Auto-Scaling Method in Hybrid Cloud for Scientific Applications

Younsun Ahn Jieun Choi Sol Jeong Yoonhee Kim
Dept. of Computer Science
Sookmyung Women's University
Seoul, Korea
ahnysun@sookmyung.ac.kr, jechoi1205@gmail.com, {jeongsol, yulan}@sookmyung.ac.kr

***Abstract*— Scientists can ease to conduct large-scale scientific computational experiments over cloud environment according to an appearance of Science Clouds. Cloud computing enables applications to apply on-demand and scalable resources dynamically. It is necessary for Many Task Computing (MTC) to offer high performance resources in a long phase and certificate stable executions of applications even dramatic changes of vital status of physical resources. Auto-scaling on virtual machines provides integrated and efficient utilization of cloud resources. VM Auto-scaling schemes have been actively studied as effective resource management in order to utilize large-scale data center in a good shape. However, most of the existing auto-scaling methods just simply support CPU utilization and data transfer latency. It is needed to consider execution deadline or characteristics of an application. We propose an auto-scaling method, guaranteeing the execution of an application within deadline. It can handle two types of job patterns; Bag-of-Tasks jobs or workflow jobs. We simulate a variable index computation application in hybrid cloud environment. The results of the simulation show the method can dynamically allocate resources considering deadline.**

Keywords—auto-scaling; hybrid cloud computing; science cloud; Bag-of-Tasks; workflow

I. INTRODUCTION

Cloud computing enables applications to apply on-demand and scalable resources dynamically. Scientists can ease to conduct large-scale scientific computational experiments over cloud environment according to an appearance of Science Clouds. It is necessary for Many Task Computing (MTC) to offer high performance resources in a long phase and certificate stable executions of applications. Cloud computing offers endless resources by a virtualization technology. Auto-scaling can automatically change the number of Virtual Machines (VM) during execution of an application. Auto-scaling can provide efficient integrative utilization of cloud resources.

Our previous paper [1] proposes an auto-scaling method that dynamically allocates resources satisfying a deadline in a hybrid cloud computing environment. However, proposed auto-scaling algorithm needs to be extended in order to support various patterns of job execution, which are Bag-of-Tasks [2] and workflows.

This paper proposes an extended version of the auto-scaling method, reflected in patterns of jobs and the requirements of an application in hybrid clouds. Chiefly, it dynamically allocates resources depending on the two patterns of jobs (Bag-of-Tasks [2] and workflow) in hybrid clouds. We developed an auto-scaling algorithm and applied it to specialized computation and data analysis application such as variable index computation in hybrid cloud environment as a proof-of-concept.

The rest of the paper is organized with the sections as follows; we introduce an overview of related works in Section 2. Section 3 explains auto-scaling algorithm, and Section 4 contains contents about a scenario of using auto-scaling algorithm and experiment results are discussed. Finally, we conclude the paper and discuss future work in final section.

II. RELATED WORK

Cloud computing provides on-demand and scalable resources by virtualization technology. Auto-scaling topics are currently being discussed and studied as useful resource management. Auto-scaling issues exist in two aspects of consideration. One side use rule-based auto-scaling method. "Auto-scaling" of AWS [3], Paraleap [4] for Windows Azure [5], and Scalr [6] is the example of the rule-based auto-scaling method. Rule-based auto-scaling method elastically increases or decreases resources to meet user-defined metrics, but it is not enough to allocate resources actually needed and it have issues with violation of deadline and cost. On the other hand, [7], [8], [9], [10], [11] are the studies of auto-scaling which consider deadline or cost for resource usage.

[7] proposes an auto-scaling method using Bag-of-Tasks [2] jobs in order to minimize resource usage. It is consider horizontal scaling and vertical scaling. Horizontal scaling adds or removes the number of VM instances and vertical scaling expands or reduces the size of a VM. However, [7] lack of considering resource requirements of dynamic workloads during execution of an application. [8] respects to finish jobs within deadlines at minimum financial cost using their auto scaling method for a workflow and fit deadline on highest performance resource. [9] proposes an algorithm to minimize execution costs while meeting deadline. However, they perform their auto-scaling method in simple environment.

