Research Article

Optimized Speculative Execution to Improve Performance of MapReduce Jobs on Virtualized Computing Environment

Lei Yang,¹ Yu Dai,² and Bin Zhang¹

¹College of Computer Science and Engineering, Northeastern University, Shenyang, China
²College of Software, Northeastern University, Shenyang, China

Correspondence should be addressed to Yu Dai; daiy@swc.neu.edu.cn

Received 26 April 2017; Revised 19 September 2017; Accepted 8 October 2017; Published 7 December 2017

Academic Editor: Mauro Gaggero

Copyright © 2017 Lei Yang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Recently, virtualization has become more and more important in the cloud computing to support efficient flexible resource provisioning. However, the performance interference among virtual machines may affect the efficiency of the resource provisioning. In a virtualized environment, where multiple MapReduce applications are deployed, the performance interference can also affect the performance of the Map and Reduce tasks resulting in the performance degradation of the MapReduce jobs. Then, in order to ensure the performance of the MapReduce jobs, a framework for scheduling the MapReduce jobs with the consideration of the performance interference among the virtual machines is proposed. The core of the framework is to identify the straggler tasks in a job and back up these tasks to make the backed up one overtake the original tasks in order to reduce the overall response time of the job. Then, to identify the straggler task, this paper uses a method for predicting the performance interference degree. A method for scheduling the backing-up tasks is presented. To verify the effectiveness of our framework, a set of experiments are done. The experiments show that the proposed framework has better performance in the virtual cluster compared with the current speculative execution framework.

1. Introduction

Recently, the MapReduce [1, 2] as a platform for massive data analysis has been widely adopted by most of companies for processing large body of data to correlate, mine, and extract valuable features. With the prevailing of the virtualized techniques, the virtual clusters can provide much more flexible mechanism for different applications sharing the common computing resources. Then, currently, lots of MapReduce jobs are deployed in a virtual cluster. However, the modern virtual techniques do not provide perfect performance isolation mechanism, for example, Xen [3], which may cause the virtual machines to compete for the limited resource and result in the performance interference among the virtual machines. Then, how to ensure the performance of the MapReduce job in the virtual cluster becomes a key issue.

Previous works focusing on the performance of the MapReduce job have indicated the performance degradation in the virtual clusters [4–7]. Other researchers have found that the performance interference [8–10] is one of the important factors causing such degradation. Then, a set of works in the field of task scheduling were conducted [11–13] to ensure the performance of the MapReduce applications. However, most of them only focus on I/O intensive applications and try to find a uniform performance interference model to predict the performance degradation for different types of the applications. In fact, for different applications, using a uniform model to evaluate its performance may not always work well.

In this paper, we present an optimized speculative execution framework for MapReduce jobs which aims to improve the performance of the jobs in the virtual clusters. The contribution of the paper is as follows.

(1) In order to predict the performance degradation, a method for predicting the performance degree is proposed. In this method, the linear regression model is used to reflect the performance degree and the system workloads and a
swarm particle algorithm is used for finding the coefficients in the model.

(2) In order to find the stragglers, the method for computing the remaining time of the task is presented with the consideration of the performance interference degree.

(3) In order to back up the stragglers, a scheduling algorithm is proposed which assigns the tasks to the slot with a global optimization.

The organization of the rest of the paper is as follows. The next part introduces the current works related to the MapReduce scheduling in the virtual cluster. Section 3 overviews our speculative execution framework. Sections 4 and 5 show how to predict the performance interference degree, identify the stragglers, and schedule the tasks. Section 6 presents the experimental result to verify our methods. Finally the paper is summarized in Section 7.

