Network and Cost Aware Scheduling Algorithm for Minimizing Recovery Time in Cloud Computing

Pham Phuoc Hung, Aymen Al-Saffar, Le Anh Tuan, Md. Motaharul Islam, Mohammad Aazam, Eui-Nam Huh
Department of Computer Engineering, Kyung Hee University

Abstract
Nowadays thousands of servers in a cloud datacenter coordinate tasks to provide more reliable and highly available cloud computing services, especially in multi-task processing. Therefore, we need mechanisms to prepare for failure of computing nodes. So far, a number of research studies have been carried out, trying to eliminate these problems, yet little have been found efficient. In this paper, we present a multi-task scheduling algorithm that makes recovery from a saved state. The approach can improve execution time including recovery time in case of failure while overhead in the case of no failure was a little in typical scenarios.

1. Introduction
Recently the rapid growth of quantity and the scale of business work flows or application that companies or commercial organization have to process everyday creates a huge pressure for individual cloud datacenters, especially some huge tasks require a lot of computing resources. Therefore, thousands of servers in the datacenter coordinates tasks in order to deliver highly available cloud computing services. In order to make these services more reliable and to improve availability, we need a scheme to deal with failure of computing nodes. While the probability of seeing any such failure in the lifetime of a server can be somewhat small, these numbers get magnified cross all devices hosted in a datacenter [1]. This problem can lead to degradation in performance to end-users because of service unavailability and can cause losses in business, both in immediate revenue [2] as well as long term reputation.

There have been a lot of research works discussing how to reduce recovery time in case of failure. However, there is not attention paid to the cost that it would take to use resources from cloud providers, particularly the access to cloud is too expensive. Hence, in this paper, we present a network and cost aware multi-task scheduling algorithm that cut down recovery time from a saved state on parallel processing in case of failure while overhead in the case of no failure is a little in typical scenarios.

We divide our paper into following sections: section 2 gives literature review on related study; section 3 presents a motivating scenario, thus reasons the necessity of our approach; section 4 specifies the proposed algorithms; the last section concludes the paper.

2. Related Study
There have been numerous studies which attempt to solve task scheduling problems. In [3], the authors propose a task scheduling approach for assigning processors to task graph templates prepared beforehand. The limitation of this method is not to consider network contention or failure of processors. Zhang in [4] introduces a solution to recover from failure according to check-pointing in stream processing system. With this way, when the system has failure, it finds the closest ancestor node which is not impacted by failure to resume the results from the saved state in that node. In [5], the proposers present a...
scheme to reduce the recovery time in case of failure but they do not speculate in the cost and reliability of suppliers’ services. Thereby, in this paper, we try to solve the above lacks by introduce a method that can reduce recovery time consumption while considering bandwidth, cost and reliability of cloud providers.

3. Motivating Scenario

The following scenario reflects the benefits behind the proposal. In a cloud datacenter, there are a lot of physical machine servers. Each of a single physical sever may not have enough capacity to host all the VMs for the users’ requirement to solve some huge tasks. Even if there is enough capacity of the machine server, users may not want to have all the VMs hosted in the machine in order to guard against complete failures in the machine having a service outage. Therefore, the tasks should be divided into subtasks executed by VMs in other different physical machines. Each of those VMs can execute tasks independently. Communication between VMs in a machine can finish instantaneously. And one network interface is shared among the VMs. Because the communication link among VMs in the machine has very high speed, existing schedulers try to utilize this high speed link. As a result, many dependent tasks are assigned to these VMs.

In case of fault, dependent tasks tend to be destroyed at a time. Especially, the worst case occurs when all tasks in critical path are assigned to VMs in a machine, during the time the last tasks are being executed, a failure happens, all tasks have to be executed again for recovery. Hence, it is recommended that the last tasks should be moved to VMs in other machines in order to improve recovery time. On the contrary, the communication overhead increases if no failure happens due to the extra communication between VMs. Moreover, if we distribute too many tasks, there is too much overhead. With that in mind, our paper tries to reduce that overhead as much as possible to deal with the following issues:
- Minimizing the total execution time in the worst case.
- Scheduling tasks to minimize the execution time before failure and recovery time.

4. Proposed Method

In this section, we are going to propose a method to solve the above problems. We first define the terms used in this paper. Then, we formulate the problem.

4.1 Definition

Here we define several terms used in this part. A task graph is presented by a Directed Acyclic Graphs DAG \( G = (V,E) \) where the set of vertices \( V = \{v_1,v_2, ..., v_l\} \) represents the set of parallel subtasks, and the directed edge \( e_i = (v_i,v_j) \in E \) presents the communication between subtasks \( v_i \) and \( v_j \). Each subtask \( i \) consists of workload \( w_i \) of subtask \( i \), the set of preceding subtasks \( prec(i) \), the set of successive subtasks \( succ(i) \) of task \( i \), the amount of data \( (d_{ij}) \in succ(i) \) which is transferred from task \( i \) to task \( j \), the amount of data \( (cd_{ij}) \in succ(i) \) stored at the VM \( k \), used for subtask \( i \) and \( m \) is the number of VMs in data center. The left of Figure 1 shows an example of task graph consisting 7 subtasks.