[10], [11] perform their auto-scaling methods using workflow application on Grids. [10] proposes an algorithm that

This research was supported by the MSIP(Ministry of Science, ICT&Future Planning), Korea, under the C-ITRC(Convergence Information Technology Research Center) support program (NIPA-2014-H0401-14-1022) supervised by the NIPA(National IT Industry Promotion Agency)

reduces cost for resource usage within deadline. [11] proposes an economical scheduling algorithm for dependent tasks. We propose a workflow scheduling algorithm referring to Reference [10]'s and [11]'s scheduling algorithms.

In this paper, we propose an extended version of an auto-scaling method based on our previous paper [1] that performs dynamically allocating VM considering the types of jobs from Bag-of-Tasks [2] as well as the workflow and the characteristics of an application.

III. AUTO-SCALING ALGORITHM

A. Assumptions

We use two kind of types of jobs, Bag-of-Tasks [2] and workflow. Jobs in Bag-of-Tasks [2] type can be scheduled on resources separately from each other while jobs in workflow can be performed in order of dependency pattern. In the workflow scheduling algorithm, we use the term 'task' for a job. Auto-scaling method can find delay and deadline violation to comparing actual start time and estimated start time of running jobs. Moreover, monitoring perform every regular term.

B. Algorithms

This paper extends the version of [1]'s algorithm in order to enable the auto-scaling method to schedule jobs in Bag-of-Tasks [2] and workflow. Assumptions and notations for the algorithms are explained in detail at the reference [1]. Algorithm 1, Run-time Scaling is extended based on [1]. We describe algorithm 3, Workflow Scheduling algorithm to deal with tasks in workflow.

In the reference [1], Run-time Scaling Algorithm has two polices and depicts our overall auto-scaling method. Additionally, we extend three policies by adding workflow scheduling. Jobs are scheduled with applying one of the three policies such as Performance-oriented Scheduling (line 5), cost-oriented Scheduling (line 8) and Workflow Scheduling (line 11) of SLA (Service Level Agreements). Jobs in Bag-of-Tasks [2] are sorted as descending order based on their execution time, while tasks in workflow are sorted as sequential order.

Algorithm 1 – Run-time Scaling

```
Input – An application,
        SLA={a policy P, a deadline D [, minimum performance
        requirement minPM ]}
Output – Scaling decision S = { toStartUp, toShutDown }
         Scheduling decision S = {jobs → VMs}

1:  SCALING ← TRUE;
2:  while (true)
3:      if SCALING is TRUE
4:          switch P
5:          case Performance:
6:              Sort waiting jobs in decreasing order of execution length;
7:                  S ← PerformanceOrientedScheduling(
                            sortedJobs, D, minPM);
8:          case Cost:
9:              Sort waiting jobs in decreasing order of execution length;
10:                 S ← CostOrientedScheduling(sortedJobs, D);
11:         case Workflow:
              Sort waiting jobs in sequential order;
12:                 S ← WorkflowScheduling(sortedJobs, D);
13:     each vm where status is running
14:     if no running/waiting jobs on vm then
15:         destroy the vm
16:     send scaling decisions to DRMS
17:     send scheduling decisions to JES
18:     waitForNextInterval();
19:     SCALING ← SLAMonitoring(runningJobs, D);
```

Algorithm2, Performance-oriented Scheduling is suitable for jobs in Bag-of-Tasks [2]. The algorithm basically accompanies cost-effective scheduling and plans jobs to meet deadline using minimum performance requirement. For more information of Performance-oriented Scheduling algorithm, it can be found in the reference [1].

Algorithm 2 – Performance-oriented Scheduling

```
Input –Sorted jobs of the application, deadline D, minimum performance
       requirement minPM
Output –Scheduling decision S = {jobs → VMs}, list toStartUp.for the
        VMs to be newly created

1: All jobs are scheduled
2: If There is no running private VM, find a private vm
   ($PM_{vm}$ >= minPM) on which job can start the most quickly within the D;
3: If There is running private VM, schedule job to vm;
4: Continue with the next job;
5: If There is no running public VM, find a public vm
   ($PM_{vm}$ >= minPM) on which job can start the most quickly within the D;
6: If There is running public VM, Choose Cheaper VM either
   running VM or VM of ITj type
7:     Schedule job to chosen vm;
8: Continue with the next job;
```