2. Related Works

Currently, lots of works in the field of performance analysis in the virtual cluster are conducted. Reference [14] presents a method for predicting the interthread cache conflicts based on the hardware activity vector. Reference [15] provides a method to characterize the application performance in order to predict the overheads caused by the virtualization. Reference [16] uses an artificial neural network to predict the application performance. References [17, 18] analyze the network I/O contention in the cloud environment. Performance interference among the CPU-intensive applications has been discussed in [11]. Reference [12] considers the performance interference of the disk I/O intensive applications and proposes a model for predicting such interference. Reference [8] analyzes the factors related to the performance interference and presents a method for estimating it. Reference [19] targets the problem of application scheduling in data centers with the consideration of the heterogeneity and the interference. Although some of the current works have noticed the performance interference and the MapReduce applications' performance caused by such interference, they only focus on I/O intensive applications and try to find a uniform performance interference model to evaluate the performance degradation for different types of the applications. In fact, for different applications, using a uniform model to evaluate its performance may not always work well as the resource usage pattern can be very different. Besides, the method proposed in [19] develops several microbenchmarks to derive interference sensitivity scores and uses a collaborative filtering method to induce the sensitivity score for a new arrival application which needs the application to run against at least 2 microbenchmarks for 1 minute to get its profile. Then, as the method relies on the microbenchmarks for analyzing the interference degree, the diversity of the microbenchmarks will affect the accuracy of the analysis. Besides, if the diversity number of the microbenchmarks is large, the score matrix for the collaborative filtering may be very sparse as the new application cannot run against many microbenchmarks for 1 minute before inducing the interference sensitivity score. Then, the collaborative filtering method may not work well as for the sparse matrix. Although some methods have been proposed to solve this problem, the effect is not very good. Meanwhile, in the field of the MapReduce job scheduling, the QoS may depend not only on the interference, but also the factor of the data locality. Then, making the MapReduce job run against the microbenchmarks may not reflect its actual performance and get its actual profile. MapReduce job may need to read the data file remotely from the microbenchmarks. Then, the runtime of the job under this situation may be different from the runtime when the job need not read input data files remotely. In this sense, the method proposed in [19] may not be used in the field of MapReduce job scheduling.

Many researchers have put their efforts in the field of task scheduling in MapReduce. Reference [20] proposes a capacity scheduler to guarantee the fairly share of the capacity of the cluster among different users. To ensure the data locality, [21] proposes a delay scheduler. With this technique, if the head-of-line job cannot launch a local task, the scheduler can delay it and look at the subsequent job. When a job has been delayed for more than the maximum delay time, the scheduler will assign the job's nonlocal map tasks. Reference [22] uses a linear regression method to model the relation between the I/O intensive applications. Reference [23] uses node status prediction to improve the data locality rate. Reference [24] uses a matching algorithm for scheduling not only considering the data locality but also respecting the cluster utilization. Reference [25] introduces a Quincy scheduler to achieve data locality. Several recent proposals, such as resource-aware adaptive scheduling [26] and cost effective resource provisioning [27], have introduced resource-aware job schedulers to the MapReduce framework. Reference [28] mentions the problem of task assignment with the consideration of the data locality in cloud computing. Reference [29] focuses on the scheduling with the consideration of the data locality to minimize the cost caused by accessing remote files. Reference [30] proposes a scheduling algorithm to make the jobs meet the SLAs. Reference [31] solves the problem of job scheduling with the consideration of the fairness as well as the data locality. Reference [19] proposes a method for application scheduling with the consideration of the interference and a greed algorithm is presented for finding the optimal assignments. However, this method is only for single application. As for our problem, we need to find optimal assignments in each time interval for a set of tasks. As stated above, most of the current works assume the perfect performance isolation among virtual machines. Then, based on such an assumption, current works seldom consider the performance interference. As stated above, some of the works consider the performance interference; for example, in [22], the scheduler optimizes the assignment with the consideration of only one task or only one slot while it is hard to achieve the global optimization of minimizing the performance interference. For example, when two slots are free simultaneously and the first job in the wait queue has the acceptable interference degree with the two nodes, in this case, one needs to determine which slot will be used to serve the job. However current works do not highlight this
issue and, in fact, it needs to make a decision with a global optimization.

As for the performance of the MapReduce in the heterogeneous environment, [32] presents a LATE method to improve the performance of MapReduce applications through speculative execution. Reference [33] proposes a method for optimizing the speculative execution by considering the computing power to optimize the method for estimating the remaining time. Reference [34] proposes a scheduling method especially for the heterogeneous environment. This algorithm according to the historical execution progress of the task dynamically estimates the execution time to determine whether to start a backup task for the task with low progress rate. However, the above literature does not consider the factor of the performance interference among virtualized computing resource on the problem of identifying the stragglers when estimating the remaining time. Besides, when assigning the backup task to the slot, current works do not consider the performance interference which may cause the future straggler again. Besides this, current work only waits for the straggler without a prediction in order to make the backup decision early. Then, the effectiveness of the method may be affected also.