![Figure 1. A sample DAG and a processor graph](image)

A processor graph is a graph that describes the topology of a network between vertexes which are VMs and edges are communication paths between VMs. The right of figure 1 illustrates a processor graph showing that 3 VMs are connected by 2 switches. Each VM \( i \) controls processing rate \( \rho_i \), bandwidth \( bw \), to communicate with other VMs, cost \( cc \); to use computing resources on a unit of time and cost \( dc \); for a unit of outgoing data (e.g MB, GB).

And task scheduling is to assign a VM node to a task node in order to minimize total task execution time. Its input is a task graph and process graph. Then the output is a schedule which is an assignment of a VM to each task node.

4.2 Problem formulation

In this part, we make some following assumptions for proposed method. A state contains the data will be transferred to the VMs that execute the child nodes of the completed task nodes. Only output data from each task node is stored as a state. If a VM failure goes on, a saved state is found in the ancestor task node. We assume that only a single stop failure of a physical machine happens. When it occurs, all VMs in the machine stop execution coincidently.

Let \( sl(s) \) be the length of a schedule \( s \). Entire schedule is denoted \( rs(s,v) \) with schedule \( s \) is executed and \( v \) is a fault that happens at node \( v \). Our
intent is to try to find and move the best last tasks in the critical path over multiple machines to minimize the execution time before failure and recovery time and avoid the worst case. That means finding the schedule $s$ such that: \( \text{Minimize } \max_{v \in V} sl(rs(s, v)) \)

If a subtask $i$ is executed on a VM $j$ then the cost of the for using the VM $j$ as following:

\[
\text{Cloud\_cost}(i,j) = \text{computation cost}(i, j) + \text{communication cost}(i, j)
\]

where computation cost $p_{Cij} = \frac{w_i \cdot r_{Cji}}{p_j}$ and communication cost

\[
t_{Cij} = \sum_{l \in \text{Cloud\_link}\_j} t_{Cij} = \sum_{l \in \text{Cloud\_link}\_j} c_{dl} \cdot d_{cl}
\]

It is also predicted the Earliest Start Time of task $i$

\[
EST_i = \max_{p \in \text{provisioned}} [AFT(j)]
\]

and its earliest execution time:

\[
EET(i, j) = \max\{\text{avail}(j), \max_{l \in \text{link}_j} (a_{lh})\}
\]

Then we can calculate the Earliest Finish Time:

\[
EFT(i, j) = EET(i, j) + w_i / p_j
\]

At last, the Actual Finish Time is computed as follow:

\[
AFT(i, j) = EFT(i, bestnode)
\]

where $bestnode$ is the most appropriate processing node with maximize of Cost–on–Makespan Ratio and probability of reliability (number of success executions/number of executions according to the machine's execution history ) as below:

\[
CMR = \frac{\min_{\text{Cloud\_cost}(i,j)}}{\text{Cloud\_cost}(i,j)} \cdot \frac{\min_{\text{EFT}(i,j)}}{\text{EFT}(i,j)} \cdot Pr_{\text{reliability}}(j)
\]

The following part will present an algorithm to schedule the tasks to VMs as algorithm 1 and algorithm 2 will find the best recovery scheduling based on network and cost as well as reliability of cloud services.

\section*{Algorithm 1}

\begin{enumerate}
  \item Input: Task graph $G=(V, E)$, processor graph $H=(P, R)$
  \item Sort nodes $n \in V$ into list $L$ according to priority.
  \item For each $n \in L$
    \begin{enumerate}
      \item Find best processor $\rho \in P$ which has $argmax_{S \subseteq S} [CMR(i,j)]$
      \item Assign $n$ on $\rho$
    \end{enumerate}
\end{enumerate}

\section*{Algorithm 2}

\begin{enumerate}
  \item Input: Task graph $G=(V, E)$, processor graph $H=(P, R)$
  \item For ($nmove = 1$: $nmove <$ number of tasks in critical path)
    \begin{enumerate}
      \item $S_1$ = schedule generated by Algorithm 1.
    \end{enumerate}
  \item For ($failtask = nmove + 1$ to number of all tasks)
    \begin{enumerate}
      \item Assuming failure happens at failtask
      \item Find set $T$ of all tasks executed after recovery if task failtask fail
      \item $S_2$ = Schedule is generated by Algorithm 1 with input is $T$ and available processors \( S = S \cup (S_1 + S_2) \)
    \end{enumerate}
  \item End For
  \item Criticalpath[++] = $\max_{v \in V} sl(S)$
  \item End For
  \item $\text{Min sl(Criticalpath)}$
\end{enumerate}

Comparing our method with existing scheduling mentioned in Related Study, we see that our proposal has more advantages. It improves execution time including recovery time in case of failure. Nevertheless, we have to sustain a little overhead if no failure.

\section{5. Conclusion}

In this paper, we have introduced a task scheduling method considering network contention, cloud cost and reliability of cloud services to reduce recovery time in case of failure in order to improve the reliability and availability of the cloud service while overhead in the case of no failure was a little in typical scenarios.

In the future, we will conduct the simulation and extend our approach in a variety of circumstances to achieve higher reliability and better performance.

\section*{Acknowledgment}

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2012-0006418). The corresponding author is Eui-Nam Huh.

\section*{Reference}