VM Auto-Scaling for Workflows in Hybrid Cloud Computing

Younsun Ahn
Dept. of Computer Science
Sookmyung Women's University
Seoul, Korea
ahnysun@sookmyung.ac.kr

Yoonhee Kim
Dept. of Computer Science
Sookmyung Women's University
Seoul, Korea
yulan@sookmyung.ac.kr

Abstract— **Appearance of Science Clouds enables scientists to facilitate large-scale scientific computational experiments over cloud environment. Many task computing (MTC) in computational science needs to certificate stable executions of applications even in rapid changes of vital status of physical resources and supports high performance resources in a long period. Auto-scaling approach on virtual machines (VM) increases efficient cloud resources management for the computational problem solving environment. Diverse auto-scaling methods which provide useful resource management presently are being debated and studied. However, most of the auto-scaling methods are just easily considered in performance metrics or execution deadline in specific workloads but not in various patterns of workflow. We propose an auto-scaling method, guaranteeing the execution of various patterns of workflow within deadline time in hybrid cloud environment. The experimental results show the method works dynamically and acceptably on hybrid cloud resources for various workflow patterns having random workload dependency.**

Keywords— auto-scaling; hybrid cloud computing; workflow; workflow dependency;

I. INTRODUCTION

Cloud computing provides on-demand and scalable resources dynamically in order to support application execution. Appearance of Science Clouds enables scientists to facilitate large-scale scientific computational experiments over cloud environment. Many task computing (MTC) needs to certificate stable executions of applications even in rapid changes of vital status of physical resources and support high performance resources in a long time. Therefore, studying computational problem solving environment has been getting more important as it supports the management of task executions or resources in large-scale computation. Auto-scaling approach on virtual machines (VM) increases efficient cloud resources management for the computational problem solving environment. Our previous paper [1] proposed an auto-scaling method to provide efficient resource utilization in a hybrid cloud computing environment. Tasks in Bag-of-Tasks (BoT) [2] can run in parallel while tasks in workflow can be executed in the order of dependency. However, the proposed auto-scaling algorithm limited to specific Bag-of-Tasks in aerodynamics and workflows in protein annotation workflow. We need an auto-scaling method in order to perform applications in a general form of workflows.

This paper proposes an extended version of the auto-scaling method, reflecting in various workflow patterns of tasks based on cloud computing environment. Especially, it dynamically allocates virtual resources depending on tasks in workflow on hybrid cloud environment. We propose an auto-scaling method that can meet a deadline in various workflow patterns. We focus on dynamically allocating VMs in order to maximize resource utilization within a deadline and dealing with task dependency in workflow application. We have evaluated the auto-scaling method with various workflow patterns which have a large number of tasks in hybrid cloud resources. The results of a simulation show the method performs automatically resource allocation satisfying deadline constraints.

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 an auto-scaling algorithm and Section 4 contains contents about a workflow generation. Section 5 explains experiment results. Finally, we conclude the paper and discuss future work in final section.

II. RELATED WORK

Auto-scaling approaches which provide useful resource management presently are being debated and studied. Auto-scaling issues are divided into two sides. First one is rule-based auto-scaling methods such as "Auto-scaling" of AWS [3], Paraleap [4] for Windows Azure [5], and Scalr [6]. This method changes the number of resources by user-defined metrics. However, rule-based auto-scaling methods could lead to execution failure of an individual application in short of consideration on its characteristics such as execution deadline.

In the other side, [7], [8], [9], [10], and [11] are the studies of auto-scaling in consideration of constraints such as a deadline of applications or cost for resource usage. [7] proposes an auto-scaling method minimizing resource usage cost. Horizontal scaling and vertical scaling are used. Horizontal scaling adds or removes the number of VMs and vertical scaling controls the size of a VM. However, this paper only provides resource allocation for Bag-of-Tasks [2] jobs and also dissatisfied resource usage during execution of an application.

This research was supported by Basic Science Research Program through the Natio nal Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2013R1A1A3007866)

978-1-4799-5841-2/14 315.00 © 2014 IEEE
DOI 10.1109/ICCAC.2014.34

However, [9] does not consider various types of workload patterns. [9] lacks considering a mixture of workload patterns. It just performs three special types of workload patterns. It is necessary for our proposed auto-scaling method to consider different workflow patterns to perform automatically. Therefore, we generate various workflow patterns and apply it to our proposed auto-scaling method.