For the limitations of the above works, this paper proposes an optimized speculative execution framework for MapReduce jobs on the virtualized computing resources. The framework considers the interference. Then, an interference prediction is employed, and, according to the prediction, the framework will compute the remaining time of the task to predict the stragglers and assign the backup task to an appropriate node.

3. Framework Overview

Figure 1 shows the optimized speculative execution framework for MapReduce jobs. This framework is mainly for the MapReduce applications running in a virtual cluster. In the cluster, there are a set of physical servers. We imagine that each of the physical servers has the same virtualized environment. Each physical server can allocate its resource to multiple virtual machines. The virtual machine can host the application. The virtual cluster serves the Hadoop framework. The Hadoop framework has one master node and multiple slave nodes. The master node is deployed on a dedicated physical host. For each of the slave nodes, it will be deployed on a VM. In the master node, there are 4 major components: *Straggler Identification Module*, *Backup Module*, *Heart Beat Receiver*, and *Performance Interference Modeling & Prediction*. *Straggler Identification Module* is to compute the remaining time of the task in order to identify the straggler; *Backup Module* is to assign the straggler tasks to the slots; *Heart Beat Receiver* is to collect the running states of the servers and the tasks by receiving the heart beat information from the slave nodes; *Performance Interference Modeling & Prediction* is to train or retrain the performance interference model for predicting.

In the Sections 4 and 5, the major components in our framework will be discussed.

4. Methods for Predicting Performance Interference

4.1. Modeling the Performance Interference. In a virtual cluster, the application *app* deployed on a virtual machine (VM) will consume the resource of this VM. Due to the contention of the limited shared resource, the resource usage of the VMs consolidated on the same physical host may affect others' access to the shared resource. Then, the performance degradation of the applications on the VMs may be caused. To mitigate such degradation, one of the important issues is to predict the extent to which the application's performance is affected by the contention of the shared resource. By this, when the predicted result shows a bad degradation, we can place this application on the other VM to mitigate the performance degradation. In the following, for simplicity, the “foreground VM” is used to signify the VM which serves the application *app* to be deployed while the other VMs consolidated with the “foreground VM” are called the “background VMs.”

As stated above, the contention of the shared resource may cause the performance interference of the VMs to be consolidated on the same physical server. Then, the resource usage pattern of the “background VM” may affect the performance of the “foreground VM.” With the difference of the resource usage of the background VM, the performance of the foreground one will be different. That is to say, the extent to which the foreground VM's performance is affected by the background one is different. Then, the term “performance interference degree” is used for signifying this extent.

**Definition 1** (performance interference degree). We use (1) to show the performance interference degree.

\[
\text{PID (FW@BW)} = \frac{\text{Perf (FW@BW)} - \text{Perf (FW@Idle)}}{\text{Perf (FW@Idle)}},
\]

where we use system-level workloads to reflect the resource usage pattern of a VM. The system-level workloads considered in this paper are shown in Table 1. FW and BW are the workloads of the foreground and background VMs, respectively. The performance of the application on FW may include response time and throughput. We use Perf(FW@BW) to signify such performance when the background VM's workload is BW. Here, Idle is especially for the background VM when no application has been deployed on it.

Since the contention of the shared resource can cause the performance degradation, the interference degree of the foreground VM will have a relation with the resource usage pattern of the background VM. We also do some experiments to show this relation as Tables 2 and 3 show.

Tables 2 and 3 show that, with the background VM serving different types of applications, the response time of the foreground one is different. Here, when the background VM serves different types of applications, it means that the resource usage pattern of the background VM is different which also causes the difference of the performance of the
Table 1: System-level workload considered in this paper.

