

# VM auto-scaling methods for high throughput computing on hybrid infrastructure

Jieun Choi<sup>1</sup> · Younsun Ahn<sup>1</sup> · Seoyoung Kim<sup>2</sup> ·  
Yoonhee Kim<sup>1</sup> · Jaeyoung Choi<sup>3</sup>

Received: 27 February 2015 / Revised: 9 May 2015 / Accepted: 20 May 2015 / Published online: 3 June 2015  
© Springer Science+Business Media New York 2015

**Abstract** Cloud computing provides on-demand resource provisioning and scalable resources dynamically for the efficient use of computing resources. Scientific applications recently need a very large number of loosely coupled tasks to be handled efficiently. In response, current computing environments often consist of heterogeneous resources such as cloud computing. To effectively use cloud resources, auto-scaling methods that consider diverse metrics such as CPU utilization and costs of resource usage have been studied widely. However it still remains a challenge to automatically and timely allocate resources such that deadline violation and application types are considered. In this paper, we propose auto-scaling methods that consider specific conditions such as application types, task dependency, user-defined deadlines and data transfer times within a hybrid computing infrastructure. Our hybrid computing infrastructure consists of local cluster and cloud resources using HTCaaS. We observe noticeable improvements in performance when our auto-

scaling methods for bag-of-tasks and workflow applications is applied.

**Keywords** Auto-scaling · Hybrid infrastructure · Cloud computing · Bag-of-tasks · Workflows

## 1 Introduction

Cloud computing provides on-demand resource provisioning and scalable resources dynamically for efficient use of computing resources. In recent research, scientific applications often need a very large number of loosely coupled tasks to be handled efficiently. These applications are classified into two types which are bag-of-tasks [1] and workflow [2] according to dependency between tasks. At the same time, the computing infrastructure available to applications is becoming more and more heterogeneous, integrating local clusters and private/public clouds. To effectively use cloud resources, auto-scaling methods that consider diverse metrics such as CPU utilization and costs of resource usage have been studied widely.

Our previous paper [3] proposed a VM auto-scaling method to provide efficient resource utilization in hybrid cloud computing environment. However, the proposed auto-scaling algorithm needs to be extended to support various patterns of task execution like bag-of-tasks and workflow. Bag-of-tasks schedules tasks to resources separately from each other, whereas workflow performs tasks in order of dependency patterns.

This paper proposes an extended version of the auto-scaling method reflecting patterns of tasks and the requirements of an application based on hybrid computing infrastructure. We propose auto-scaling methods that finish all tasks, while meeting a deadline and considering task dependency

---

✉ Yoonhee Kim  
yulan@sookmyung.ac.kr  
Jieun Choi  
jechoi1205@sookmyung.ac.kr  
Younsun Ahn  
ahnysun@sookmyung.ac.kr  
Seoyoung Kim  
ssyy77@kisti.re.kr  
Jaeyoung Choi  
choi@ssu.ac.kr

<sup>1</sup> Department of Computer Science, Sookmyung Women's University, Seoul 140-742, Korea

<sup>2</sup> National Institute of Supercomputing and Networking, KISTI, Daejeon 305-806, Korea

<sup>3</sup> School of Computer Science & Engineering, Soongsil University, Seoul 156-743, Korea

and data transfer time. Our hybrid computing infrastructure consists of cluster and cloud resources using HTCAaaS [4]. We analyze ten thousand ( $10^4$ ) tasks using various empirical combination of cluster and cloud resources. We evaluate the auto-scaling methods with bag-of-tasks application and extension protein annotation workflow application [2]. The results of these experiments showed that the methods automatically allocated resources efficiently to user-defined deadline constraints.

The rest of this paper is structured as follows: Sect. 2 discusses related work while Sect. 3 introduces the service architecture of the auto-scaling framework. In Sect. 4, auto-scaling methods are discussed in detail, while experiments and results are presented in Sect. 5. Section 6 concludes the paper and discuss future work.

## 2 Related work

In this section, we introduce several related works with a focus on various aspects such as horizontal/vertical scaling, Bag-of-tasks [1]/Workflow [2], CPU/memory, costs, job deadline and other policies compared with our work.