[10] and [11] propose their auto-scaling methods for the execution of workflow applications on Grids. [10] minimizes cost by using sub-deadline and also can reduce execution time for the entire workflow. [11] schedules dependent jobs efficiently. Reference [10]'s and [11]'s workflows are just too simple to evaluate their auto-scaling method. Various patterns of workflows exist in applications. Workflow patterns affect auto-scaling method by the number of tasks and dependency. It may not be possible to schedule an amount of workflow tasks actually needed. We propose an algorithm for workflow referring to [10] and [11].

In this paper, we propose an extended version of an auto-scaling method based on our previous research [1] which can support efficient resource utilization considering the types of jobs in Bag-of-Tasks [2] as well as workflows. Proposed auto-scaling method reflects complex structures of workflow by increasing the number of tasks and dependency. Auto-scaling method can automatically allocate cloud resources by task dependency in a various workflow patterns within deadline.

III. AUTO-SCALING ALGORITHM

We extend [1]'s auto-scaling algorithm which consider only tasks in Bag-of-Task, to support workflow as well. Initial scheduling schedules tasks to prevent waste of VMs within a deadline. Auto-scaling method can perceive delay and deadline violation to comparing actual start time and estimated start time of running tasks during monitoring interval. Algorithm's assumption and notation are referred to [1].

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* = { *tasks* → *VMs*}

```
1:  If SCALING is TRUE
2:      If Job pattern is bag-of-tasks job
3:          Sort waiting tasks in decreasing order of execution length;
4:          S ← Cost-aware Scheduling(sortedTasks, D, minPM);
5:      If Job pattern is workflow.
6:          Sort waiting tasks in sequential order;
7:              S ← WorkflowScheduling(sortedTasks, D);
8:          each vm where status is running
9:              if no running/waiting tasks on vm then
10:                 destroy the vm
11:             send scaling decisions to DRMS
12:             send scheduling decisions to JES
13:         waitForNextInterval();
14:         SCALING ← SLAMonitoring(runningTasks, D);
15:
```

Algorithm 1, Run-time Scaling is extended based on [1]. We newly propose algorithm 2, Workflow Scheduling algorithm to perform tasks in workflow. In the reference [1], Run-time Scaling algorithm can choose an appropriate policy by a pattern of tasks.

Additionally, we extend policies in order to perform workflow as well as Bag-of-Tasks [2] patterns of tasks. Tasks are scheduled with applying one of the two policies such as Cost-aware Scheduling (line 2) and Workflow Scheduling (line 5) of SLA (Service Level Agreements). Cost-aware Scheduling is suitable for a type of tasks in Bag-of-Tasks [2], but Workflow Scheduling is proper to workflow patterns. Cost-aware Scheduling chooses VMs which considered billing time unit to save the cost and user specific minimal performance. Tasks in Bag-of-Tasks [2] are sorted as descending order based on their execution time, while tasks in workflow are performed sequential order.

Algorithm 2 describes our Workflow Scheduling algorithm. The algorithm discovers appropriate a critical path for processing the workflow tasks and schedules the tasks. Proposed Workflow Scheduling algorithm is based on a PCH algorithm [11]. When our auto-scaling method tries to schedule VMs, our method has to adopt a private cloud resource first. When our algorithm tries to allocate public cloud resources for tasks, it considers VMs in the order of running ones, one having the fastest start time, and ones within application deadline. First of all, we find a critical path by using PCH algorithm [11]. Tasks on a critical path are scheduled in a private cloud resource in order to reduce a cost of resource usage (line 3). The total execution time of the critical path is decided by a deadline and additional margin value. It is important to consider task dependency in workflow. Each task could get an EFT (Estimated Finish Time) of a parent task and set an EST (Earliest Start Time) value to EFT of a parent task in order to reflect the order of tasks (line 5). When tasks that are not on a critical path are allocated to VMs, tasks are checked whether their parent tasks have been performed or not. If parent tasks are not allocated in cloud resources, child tasks must wait to be scheduled. The algorithm could calculate an EST of a child task considering EFT of its parent tasks (line 10 (private), line 12 (public)). The algorithm can perform tasks using the appropriate number of VMs. The algorithm can perform tasks using the appropriate number of VMs. Workflow Scheduling algorithm can execute tasks considering task dependency and meeting deadline in a various workflow patterns.

