackers, the default scheduler might speculatively execute task $t_1(2)$ at minute 2 since the progress of task $t_1(2)$ is beyond the average progress of active tasks.

In view of this, Zaharia et. al. proposed in [13] the LATE scheduler to avoid such problem by introducing the concept of *progress rate*.[4] Since the time to finish estimation formula proposed in [13] will take the capability of TaskTrackers into account, the LATE scheduler will perform better than the default speculative task scheduler of Hadoop in heterogeneous environments.

3. LOAD-AWARE SCHEDULER

3.1 Motivation

Although being able to outperform the default speculative task scheduler of Hadoop in heterogeneous environments, the LATE scheduler is of the following two drawbacks.

1. *The LATE scheduler is designed to minimize the response time of first job in the job queue, and will prolong the response time of the other jobs in the job queue.*

   We use the example in Figure 3 to illustrate such phenomenon. Suppose that these two TaskTrackers are of different hardware so that TaskTracker 1 is 1.5 faster than TaskTracker 2. A user submits two jobs, job 1 with 4 tasks and job 2 with 2 tasks. Let $t_i(j)$ represent the j-th task of job i. Assume that each task of job 1 will run one minute on TaskTracker 1 and 1.5 minute on TaskTracker 2. Since being designed to optimize the first job in the job queue, the LATE scheduler will launch a speculative task for job 1 (i.e., task $t_1(4)$) in the second minute. However, it is obvious that launching a speculative task for task $t_1(4)$ will not shorten the response time of job 1. Thus, the resource for such speculative task is totally wasted.

2. *The LATE scheduler uses the past progress rate to estimate the time to finish of tasks and is not suitable for environments with dynamic loading.*

   LATE scheduler does not consider the fact that performance of each TaskTracker might vary from time to time. LATE scheduler uses a heuristic to evaluate node performance by the sum of its previous working progress. This heuristic might misjudge some nodes as slow nodes if those nodes happened to run some heavy, non-Hadoop processes.

   We use the example in Figure 4 to illustrate such phenomenon. There are heavy non-Hadoop processes in TaskTracker 2 and TaskTracker 3 (H.264 compression in this example), and these heavy process will terminate in the second minute. In the second minute, a speculative task $t_1(2)$ is launched in TaskTracker 3 since the progress rate of TaskTracker 2 is quite low. Several seconds later, the scheduler has to determine whether to launch another speculative task for task $t_1(2)$ or to execute an ordinary task $t_2(1)$. Since the heavy process in TaskTracker 3 is finished, the computation power of TaskTracker 3 is dedicated for Hadoop after the second minute. Since using the past progress rate to predict the progress rate in the future, the LATE scheduler will consider TaskTracker 3 is a slow node (even though the heavy, non-Hadoop process in TaskTracker 3 has finished) and will launch a speculative task for task $t_1(2)$. However, it is obvious that computation power of TaskTracker 3 on processing Hadoop tasks is higher than before. Thus, launching a speculative task $t_1(2)$ in TaskTracker 1 will not shorten the response time of job 1, thereby wasting the resource in TaskTracker 1.

[4]Interested readers can refer to [13] for the detail description of progress rate.

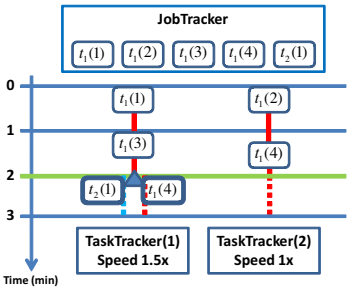


Figure 3: Consideration of multiple jobs

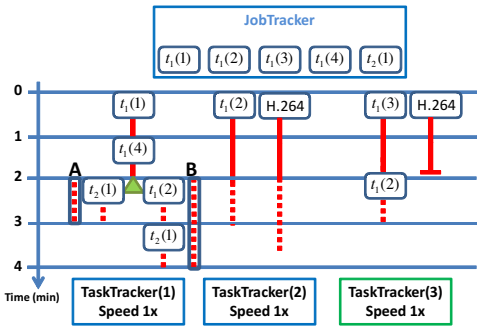


Figure 4: Consideration of dynamic loading

In view of this, we have the following two design guidelines to address the above problems.