<table>
<thead>
<tr>
<th>System-level workload</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpuutil</td>
<td>Average CPU utilization</td>
</tr>
<tr>
<td>memutil</td>
<td>Average memory utilization</td>
</tr>
<tr>
<td>rps</td>
<td>Average number of read operations per second</td>
</tr>
<tr>
<td>wps</td>
<td>Average number of write operations per second</td>
</tr>
<tr>
<td>await</td>
<td>Average waiting time of the I/O operations</td>
</tr>
<tr>
<td>svctm</td>
<td>Average time spent for the request in the disk device</td>
</tr>
</tbody>
</table>

foreground VM. For example, with the background VM's system-level workloads varying, the application *cat* has different response time. Then, we use the system-level workloads to reflect the interference degree as (2) shows.

\[
\text{PID (FW@BW)} = a_0 + a_1 \times \text{cpuutil}_{BW} + a_2 \\
\times \text{memutil}_{BW} + a_3 \times \text{rps}_{BW} + a_4 \\
\times \text{wps}_{BW} + a_5 \times \text{await}_{BW} + a_6 \\
\times \text{svctm}_{BW},
\]

(2)

where \(a_0, a_1, a_2, a_3, a_4, a_5, \) and \(a_6\) are coefficients.

By using (2), the interference degree can be known if the coefficients are known. Then, we need to estimate the coefficients. Imagine that the estimated coefficients are \(a_0', a_1', a_2', a_3', a_4', a_5', \) and \(a_6'.\) Then, according to (2), the model for estimating the performance interference degree can be as follows:

\[
\text{PID (FW@BW)} = a_0' + a_1' \times \text{cpuutil}_{BW} + a_2' \\
\times \text{memutil}_{BW} + a_3' \times \text{rps}_{BW} + a_4' \\
\times \text{wps}_{BW} + a_5' \times \text{await}_{BW} + a_6' \\
\times \text{svctm}_{BW}.
\]

(3)

Then, when the background VM's workloads are fed into the above equation, we can estimate the performance interference degree. To estimate the coefficients, we need to compute the error between the predicted interference degree and the actual one according to the observed data record.

Then, the problem of finding the combination of the coefficients can be mapped to a problem according to the set of observed data \(\{(\text{pid}_1, \text{cpuutil}_1, \text{memutil}_1, \text{rps}_1, \text{wps}_1, \text{await}_1, \text{svctm}_1), \ldots, (\text{pid}_n, \text{cpuutil}_n, \text{memutil}_n, \text{rps}_n, \text{wps}_n, \text{svctm}_n)\}\), where \(\text{pid}_i, \text{cpuutil}_i, \text{memutil}_i, \text{rps}_i, \text{wps}_i, \text{await}_i, \text{svctm}_i\) are coefficients.
The particle problem, the first task is to define the particle. For this problem, we use a swarm particle algorithm. The error between the predicted interference degree and the optimal combination of the coefficients, in order to make \( i \) particle swarmsgnified as \( m \) in the direction which can be seen in

\[
\text{Error} = \sum_{i=1}^{n} \left[ \text{pid}_i - \left( a'_i + a'_2 \times \text{cpuutil}_i + a'_3 \times \text{memutil}_i \right) + a'_4 \times \text{rps}_i + a'_5 \times \text{wps}_i + a'_6 \times \text{await}_i + a'_7 \times \text{svctm}_i \right]^2.
\]  

The above problem can be seen as a problem of finding the optimal combination of the coefficients, in order to make the error between the predicted interference degree and the actual one the minimum. In this paper, for solving the problem efficiently, we use a swarm particle algorithm.

When using swarm particle algorithm to solve such problem, the first task is to define the particle. For this problem, the particle \( i \) in the swarm can be defined as \( p_i = [a_{q_0}, a_{q_1}, a_{q_2}, a_{q_3}, \ldots, a_{q_6}] \). Here, \( a_{q_j} \) signifies the location of the particle \( i \) in the direction \( j \). The number of particles in a swarm is signified as \( n \). The particle \( p_i \) will update its location in the direction \( j \) with a speed \( v_{ji} \). The particle will compute the speed according to the best location \( p_{Best} \) the particle is experiencing and the best location \( q_{Best} \) the swarm is experiencing. The best location means the location which is the closest one to the optimal solution which usually is expressed as the fitness function. As for our problem, the fitness function should evaluate how the swarm is close to the optimal solution. Then, according to formula (4), the fitness function of a swarm can be defined as follows:

\[
\text{fitness} (p_i) = \sum_{i=1}^{n} \left[ \text{pid}_i - \left( a_{q_0} + a_{q_1} \times \text{cpuutil}_i + a_{q_2} \times \text{memutil}_i \right) + a_{q_3} \times \text{rps}_i + a_{q_4} \times \text{wps}_i + a_{q_5} \times \text{await}_i + a_{q_6} \times \text{svctm}_i \right]^2.
\]