Liu et al. [5] focused on vertical and horizontal scaling mechanisms that allocate resources according to *cost/efficiency* and the execution pattern of the applications. In horizontal scaling (scaling out/in), new VMs are added or released as needed. On the other hand, vertical scaling (scaling up/down) changes the resources assigned to an already created VM by increasing (or reducing) the allocated CPU power or memory [6]. Bao et al. [7] proposed a novel auto load-aware scale scheme for an OpenStack-based private cloud environment to provide QoS guarantees and ensure system health. Based on prediction algorithms, Bao et al. [7] describes scale-in and scale-out strategies for situations where resources are both sufficient and insufficient. Similarly, in [8] auto scaling in and out is based on prediction algorithms; however, these algorithms take into consideration scaling costs such as virtual resource cost and license costs. Saleh et al. [9] automatically detected complex patterns and relationships among events and performed horizontal auto-scaling based on these patterns.

On the user side, [10], [11], [12], and [13] are studies of auto-scaling considering deadlines for applications or costs of resource usage. There are two kinds of studies that are scheduling a bag-of-tasks which does not consider dependency and workflow scheduling which does consider dependency between tasks. Dutta et al. [10] proposed an auto-scaling method that minimized resource usage costs for bag-of-tasks jobs. It used horizontal scaling which added or removed VMs and vertical scaling which expanded or reduced the size of a VM. However, it is still deficient for resource requirements of dynamic workloads because it lacks

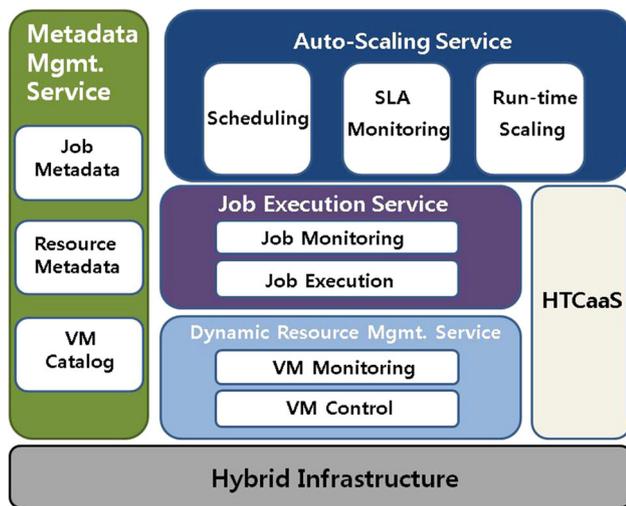
consideration of resource usage during execution of an application.

There are a few studies that have considered workflow scheduling. Mao et al. [11] assessed the workload as a stream of unpredicted workflow jobs and proposed two auto-scaling mechanisms. Mao et al. [11] expects to finish the execution of jobs within deadlines at minimum financial costs. It uses three special types of workload patterns (pipeline, parallel, hybrid). However, this is an insufficient mixture of workload patterns. Thus we considered diverse workflow patterns and applied our proposed auto-scaling methods. Bittencourt et al. [12] eased the execution of workflow applications on Grids, which may be changed to revise the environment. Bittencourt et al. [12] minimizes execution costs and data transmission costs within a given deadline using a proposed auto-scaling algorithm. Bittencourt et al. [12] divides a workflow into partitions and assigns to each partition a sub-deadline; thus, it can minimize execution time for the entire workflow. Abrishami et al. [13] performed a workflow scheduling algorithm that considers a sub-deadline to meet users' deadlines. Abrishami et al. [13] evaluates a proposed algorithm using different structural properties and different sizes of workflow. Bittencourt et al. [14] proposed an efficient scheduling algorithm for dependent jobs using a PCH algorithm. Also, Bittencourt et al. [14] considers the communication costs of data transfer. Sakellariou et al. [15] proposed an algorithm to meet budget constraints within a minimum possible execution time. Niu et al. [16] adopted a cost-effectiveness algorithm to execute tightly coupled applications by integrating resource provisioning. We propose an algorithm for workflow referring to workflow scheduling algorithms.

Based on our previous research [3], an extended version of an auto-scaling method is proposed in this paper. The proposed auto-scaling method can achieve dynamic resource allocation considering types of jobs, from bag-of-tasks to workflow. The auto-scaling methods can automatically allocate cloud resources considering task dependency and data transfer time in various workflow applications.