Algorithm 3 defines our workflow scheduling mechanism. Proposed Workflow Scheduling algorithm is based on a PCH algorithm [11] and followed the performance-oriented policy. We could get tasks on a critical path and group of tasks using PCH algorithm [11] and then each group can be scheduled. The total execution time of critical path is calculated. Tasks on a critical path are scheduled on a same resource, which can execute all tasks in the critical path in order to reduce communication overhead. Every time we try to schedule tasks, it is a rule to choose private cloud resource prior to public cloud. Each task considers estimated finish time of a parent task and calculates earliest start time. When tasks which are not on a critical path are scheduled to VMs, they check a parent task's estimated finish time.

IV. SCENARIO AND EXPERIMENTS

We simulated variable index computation application in Bag-of-Tasks [2] with CloudSim [12].

We simulated variable index computation application in

Bag-of-Tasks [2]. Our simulation is based on performance-oriented policy. Parameters determine the number of jobs. Execution time of the application takes relatively a long time because data size is about 16 millions.

We use standard values extracted from the observation data of SuperWASP project [13]. Resource specifications used in our experiment are equal with [1]'s. Two of the four private clouds have 1000 MIPS and the others have 2000 MIPS. Public clouds have 600 MIPS or 2400 MIPS.

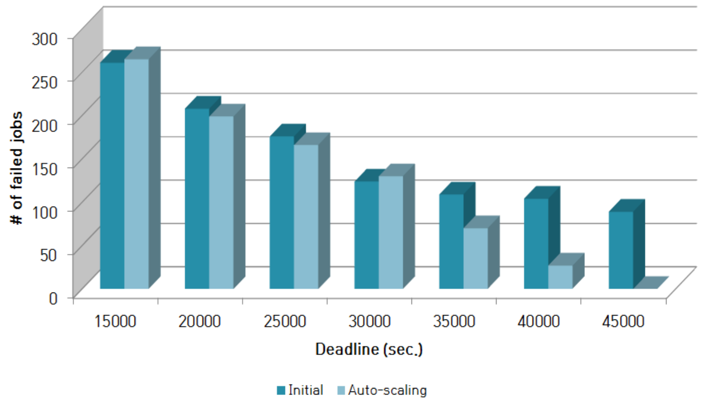

Fig. 1. The number of VM by each deadline

Fig. 7 shows performance of the proposed auto-scaling method comparing to initial scheduling on various deadlines. The simulation was undertaken on an application with 1,000 jobs and 30 jobs among them have various delay. Performance-oriented policy was chosen for Initial scheduling. On the other hand, the auto-scaling method, periodically monitoring the execution of an application, performed re-scheduling dynamically when difference between estimated execution time and real execution time was above threshold.

In this simulation, it shows that the more the deadline increases, the more the number of failed jobs reduces in both auto-scaling and initial scheduling. The failure ratio of jobs using auto-scaling method is relatively lower than initial scheduling since auto-scaling method can handle delays by rescheduling. As a result, all jobs are successfully finished within 45000sec deadline using auto-scaling method, but 89 jobs are not executed using initial scheduling.

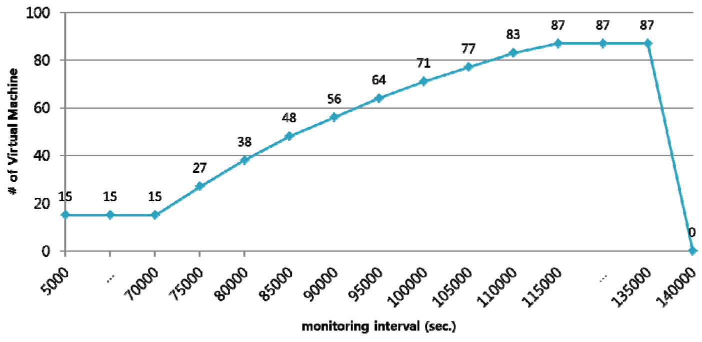

Fig. 2. The changes in the number of VM when delay occurs

Fig. 8 shows scale-out over time when using auto-scaling method. Before we start our simulation, we set delay for 118 of 1060 jobs and deadline to 14000 seconds. Also, monitoring is conducted at 5000 seconds intervals. Auto-scaling method decreases failure rate of jobs by dynamically allocating resources whenever delay occurs. In the simulation, only 15 VMs run from 5000sec to 70000sec and the number of VMs linearly increases from 70000sec. For instance, 27 VMs run at 75000sec and 12 VMs are more extended than before. Finally, the simulation is completed scale-out from 115000sec because deadline violation is not detected.