Algorithm 2– **Workflow Scheduling**
Input –Waiting tasks of the application,
Output –Scheduling decision *S* = { *tasks* → *VMs*}, VM list *toStartUp*.for the VMs to be newly created

```
1:  All tasks are scheduled in sequential order
2:  Task in Critical path is allocated VM
3:  If There is no running VM, Find a vm on which all task in Critical path can run within the D;
4:      EST VM is related previous task's EFT VM
5:      Calculate EFT VM by adding EST VM and ET(Execution Time) VM;
6:  Task which has allocated related previous tasks on vm
7:  If There is no running private VM, find a private vm
```

(PM_{vm} >= minPM) on which *task* can start the fastest within the *D;*

8: EST_{VM} is related previous *task's* EFT_{VM}

9: Calculate EFT_{VM} by adding EST_{VM} and ET_{VM};

10: If There is running *private VM,* schedule *task* to *vm;*

11: Continue with the next *task;*

12: If There is no running *public VM*, find a *public vm* *(PM_{vm} >= minPM)* on which task can start the fastest within the *D;*

13: EST_{VM} is related previous *task's* EFT_{VM};

14: Calculate EFT_{VM} by adding EST_{VM} and ET_{VM};

15: If There is running *public VM,* Choose Cheaper *VM* either running *VM or* chosen *VM*

16: Schedule *task* to chosen *vm;*

17: Continue with the next *task;*

IV. Workflow Generation

A workflow is commonly represented by a directed acyclic graph (DAG). In a workflow, tasks have their own order, that is child tasks, can execute when parent tasks are finished. Tasks which have a workflow pattern are important to consider dependency and their order during the auto-scaling method. We experiment our auto-scaling method to prove our Workflow Scheduling can allocate VM to various kinds of workflow patterns. We develop random workflow generation in order to apply a various workflow patterns. We generate various patterns of workflow by using the random number of depth and the random number of parent tasks which represent dependency. And we also make the random number of tasks at each level.

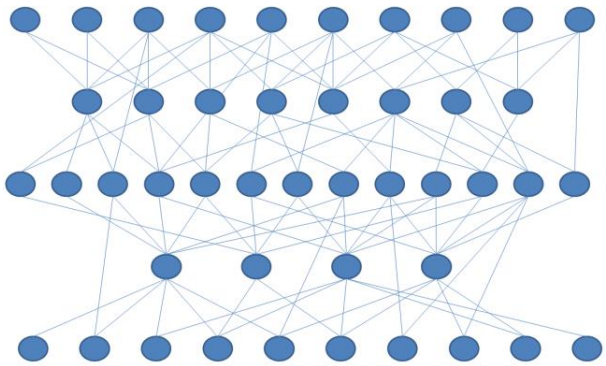

Figure 1. The example of a workflow

V. Expriments

We use CloudSim [12] to simulate various workflows. We simulate our proposed auto-scaling algorithm in hybrid cloud environment. In this experiment, we use four private clouds (600 MIPS) and public clouds (Amazon EC2) and the values of MIPS for public cloud resources range from 200 to 2000. We use a fixed length of tasks in order to analysis the effects of workflow depth and workflow dependency.