## 3 Service architecture of auto-scaling framework

We design a service architecture of an auto-scaling framework, as shown in Fig. 1. The architecture is an extended version of that in our previous paper [3] within a hybrid computing infrastructure. It consists of four major services. *Metadata Mgmt. Service (MMS)* maintains information of three different categories—job, resource, VM—and stores mainly static information. *Job Execution Service (JES)* and *Dynamic Resource Mgmt. Service (DRMS)* monitor and control jobs and VMs. *Auto-Scaling Service (ASS)* is a core of the proposed service framework. This service consists of three modules: 'Scheduling', 'SLA Monitoring' and 'Run-time



**Fig. 1** Service architecture

Scaling' which are dependent on each other. The Scheduling module allocates jobs to VMs that fit resource requirements of jobs according to application types and SLA. We consider two kinds of applications and develop two algorithms (described in Sects. 4.2 and 4.3). One is a bag-of-tasks application: loosely coupled tasks without dependency. The other is a workflow application which has a critical path with task dependency. The SLA Monitoring module is responsible for estimating of performance. Performance can be estimated by comparing the biggest Estimated Finish Time (EFT) of the last job on a VM with the deadline in SLA, and whether the EFT would exceed the deadline. The SLA Monitoring module checks deadline violations of running applications and sets scaling to TRUE or FALSE. The SLA Monitoring is conducted at specified interval, and monitoring returns the value of SCALING which indicates whether scaling is necessary. The Run-time Scaling module decides the number of VMs to create or destroy for application execution and then passes the decisions to DRMS and JES.

*HTCaaS* [4] allows users to effectively use available resources in heterogeneous computing environments and efficiently submit a large number of jobs at once. HTCaaS comprises the JES and DRMS levels. Thus, instead of using JES and DRMS, HTCaaS is selected.

A *Hybrid Infrastructure* consists of heterogeneous computing resource such as cluster and public/private cloud computing resources. Amazon EC2 can be used as a public cloud resource and a private cloud can be built using OpenStack [17].

## 4 Auto-scaling methods

Auto-scaling is currently being discussed and studied as a useful resource management approach. With increasing

**Table 1** Notations for three algorithms

Notation	Description
$task_i (i = 1, 2, \dots, N)$	Tasks in an application
$R_i$	Resources which schedule $task_i$
$toStartUp$	List for VM creation
$toShutDown$	List for VM deletion
$EFT_{R_i}$	Estimated finish time of a resource
$EST_{R_i}$	Earliest start time of a task $i$ on a resource
$ET_{R_i}$	Execution time of a task $i$ on a resource
$D$	Deadline of application
$CP$	Critical path in a application

focus on auto-scaling in several studies, much effort has been made to dynamically provide optimized resource scaling in cloud computing environments. Auto-scaling issues are divided into various categories such as application types, performance metric, resource usage, costs, user-defined SLA and QoS.

The auto-scaling processes correspond to the MAPE [4] loop of autonomous systems, which consists of four steps: Monitoring, Analysis, Planning and Execution. The proposed auto-scaling method also considers the following steps:

- *Monitoring*: providing measurements about user requirements, job queue, job status, SLA violation and resource status to use scaling metrics.
- *Analysis*: obtaining data from monitoring phase about current system utilization, and optionally predictions of future needs.
- *Planning*: planning how to scale the resources assigned to the application to satisfy the scaling policies.
- *Execution*: executing the scaling actions decided in the previous step.

In this section, three algorithms that can schedule tasks to VM for bag-of-tasks and workflow applications are introduced. The two algorithms make job scheduling decisions depending on application types. The algorithms' assumption and notation can be seen in [3]. Some notations for the algorithms are presented in Table 1. In all algorithms, computing resources consist of cluster and private and public cloud resources.

### 4.1 Run-time scaling algorithm

Algorithm 1 describes run-time scaling which is overall the auto-scaling algorithm described in our previous research

[3]. Algorithm 1 can be used to scale computing resources for resource utilization while an application is running. Also, it considers application types such as bag-of-tasks and workflow within a given deadline.