Then, we can use formula (6) to update the speed of the particle \( p_i \) in the direction \( j \) and compute the location of the particle in the same direction as formula (7).

\[
\begin{align*}
\text{where} \quad v_{ji}(t+1) & = \omega \times v_{ji}(t) + c_1 \times r_1 \left( p_{Best_{ji}} - x_{ji}(t) \right) \\
& \quad + c_2 \times r_2 \left( q_{Best_{ji}} - x_{ji}(t) \right) \\
& \quad + x_{ji}(t-1) + v_{ji}(t-1),
\end{align*}
\]

Then, the PSO algorithm can find the optimal combination of the coefficients of each attribute. Algorithm 1 presents the detailed algorithm.

The method which uses regression model for estimating the performance interference degree can work well when there are historical data for training the coefficients. However, as for the problem of MapReduce job scheduling, such historical data may not always be available. This is because the new arriving jobs may not have the historical data about the running status together with the consolidated VM in the same physical host. Then, in this case the historical data for training may not be available. For this situation, we will discuss the corresponding method in the following.
4.2. Inferring the Performance Interference Degree. For two applications, if their resource usage patterns are similar, with the same background VM, their extents of the performance degradation may be similar. Then, when one of the applications is new and little historical data can be used for training its performance interference degree model, we can predict its performance interference by looking at another one's model. Based on this idea, we will discuss our method in the following.

Imagine that the performance interference degree models can be kept and stored. Then, all the models can be a set \( H = \{\text{PID}(\text{FW}_1@), \text{PID}(\text{FW}_2@), \ldots, \text{PID}(\text{FW}_n@)\} \). Here, \( \text{FW}_i \) of each item \( \text{PID}(\text{FW}_i@) \) in \( H \) is called the workload pattern. Then, if we do not have enough historical data for training application \( A \)’s performance interference model, we can use an available and appropriate model in \( H \) for prediction.

Let \( \text{wp} \) be the workload pattern of the virtual machine \( \text{vm} \). To find an appropriate equation in \( H \) is to find the equation whose workload pattern is the most similar to \( \text{wp} \).

Then, in the following, we will show how to compute the similarity degree.

For comparing the similarities, we will use an Euclidean distance. For two VMs \( \text{vm}_i \) and \( \text{vm}_j \), the similarity degree between their workload patterns can be computed as follows:

\[
d(\text{wp}_i, \text{wp}_j) = \left( \frac{\text{cpuutil}_{\text{vm}_i} - \text{cpuutil}_{\text{vm}_j}}{\text{cpuutil}_{\text{vm}_i} \times \text{cpuutil}_{\text{vm}_j}} \right)^2 + \left( \frac{\text{memutil}_{\text{vm}_i} - \text{memutil}_{\text{vm}_j}}{\text{memutil}_{\text{vm}_i} \times \text{memutil}_{\text{vm}_j}} \right)^2 + \left( \frac{\text{wps}_{\text{vm}_i} - \text{wps}_{\text{vm}_j}}{\text{wps}_{\text{vm}_i} \times \text{wps}_{\text{vm}_j}} \right)^2 + \left( \frac{\text{await}_{\text{vm}_i} - \text{await}_{\text{vm}_j}}{\text{await}_{\text{vm}_i} \times \text{await}_{\text{vm}_j}} \right)^2 + \left( \frac{\text{svctm}_{\text{vm}_i} - \text{svctm}_{\text{vm}_j}}{\text{svctm}_{\text{vm}_i} \times \text{svctm}_{\text{vm}_j}} \right)^2 -1/2.
\]