- *A speculative task will be launched only when launching such speculative task will shorten the response time of the corresponding job.*

  Consider the case that the scheduler is determining whether to launch a speculative task for $t_i(j)$. If the current execution of task $t_i(j)$ will finish before the speculative task or other ongoing tasks of job j will not finish after the speculative task, we should not

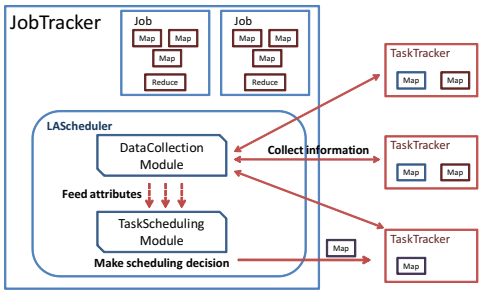


Figure 5: Load-Aware scheduler architecture

launch such speculative task and will launch an ordinary task for the next job in the job queue. Thus, in the example in Figure 3, we should assign a task of the second job in the job queue (i.e., task $t_2(1)$) to TaskTracker 1 at the second minute to improve the utilization of the Hadoop cluster.

- *Use current system-level information instead of the past Hadoop-level information to estimate the response time of tasks.*

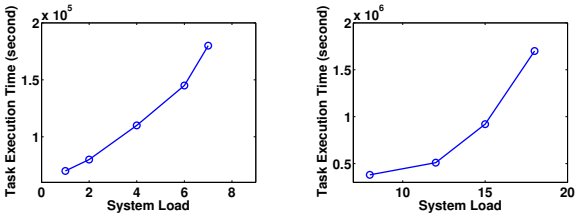
  In practice, a company usually establishes a cluster running several cluster frameworks. For example, a large Taiwan semiconduction manufacturing company establishes a cluster running Hadoop MapReduce, HBase[5] and GridGain[6]. Thus, the system loading of nodes will vary from time to time and cannot be obtained from Hadoop-level information. Fortunately, using current system-level information on task response time estimation will tackle the problem resulting from running heavy, non-Hadoop processes. In the example in Figure 4, when the heavy, non-Hadoop process in TaskTracker 3 has finished, the system load in TaskTracker 3 will become lower. By using current system-level information of TaskTracker 3, the scheduler is able to estimate the response time of speculative task $t_1(2)$ in TaskTracker 3 and will make correct decision in the second minute. To achieve this, a response time estimation method using system-level information is required.

3.2 The Proposed Scheduler

We propose in this subsection the Load-Aware scheduler, abbreviated as the LA scheduler, based on the design guidelines in Section 3.1. The LA scheduler consists of two modules, the data collection module and the task scheduling module, in the JobTracker and the architecture of the LA scheduler is shown in Figure 5.

The data collection module provides an interface for the JobTracker to gather system-information of TaskTrackers to estimate the response time of tasks. Attribute that might affect the performance of a TaskTracker can be put into the collection list of the data collection module. CPU frequency, system loading, I/O rate and memory usage are general attributes that might be selected into the collection list.

[5]HBase, `http://hbase.apache.org`.
[6]GridGain, `http://www.gridgain.com`.



(a) System Loading Smaller than the Number of Cores (b) System Loading Larger than the Number of Cores

Figure 6: System loading versus Task response time

The task scheduling module periodically estimates the response time of each task according to the system-level information of TaskTrackers collected by the data collection module. The details of response time estimation are given in Section 3.3. Whenever an empty working slot is available on a TaskTracker, the task scheduling module uses the system-level information of TaskTrackers to estimate response time of tasks, and schedules a task to a TaskTracker by the following steps.

- Step 1: Check if there are tasks of the job need to be scheduled (including failed tasks, non-running tasks and tasks need to be executed speculatively). If there is no such task, continue Step 1 to check the next job.
- Step 2: Estimate the task response time on the TaskTracker, and check whether schedule this task can shorten the job's response time. If not, continue Step 1 to check the next job.
- Step 3: Assign a task of the current job to the TaskTracker.

3.3 Implementation

We implement the Load-Aware scheduler into Hadoop MapReduce by merging our modules into FairScheduler. Thus, our scheduler still retains the fairness characteristic of FairScheduler since we only modify the scheduler on speculative tasks to avoid unnecessary speculative tasks.

For data collection module, we need an information exchange mechanism between the JobTracker and each TaskTracker, and we choose SNMP as our data collection protocol. The data collection module in the JobTracker will send a SNMP query to each TaskTracker periodically[7] to retrieve the TaskTracker's system-level information such as system loading[8], CPU frequency, the number of cores, memory usage, and so on.

To implement the task scheduling module, we need a method to estimate response time of tasks. Our target in this paper is CPU-intensive jobs. System loading and CPU frequency are considered in our prototype. System loading gives us a global view of CPU usage of a TaskTracker.

[7]The size of a SNMP query packet is 87 bytes and the size of a SNMP result packet takes around 93 bytes.
[8]System loading is a common metric to measure the load of CPU(s) in Unix-like platforms.