---

**Algorithm 1** *Run-time Scaling algorithm*


---

**Input:** An application, application type  $T$ , a deadline  $D$   
1:  $SCALING \leftarrow \text{TRUE}$ ;  
2: **while** (true)  
3:   **if**  $SCALING == \text{TRUE}$   
4:     **switch**  $T$   
5:       **case** bag-of-tasks  
6:          $JS \leftarrow \text{Bag-of-tasksScheduling}(\text{application}, D)$   
7:       **case** workflow  
8:          $JS \leftarrow \text{WorkflowScheduling}(\text{application}, D)$   
9:     **end switch**  
10:    **for each**  $VM$  **where** status is running **do**  
11:     **if** no running/waiting tasks on  $VM$  **then**  
12:       add  $VM$  to  $toShutDown$ ;  
13:     **end if**  
14:    **end for**  
15:    WaitForNextInterval();  
16:     $SCALING \leftarrow \text{SLAMonitoring}(\text{runningTasks}, D)$ ;  
17:    **end if**  
18: **end while**  
**Output:** Scaling decision  $S = \{toStartUp, toShutDown\}$   
Scheduling decision  $JS$   
 $= \{ (task_i, R_i) \mid i = 1, 2, \dots, N, R_i \in (\text{Clusters, Private VMs, Public VMs}) \}$

---

If  $SCALING$  is true (line 3), an application is scheduled using the applicable type of application (lines 4–9). After scheduling, the algorithm checks whether there is any VM that does not have a running job or a queued job. If so, the VMs are added to the  $toShutDown$  list for deletion (lines 10–14). The output of algorithm 1 is the scaling decision  $S$  and the scheduling decision  $JS$ . These outputs are sent to DRMS and JES respectively. The SLA Monitoring module checks deadline violations of running application and sets scaling to TRUE or FALSE. SLA Monitoring is conducted at given intervals, and the monitoring returns the value of  $SCALING$  which indicates whether scaling is necessary.

#### 4.2 Bag-of-tasks scheduling algorithm

Algorithm 2 shows a scheduling algorithm for bag-of-tasks during an auto-scaling process in a cluster and cloud computing environment. First, the algorithm verifies whether there are available cluster resources (line 2). If there are available cluster resources then the task is scheduled to the cluster resources that can then the task earliest within the application deadline (line 3). If there are no available cluster resources then the task is scheduled to the private VM that can start the task the earliest within the application deadline (line 4), but if there is no private VM running then the

scheduler finds a new private VM and adds the VM to the  $toStartUp$  list (lines 5–8). In the same way, if there is no VM running in private then the task is scheduled to public VM (lines 11–17). Finally, the  $task_i$  is scheduled to  $R_i$  and the Estimated Finish Time (EFT) is calculated while considering the Earliest Start Time (EST) and the Execution Time (ET) on the  $R_i$ .

$$EFT_{R_i} = EST_{R_i} + ET_{R_i} \quad (1)$$

---

**Algorithm 2** *Bag-of-tasks scheduling algorithm*


---

**Input:** Waiting application,  $D$   
1: **for each**  $task_i$  **do**  
2:   **if** Available(Cluster) **then**  
3:      $R_i \leftarrow \text{FindEarliest}(\text{Cluster}, D)$ ;  
4:   **else if** Available(Private VM) **then**  
5:     **if** there is no private VM running **then**  
6:        $R_i \leftarrow \text{FindNew}(\text{Private VM}, D)$ ;  
7:       add  $R_i$  to  $toStartUp$ ;  
8:     **else**  
9:        $R_i \leftarrow \text{FindEarliest}(\text{Private VM}, D)$ ;  
10:     **end if**  
11:   **else** // Available(Public VM)  
12:     **if** there is no public VM running **then**  
13:        $R_i \leftarrow \text{FindNew}(\text{Public VM}, D)$ ;  
14:       add  $R_i$  to  $toStartUp$ ;  
15:     **else**  
16:        $R_i \leftarrow \text{FindEarliest}(\text{Public VM}, D)$ ;  
17:     **end if**  
18:   **end if**  
19:   Schedule  $task_i$  to  $R_i$ ;  
20:    $EFT_{R_i} \leftarrow EST_{R_i} + ET_{R_i}$ ;  
21: **end for**  
**Output:** Scaling decision  $S = \{toStartUp\}$   
Scheduling decision  $JS$   
 $= \{ (task_i, R_i) \mid i = 1, 2, \dots, N, R_i \in (\text{Clusters, Private VMs, Public VMs}) \}$

---

#### 4.3 Workflow scheduling algorithm

Algorithm 3 shows the workflow scheduling procedure. The proposed auto-scaling technique can discover delay and deadline violations by comparing actual start times and estimated start times of running tasks. Tasks in the workflow are sorted in sequential order and are connected with other tasks. Thus each task could get an EFT from a related previous task and set an EST value to the EFT of a related previous task to consider the order of tasks.