Then, we can use (9) to find the workload patterns which are similar to the workload pattern of the VM to be predicted. Then, for a workload pattern \( \text{wp} \), by comparing the similarity degrees, we may find multiple workload patterns satisfying the predefined threshold requirement. Then, we can use the following equation to generate a combined equation. By using such combined equation, we can estimate the performance interference degree for the VM which has no historical data for training the model.

\[
\text{PID}(\text{FW}@\text{BW}) = \sum_i \left( \frac{d_i}{\sum_j d_j} \right) \times \text{PID}(\text{FW}_i@\text{BW}) \tag{10}
\]

where, for the VM which is used to predict the performance, \( \text{FW} \) is used for signifying its workload. Imagine the workload patterns satisfying the threshold requirements form the set \( R \). \( \text{PID}(\text{FW}_i@\text{BW}) \) is the interference model corresponding to the \( i \)th workload pattern in \( R \). \( d_i \) is the similarity degree between \( \text{FW} \) and \( \text{FW}_i \).

Then, by using the above methods, the performance interference model can be generated. By using the model, we can estimate the performance interference degree of an application. For a MapReduce job, it may contain a set of tasks. The resource usage patterns of these tasks are always similar [36]. And there are also many research works for predicting the resource demand of the MapReduce jobs. Then, using this information, the performance interference degree between the tasks to be assigned (no matter whether the corresponding job is newly submitted or runs for a while) and the VMs on the candidate physical host can be predicted.

5. Methods for Identifying Straggler and Backing-Up in Virtualized Environment

In our framework, the task trackers will send the heart beat information which includes the resource status of the VMs. Taking the task profile, the status of VMs, and the physical host as inputs, the module of Performance Interference Modeling & Prediction will return a value to evaluate the interference. Then, in every interval, the Straggler Identification Module will predict the remaining time of each running task in the next time interval according to the heart beat information from the slave node and the performance interference degree provided by the Performance Interference Modeling & Prediction. The backup module will back up a new task for the straggler by assigning a new slot to it.
In the speculative execution, the task which will finish farthest into the future will be backed up since the backed up task will have a greater opportunity to overtake the original one and reduce the overall response time of the job. Then, the core of identifying a straggler is to estimate whether the task has a bad progress rate; that is to say, compared with other tasks in a job, it has a longer remaining time to be finished. Then, in the following, we will introduce how to estimate the remaining time of the task in order to identify the stragglers.

Imagine we have a job \( j = \{ t_1, t_2, \ldots, t_n \} \) which contains a set of tasks. Then, we will introduce how to find the straggler tasks in the job. Imagine that the number of the allocated map slots for this job is \( s_m \) and the number of the allocated reduce slots for this job is \( s_r \). Imagine that the number of the map tasks in this job to be executed is \( n_m \) and the number of the allocated reduce slots for this job to be executed is \( n_r \). The overall remaining time of the job is a sum of the remaining time of the map phase and the reduce phase. The remaining time of either the map phase or the reduce phase depends on the slowest task. Then, the remaining time of \( t_i \) can be computed as (5).

According to (5), \( t_{i,m}^{\text{predict}} \) is the predicted completion time of the current running map task \( i \) which can be computed as (11), \( t_{i,r}^{\text{predict}} \) is the predicted completion time of the current running reduce task \( i \) which can be computed as (12), \( t_{i,m}^m \) is the execution time of map task \( i \), \( t_{i,r}^r \) is the execution time of reduce task \( i \), \( t_{\text{max}}^m \) and \( t_{\text{avg}}^m \) are the maximum and average completion time, respectively, of all the map tasks which have been executed completely, and \( t_{\text{max}}^r \) and \( t_{\text{avg}}^r \) are the maximum and average completion time, respectively, of all the reduce tasks which have been executed completely.