High loading means that there are fewer resources available for processing new-coming tasks. We perform several experiments to observe the relation between system loading and task response time, and the results are shown in Figure 6. Our node is of eight cores and we can observe that the growing curve differs when system loading becomes larger than the number of cores in the node. We can see in Figure 6(a) that the response time increases linearly when the number of cores is smaller than or equal to eight. On the other hand, as shown in Figure 6(b), the response time increases exponentially when the number of cores becomes larger than eight. Thus, we use the following equations to approximate the experimental results in Figure 6:

$$Time' = 17700 * (Load' - Load) * Time \quad (1)$$

$$Time' = Time * e^{0.12*(Load'-Load)} \quad (2)$$

Equation 1 is used when system load is smaller than or equal to the number cores, while equaton 2 is used when system load is larger than or equal to the number cores. Given the fact that the response time of a task of a job running on a TaskTracker under system loading $Load$ is $Time$, we can estimate the response time $Time'$ of the task of the same job running a TaskTracker under system loading $Load'$ by the above formulas.

We use the same technique to predict the effect of CPU frequency. We perform an experiment to observe the relation between CPU frequency and task response time, and the experimental results are shown in Figure 7. Thus, we have an estimation function of CPU frequency:

$$Time' = Time * Freq'/Freq \quad (3)$$

Given the fact that the response time of a task of a job, say task $t_i(j)$, running on a TaskTracker with CPU frequency $Freq$ is $Time$, we can estimate the response time $Time'$ of the task of the same job, say task $t_i(j')$ running a TaskTracker with CPU frequency $Load'$ by Equation 3.

With the above equations, the task scheduling module is able to estimate response time of task $t_i(j')$ on a TaskTracker with CPU frequency $Freq'$ under system loading $Load'$ by the following steps.

- Step 1: Select a finished task of job i, say task $t_i(j)$. Suppose that task $t_i(j)$ is finished by a TaskTracker with CPU frequency $Freq$ under system loading $Load$.
- Use Equation 1 or Equation 2 to estimate response time, denoted as $Time^*$ of task $t_i(j)$ on a TaskTracker with CPU frequency $Freq$ under system loading $Load'$.
- Apply Equation 3 by setting the value of variable $Time$ to $Time^*$ to estimate response time of task $t_i(j')$ on the TaskTracker with CPU frequency $Freq$ under system loading $Load'$.

4. PERFORMANCE EVALUATION

Our testbed is a Hadoop cluster consisting of 8 nodes. Each node has one Xeon CPU with four cores supporting Hyper-Threading (thus, 8 cores in the operating system's perspective), 12GB of RAM and a 500GB SATA drive. All

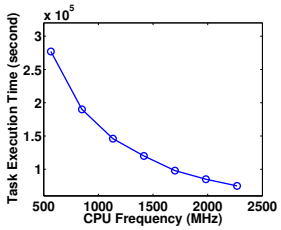
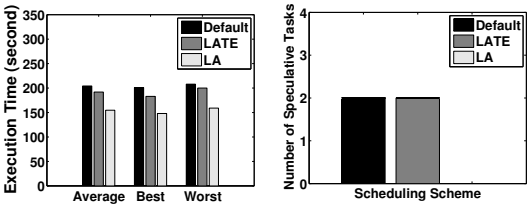


Figure 7: CPU frequency versus Task execution time



(a) Total Response Time

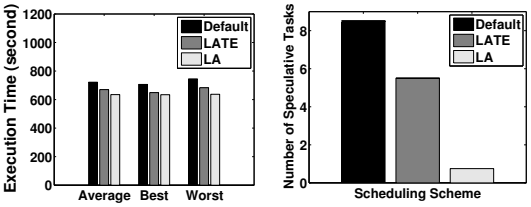
(b) Number of Speculative Tasks

Figure 8: Impact of Multi-Job Consideration

nodes are connected via a gigabit Ethernet channel. The load of each node might vary from time to time in order to observe the impact of dynamic loading. We extend a local sudoku game solver to a MapReduce version that solves multiple sudoku games in our Hadoop cluster. Input sudoku games are randomly generated in advanced by a soduku game generator. Two performance metrics, the total response time of a set of jobs and the number of speculative tasks launched, are used to measure the performance of the default speculative task scheduler (denoted as Default), the LATE scheduler [13] and our scheduler (denoted as LA).