The proposed workflow scheduling algorithm is based on a PCH algorithm [14]. The scaling method tries to allocate cloud resources based on SLA monitoring. We can get tasks on a critical path by grouping the tasks using the PCH algorithm [14]. The total execution time of a critical path in a

resource is calculated and set to a deadline value. Also, additional time is added to the deadline value.

---

**Algorithm 3** Workflow scheduling algorithm
 

---

**Input:** Waiting application,  $D$

```

1: for each  $task_i$  considering  $EFT_{R_j}$  do
2:   if  $task_i \in CP$  then
3:     if  $task_i$  is the first task  $\in CP$  then
4:       if Available(Cluster) then
5:          $R_i \leftarrow$  FindEarliest(Cluster,  $D$ ,  $CP$ );
6:       else if Available(Private VM) then
7:          $R_i \leftarrow$  FindEarliest(Private VM,  $D$ ,  $CP$ );
8:       else // Available(Public VM)
9:          $R_i \leftarrow$  FindEarliest(Public VM,  $D$ ,  $CP$ );
10:      else // not a first task  $\in CP$ 
11:         $R_i \leftarrow$  Current  $R$  on which all tasks  $\in CP$ 
           within the  $D$ ;
12:      end if
13:    else
14:      if Available(Cluster) then
15:         $R_i \leftarrow$  FindEarliest(Cluster,  $D$ );
16:      else if Available(Private VM) then
17:        if there is no private VM running then
18:           $R_i \leftarrow$  FindNew(Private VM,  $D$ );
19:          add  $R_i$  to  $toStartUp$ ;
20:        else
21:           $R_i \leftarrow$  FindEarliest(Private VM,  $D$ );
22:        end if
23:      else // Available(Public VM)
24:        if there is no public VM running then
25:           $R_i \leftarrow$  FindNew(Public VM,  $D$ );
26:          add  $R_i$  to  $toStartUp$ ;
27:        else
28:           $R_i \leftarrow$  FindEarliest(Public VM,  $D$ );
29:        end if
30:      end if
31:    end if
32:    Schedule  $task_i$  to  $R_i$ ;
33:     $EST_{R_i} \leftarrow EFT_{R_j}$ 
34:     $EFT_{R_i} \leftarrow EST_{R_i} + ET_{R_i}$ 
           if  $(R_i \neq R_j) + DTT_{ij}$ ;
35:  end for

```

**Output:** Scaling decision  $S = \{toStartUp\}$

Scheduling decision  $JS$

$$= \{ (task_i, R_i) \mid i = 1, 2, \dots, N, \\ R_i \in (\text{Clusters, Private VMs, Public VMs}) \}$$


---

First, each  $task_i$  in a workflow application, we must consider a related previous  $task_j$  and the EFT of  $task_j$  on resource  $R_j$  (line 1). Next, tasks on a critical path are scheduled on the same resource, which can execute all tasks in a critical path (line 2). If the  $task_i$  is the first task in a critical path,  $task_i$  is scheduled to  $R_i$  which is the available resource on which all tasks in a critical path can run within the deadline (lines 3–9). If there are available cluster resources, the scheduler finds a cluster on which all tasks in a critical path can run within the deadline (lines 4–5). In our hybrid computing infrastructure, it is rare that tasks fail in cluster resources. For this reason, cluster resources are considered first for workflow applications. Thus in scheduling tasks, it

is a rule to choose private cloud resources (lines 6–7) prior to public cloud resources (lines 8–9). After that, other tasks in a critical path are scheduled to the currently selected resource (lines 10–12). The scheduler then schedules tasks that are not in a critical path (lines 14–31) with consideration to a related previous task’s EFT. If there are available cluster resources, then the task is scheduled to the cluster resources that can execute the task earliest within the application deadline (line 14). If there are no available cluster resources, the task is scheduled to the private VM that can start the task earliest within the application deadline (line 21), but if there is no private VM running then the scheduler finds a new private VM and adds the VM to the  $toStartUp$  list (lines 17–19). In the same way, if there is no VM running in private then the task is scheduled to public VM (lines 24–30). Finally, the  $task_i$  is scheduled to  $R_i$  and EFT is calculated (line 32–34). It is important to consider task dependencies and data transfer time in workflows. In this algorithm, Data Transfer Time (DTT) from  $task_j$  to  $task_i$  can be defined as shown in Eq. 2.