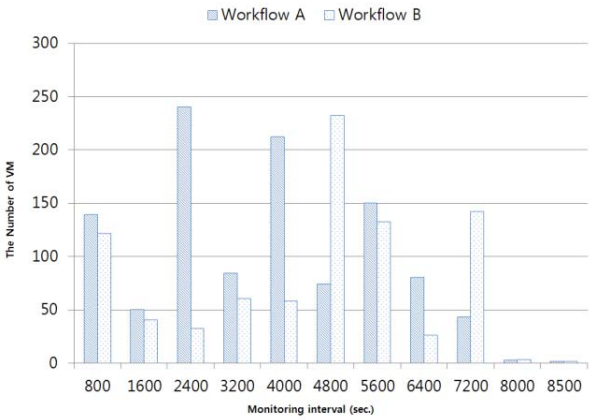

Figure 2. The result of auto-scaling method comparing with various workflows

Fig. 2 shows the performance of the proposed auto-scaling method comparing with two different workflow patterns. Monitoring interval is 800 seconds. We compare two specific workflow patterns among a various workflow patterns to prove that our proposed auto-scaling method automatically allocate tasks to VMs. Workflow A and B have same number of tasks, but they have different workflow patterns. Workflow A and B have 1000 tasks and 25 depths. Workflow A has 47208 dependency edges, while Workflow B has 53661 dependency edges. Workflow A has the number of tasks at each level, 139, 50, 240, 80, 150, 130, 110, 55, 45, and 1. The number of tasks has at a level, 121, 40, 100, 172, 200, 153, 14, 132, 65, and 3 in workflow B. Each workflow finishes and meets the deadline which is 8600 seconds. Initially, Workflow A uses 139 VMs, but Workflow B allocates 121 VMs to perform tasks which can execute in parallel. In 4800 seconds, Workflow A allocates cloud resources less than workflow B, because workflow A has tasks which wait for VM more than workflow B has. Workflow A and B allocate cloud resources dynamically considering dependency within deadline. The Fig. 2 shows our proposed auto-scaling method, allocates resources dynamically actually needed. The key factors for the number change of VM are dependency and the number of each level's tasks. The proposed auto-scaling algorithm successfully performs automatically allocating tasks with dependency in workflows.

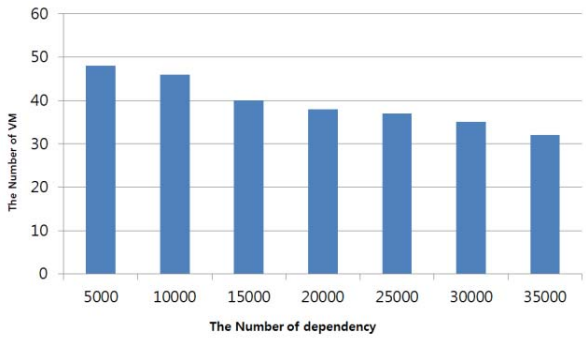

Figure 3. The number of VM by changing the number of dependency

Fig. 3 shows the number of VM and the execution time by changing the number of dependency.

In this experiment, we fix the number of depth in order not to be affected by depth and use 1000 tasks. The Fig. 3 shows the number of VMs have come down during the increasing number of dependency. In all cases, Tasks are finished within deadline, 8600 seconds. A workflow has 5000 dependency edges, it uses 48 VMs. In a case of 35000 dependency edges, it uses the least the number of VMs to complete executions because they have a large number of waiting task.

If a task has many parent tasks, it would wait until all of parent tasks are finished. So, dependency prevents tasks from allocating cloud resources independently.

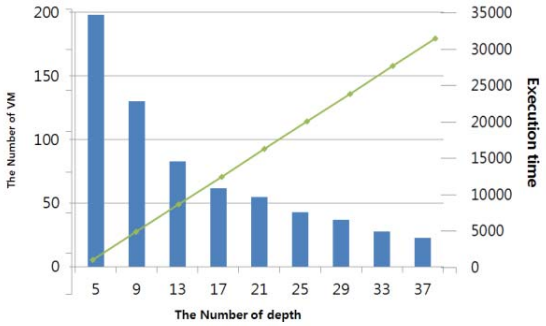

Figure 4. The number of VM by changing the number of depth

In Fig. 4, each the number of depth represents the number of VM and execution time. 1000 tasks are generated for the experiment. The Figure shows the number of VM is affected by the number of depth. It generally decreases the number of VMs when the number of depth increases. The execution time extends by increasing the number of depth. In case of a workflow having 5 depths, it uses 198 VMs in order to finish all tasks within deadline. In a case of one with 17 depths, it allocates 62 VMs. When a workflow has 37 depths, it uses the least amount of VMs. The Figure shows that it reduces the number of the VMs up to 23. It is necessary to consider the number of depth in a workflow. The number influences task's waiting time according to the auto-scaling method.

VI. Conclusion and Future Work

In this paper, we proposed an auto-scaling method that allocated effective resource utilization for workflow in hybrid cloud computing. We conducted experiments with various types of workflow to cover diverse types of applications.

The proposed auto-scaling method performs dynamic resource allocation for diverse workflows within a deadline. Scale-in and scale-out were automatically made within a workflow deadline by considering task dependency in various patterns of workflow.

For the future, we plan to add diverse policies such as semantic policy considering characteristics of an application.

References

[1] Hyejeong Kang, Jung-in Koh, Yoonhee Kim, *A SLA driven VM Auto-Scaling Method in Hybrid Cloud Environment*. APNOMS 2013, Hiroshima, Japan, September 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 21-23 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, December 2011.

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

[10] J. Yu, R. Buyya, and C. K. Tham, *Cost-based scheduling of scientific workflow applications on utility grids*. in 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, December 5-8 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.