4.1 Impact of Multi-Job Consideration

To evaluate the impact of multi-job consideration, two jobs are submitted into our Hadoop cluster and each job takes 9000 sudoku games as their input. When the first job is almost finished, some speculative tasks might be launched. The system loading of TaskTracker 1 is set to 6. Figure 8(a) shows the total response time of these two jobs. On average, the LA scheduler is about 20% faster than other schedulers. It is due to the reason that the Default scheduler and the LATE scheduler will launch some speculative tasks to try to shorten the response time of the first job without checking the necessity of speculative tasks. Thus, some unnecessary speculative tasks will be launched and TaskTrackers will waste their CPU time on some unnecessary speculative tasks. In addition, using too many resources on minimizing the response time of the first job will probability prolong the response time of the other jobs in the job queue. Although the LATE scheduler is able to choose the better tasks to be speculatively executed, the number of speculative tasks launched by the LATE scheduler is equal to the number of speculative tasks launched by the Default scheduler. On the other hand, the LA scheduler only launches the speculative tasks that will shorten the re-



(a) Total Response Time

(b) Number of Speculative Tasks

Figure 9: Impact of Dynamic Loading

sponse time of the corresponding jobs. The LA scheduler does not waste resource on unnecessary speculative tasks, thus reducing the total response time of these two jobs. We can observe the above phenomenon in Figure 8(b).

4.2 Impact of Dynamic Loading

The dynamic loading scenario is set as follows. We submit 4 jobs which process 30000, 20000, 10000 and 10000 sudoku games, respectively. TaskTrackers 1-4 execute some heavy non-Hadoop processes at the beginning. The system loading of TaskTrackers 1 and 2 is 4 from the beginning to the third minutes, and increases to 8 from the third minutes. The system loading of TaskTrackers 3 and 4 is 4 from the beginning to the third minute. TaskTrackers 3 and 4 are dedicated for Hadoop after the third minute. TaskTrackers 5-8 are dedicated for Hadoop throughout the experiment.

Since the default Hadoop scheduler considers that each node is of equal capability, a speculative task will be launched when an active task performs badly. The LATE scheduler records how each node performed in the past, and launches speculative tasks only on the nodes which performs well in the past. Since TaskTrackers 3 and 4 are dedicated for Hadoop after the third minute, the JobTracker can schedule some speculative tasks to TaskTrackers 3 and 4 to shorten total response time. Unfortunately, the LATE scheduler will still treat TaskTracker 3 and 4 as slow nodes according to their past performance. On the other hand, since periodically gathering system loading of TaskTrackers, the LA scheduler can use TaskTrackers' current status (e.g., system loading) to estimate task response time. Thus, TaskTracker 3 and 4 will be recognized as fast node after the third minute. Figure 9 shows that the LA scheduler can improve up to 15% in terms of total response time by avoiding 90% speculative tasks.

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed the LA scheduler for MapReduce framework to improve the utilization of a MapReduce cluster by avoiding unnecessary speculative tasks on heterogeneous environments with dynamic loading. To measure the performance of the LA scheduler, we implemented it on a popular open source MapReduce framework, Hadoop MapReduce. Experimental results show that the LA scheduler is able to reduce up to 20% in average response time by avoiding unnecessary speculative tasks. In the future, we will study response time estimation to design novel response time estimation methods for Hadoop clusters.

6. REFERENCES

[1] Hadoop CapacityScheduler. `http://hadoop.apache.org/common/docs/current/capacity_scheduler.html`.

[2] Hadoop FairScheduler. `http://hadoop.apache.org/common/docs/current/fair_scheduler.html`.

[3] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In *Proceedings of the 35th International Conference on Very Large Data Bases*, 2009.

[4] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-Reduce for Machine Learning on Multicore. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems*, 2006.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified Data processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design and Implementation*, 2004.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1), 2008.

[7] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars: A MapReduce Framework on Graphics Processors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[8] S. Papadimitriou and J. Sun. DisCo: Distributed Co-clustering with Map-Reduce: A Case Study towards Petabyte-Scale End-to-End Mining. In *Proceedings of the 8th IEEE International Conference on Data Mining*, 2009.

[9] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, 2007.

[10] A. Thusoo and N. Jain. Facebook's Petabyte Scale Data Warehouse using Hive and Hadoop. `http://www.infoq.com/presentations/Facebook-Hive-Hadoop`.

[11] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murth. Hive - A Warehousing Solution Over a Map-Reduce Framework. In *Proceedings of the 35th International Conference on Very Large Data Bases*, 2009.

[12] R. Varela. Introduction to Data Processing using Hadoop and Pig. `http://www.slideshare.net/phobeo/introduction-to-data-processing-using-hadoop-and-pig`.

[13] M. Zaharia, A. Konwinski, A. D. Joseph, Y. Katz, and I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, 2008.