\[
\begin{align*}
  t_{i,m}^{\text{predict}} &= t_{i,m}^m \times \frac{\text{PID}_{\text{predict}}^{\text{slot}(i)}}{\text{PID}_{\text{avg}}^{\text{slot}(i)}} \\
  t_{i,r}^{\text{predict}} &= t_{i,r}^r \times \frac{\text{PID}_{\text{predict}}^{\text{slot}(i)}}{\text{PID}_{\text{avg}}^{\text{slot}(i)}}
\end{align*}
\]

where \( \text{slot}(i) \) is the function to return the slot where the task \( i \) is deployed on, \( \text{PID}_{\text{predict}}^{\text{slot}(i)} \) is the predicted performance interference degree among the slot \( \text{slot}(i) \) and the other slots consolidated on the same physical server in the next time interval, and \( \text{PID}_{\text{avg}}^{\text{slot}(i)} \) is the average performance interference degree among the slot \( \text{slot}(i) \) and the other slots consolidated on the same physical server in the last interval from the beginning of the execution to the current time.

The remaining time of the job can be predicted. If there exists a running task whose predicted completion time makes the remaining time bigger than the required one, this task will be the straggler.

Then, after identifying the stragglers, a backup task for the stragglers needs to be initiated by assigning a slot for this task. Since, from every time interval, the Straggler Identification Module will predict the stragglers in the next time interval, there may be a set of straggler tasks to be backed up. This problem can be seen as a problem of scheduling this set of tasks in a virtualized computing environment. As the performance interference is an important factor which may affect the execution of the tasks, when scheduling the task to a slot with high time interference degree with others, the task may become a new straggler in the future again which may result in the bad performance of the job. Then, when dealing with the problem of how to back up the stragglers, the performance interference degree needs to be considered also. Previous works [37] schedule the tasks to the slot, if the predicted interference degree is not higher than a predefined threshold \( T \); otherwise, the task will wait for the available node with the required interference degree or will be assigned to a slot when the task is waiting for a long time. In these works, the scheduler optimizes the assignment with the consideration of only one task or only one slot while it is hard to achieve the global optimization of minimizing the performance interference. For example, when two slots are free simultaneously and the first task in the wait queue has the acceptable interference degree with the two nodes, which slot is used to place the task in will affect the following assigning...
Input: the set $SL$ of slots to be free in the next interval; the queue $Q$ of tasks to be assigned.
Output: assignment plan $AP$.

Begin
(1) While $Q$ is not empty do
(2) Begin
(3) $\text{min} = 10000$;
(4) For each slot in $SL$ do
(5) Begin
(6) If $\text{slot}\_\text{capacity} >= Q\_\text{element}[i]\_\text{demand}$ then
(7) $PID = \text{GetPID}(Q\_\text{element}[i], \text{slot}_j, \text{Background})$;
(8) If $\text{min} > PID$ then
(9) begin
(10) $\text{min} = PID$;
(11) $AP\_\text{candidate}[i] = \text{slot}_j$;
(12) end
(13) End
(14) If $\text{min} < \text{threshold}$ then
(15) $AP[i] = AP\_\text{candidate}[i]$;
(16) end
(17) Return $AP$;
End

Algorithm 2: Backing up the stragglers with a global optimization.

plan. That is to say, a decision with a global optimization needs to be made.

This paper presents a scheduling strategy with a global optimization as mentioned in Algorithm 2. In each interval, the backup module will collect the status of the tasks running in the slots and estimate which slots will be free in the next interval by computing the remaining time of the task. Then, in each interval, the backup module will assign a set of tasks to the set of free slots for the next interval with the global optimization of minimizing the performance interference degree of each task. Optimally finding the solution to the above problem is an NP-complete problem. Then, we propose a greedy algorithm for solving this problem with better efficiency. Firstly, the algorithm will place the task on the slot with least interference degree. Then, for the remaining slots to be free in the next interval, redo the first step until all the slots are assigned with a task.

6. Simulation Results

We evaluate our framework in a 24-node virtual cluster. The cluster has 6 physical servers; one is for the mast node. The configuration of each server is as follows: the memory is 4 G, disk amount is 250 G, and the version of CPU is i3. On each physical server, 4 virtual machines are deployed. Each VM is created using Xen hypervisor and has 4V CPU and 1 GB memory. We configured each virtual machine with 1 slot which can be a map slot or a reduce slot. In the whole virtual cluster, we allocate 16 map slots and 8 reduce slots.