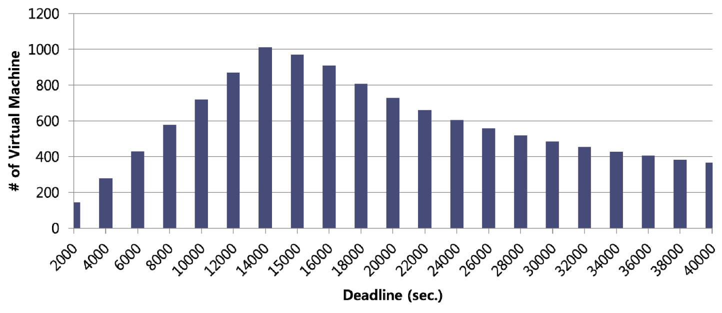

Fig. 3. The number of virtual machine by each deadline

Fig. 9 shows a total number of VM during each interval. In this experiment, all jobs are finished when deadline value is 15000 seconds. Jobs are allocated an order of execution time. Therefore, if jobs have longer execution time than deadline value, they cannot be allocated on VMs and fail in an initial stage. Fig. 9 shows that proposed auto-scaling method doesn't use maximum number of VM until 14000 seconds because of failed jobs. In the simulation, the number of VMs linearly increases up to 14000 seconds in order to be allocated to jobs within deadline. From 15000 seconds to 400000 seconds, the number of VMs decreases. Proposed auto-scaling method allocate resources automatically, it is possible to satisfy an amount of resources actually needed.

V. Conclusion and Future Work

In this paper, we proposed auto-scaling method that effectively managed applications in Bag-of-Tasks [2] type and workflow type in hybrid cloud computing. We conducted experiments with two applications. In the simulation with variable index computation application in Bag-of-Tasks [2] type, the failure ratio of our method showed low even though delay occurred. The method added resources dynamically by rescheduling in order to handle delays. That is resources can be used efficiently as needed using proposed auto-scaling method.

Furthermore, we plan to add diverse polices such as eco-green policy and semantic policy to our auto-scaling method.

References

[1] Hyejeong Kang, Jung-in Koh, Yoonhee Kim, A SLA driven VM Auto-Scaling Method in Hybrid Cloud Environment, APNOMS 2013, Hiroshima, Japan, Sept. 25~28, 2013.

[2] W. Cirne, F. Brasileiro, J. Sauvé, Na. Andrade, D. Paranhos, E. Santos-Neto, R. Medeiros, "Grid Computing for Bag of Jobs Applications," Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and E-Government, September 2003.

[3] Amazon Web Service, http://aws.amazon.com/

[4] Paraleap, https://www.paraleap.com/

[5] Windows Azure, http://www.windowsazure.com/

[6] Scalr, http://scalr.com/

[7] S. Dutta, S. Gera, A. Vermam, and B. Viswanathan, "Smartscale: Automatic application scaling in enterprise clouds", in 5th IEEE International Conference on Cloud Computing (CLOUD), pp. 221-228, June. 2012.

[8] L. Bittencourt, and E. Maderia, "HCOC: A Cost Optimization Algorithm For Workflow Scheduling in Hybrid clouds", Journal of Internet Services and Applications, Vol.2, Springer-Verlag, pp. 207-227, Dec, 2011.

[9] M. Mao, and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows", High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for. Nov. 12-18, 2011.

[10] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids", e-Science and Grid Computing, 2005 First International Conference on IEEE, pp. 8-147, 2005.

[11] L. F. Bittencourt, and E. R. Madeira, "A performance oriented adaptive scheduler for dependent tasks on grids", Concurrency and Computation: Practice and Experience, Vol. 20 Issue. 9, pp. 1029-1049, 2008.

[12] Rodrigo N., Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” Software: Practice and Experience, 41(1):23–50, 2011.

[13] SuperWASP, http://www.superwasp.org/