We evaluate the framework using 10 MapReduce applications, seen in Table 4. These applications are widely used for evaluating the performance of MapReduce framework in the previous research works [21, 32, 38, 39]. To verify the effectiveness of our works, the experiments will be carried out for some comparisons between our scheduler and other main competitors which also consider the performance interference in the scheduling.

In this section, we evaluate whether our method is effective in estimating the interference degree. We will compare it with the model discussed in previous works [12] which uses a uniform model for evaluating all the applications. In our experiment, the predicted and actual performance interference degrees are considered. Figure 2 shows the prediction error for each type of jobs using different models.

From Figure 2, we can see that the current method led to an average of 29% error rate while our method can achieve the average rate of 15%. This is because our method trains the model with the consideration of no historical data about performance interference while the current method relies
on establishing a uniform model to evaluate all the types of applications which will sacrifice the prediction accuracy.

In the following part, the experiments will be done to show whether our method is effective in predicting the remaining time in every time interval.

From Figure 3, we can see that the current method led to an average of 20%. This is because our method considers the performance interference in the estimation of the remaining time while the current method in [32] only takes an average progress rate for the estimation.

In the following, the experiments will show the effectiveness of our method in speculative execution. The performance of the backup module is also affected by the data locality. Then, to emphasize the performance interference only, we conduct the experiment in an intranet environment where when accessing the data, it does not need to read the data remotely which minimizes the effect caused by the data locality as much as possible. We select the applications of Matrix and TeraGen which need no input and we also select the applications of TeraSort and Gzip which need to read data. We set the numbers of map tasks in the Matrix job, TeraGen job, TeraSort job, and Gzip job which are 15, 10, 10, and 5, respectively. Every 15 seconds, a batch of jobs which contains 3 Matrix jobs, 3 TeraGen jobs, 5 TeraSort jobs, and 2 Gzip jobs will be submitted in the virtual cluster. The average normalized completion time is used for evaluation. In our method, we model the relation between the performance interference degree and the background workload. Then, in the experiment, we will show the effectiveness of our scheduler under the different status of the background workload.

We will adjust the background workload in this way that we let different jobs run on the virtualized slave node in order to adjust the cpu, memory, and other system load to simulate the variations of the background workload. Figures 4 and 5 show the result when using different schedulers in the master node.

From Figures 4 and 5, when the workload of the background is heavy, for example, with the high CPU and memory utilization, all the applications suffer the performance degradation severely when using the FairScheduler [37] and CapacityScheduler [20]. Even under the situation with the light workload of the background, the speculative execution has the better performance than the FairScheduler and CapacityScheduler. The reason is that speculative execution can identify the stragglers and speed up the speed of the
application. Besides, our speculative execution outperforms the current speculative execution. This is because ours finds the stragglers by prediction while the current one finds them by waiting for the degradation. Besides, the backing-up module in our framework also considers the performance interference when assigning the slots which may reduce the future risk of the degradation caused by the performance interference. However, we also notice that when the background workload is light, the performance of the different schedulers is not too different. This is because, with the light background workload, the application suffers not too bad performance as a result of the interference among virtualized slave nodes. However, in reality, maintaining a light background workload is usually not an easy task especially with the consideration of the cost of the hardware and the system utilization.

7. Conclusions
This paper presents an optimized speculative execution framework for MapReduce jobs which aims to improve the performance of the jobs on the virtual cluster. Firstly, we analyze the factors related to the performance degradation in the virtual cluster and present a model for modeling how the factors affect the degradation. Secondly, we develop an algorithm that works with the performance interference prediction to identify the stragglers and assign the tasks.

In this work, when predicting the remaining time of the MapReduce job, only the performance interference factor is considered. In fact, there are other factors such as the fault ratio of the physical server which can also affect the accuracy of estimating the remaining time. Then, in the future works, we will optimize our method in predicting the remaining time of the MapReduce jobs.

Conflicts of Interest
The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments
This work was supported in part by the National Key Technology R&D Program of the Ministry of Science and Technology (2015BAH09F02 and 2015BAH47F03), National Natural Science Foundation of China (60903008 and 61073062), and the Fundamental Research Funds for the Central Universities (N130417002 and N130404011).

References


Submit your manuscripts at https://www.hindawi.com