$$DTT_{ij} = \frac{Data(task_i, task_j)}{NetworkBandwidth} \quad (2)$$

With the execution of tasks on the critical path on the same resource, communication overhead is reduced. That is, when parent tasks and child tasks are all performed on the same VM, data transfer time is zero. When tasks that are not on a critical path are scheduled to VMs, tasks check a parent tasks state. If parent tasks are allocated to cloud resources, child tasks could be allocated as well. Otherwise, while parent tasks are scheduled to cloud resources, child tasks must wait to be allocated. As mentioned above, each task could get an EFT of a parent task and calculate an EST value considering EFT of a parent task and data transfer time. This can be calculated as presented below:

$$EFT_{R_i} = EST_{R_i} + ET_{R_i} + DTT_{ij} \quad (3)$$

## 5 Experiments

Experiments that validated our auto-scaling methods are presented in this section. First, the hybrid computing infrastructure and empirical combination of hybrid resources is presented in Sect. 5.1 and the subsequently, the experimental results are shown in Sects. 5.2 and 5.3.

### 5.1 Experiment environment

The hybrid computing infrastructure consist of local cluster and private cloud resources using HTCaaS [4]. HTCaaS allows users to effectively use available resources in a heterogeneous computing environment and efficiently submit

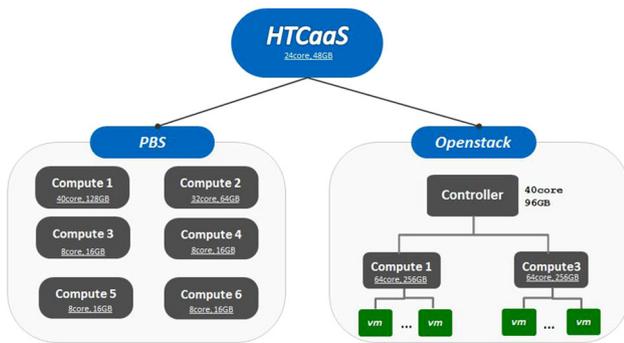


Fig. 2 System architecture

a large number of jobs at once. Figure 2 shows the system architecture using HTCaaS. We use OpenStack [17], an open source software that provides large pools of compute, storage and networking resources used for the private cloud.

Table 2 shows the specifications of the cluster machines and Table 3 shows the cloud machines’ specifications.

On the hybrid computing infrastructure, we conducted empirical experiments that collected preliminary data for the bag-of-tasks application. The application used in our experiments consisted of a large number of tasks. The tasks have an average of 10 s as their execution time.

Figure 3 shows the experimental results. We analyze the application using various empirical combinations of cluster and cloud resources in our hybrid computing infrastructure. The application has ten thousand ( $10^4$ ) tasks executed in the cluster and the cloud. The x-axis represents the number of tasks that are executed using a combination of cluster and cloud resources; that is, the number of tasks in the cluster/the number of tasks in the cloud. We use cluster resources that have 104 cores and use cloud resources that have 208 virtual machines and each VM was created identically using Ubuntu 12.04 Server image with a memory allocation of 3GB and 1 vCPU. In Fig. 3, a combination (6000/4000) performs 6000 tasks in cluster resources and executes 4000 tasks in VM represented the ‘best’ result.

Table 2 Cluster machines’ specifications

	CPU	Core	RAM
Compute 1	2 × Intel Xeon Processor E5-2690v2 (3.0 GHz, 25 MB L3, QPI 8.0 GT/s, 115 W, 10-Core)	40	128 GB
Compute 2	Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60 GHz	32	64 GB
Compute 3	Intel(R) Xeon(R) CPU X5560 @ 2.80 GHz	8	16 GB
Compute 4~6	Intel(R) Xeon(R) CPU E5420 @ 2.50 GHz	8	16 GB

Table 3 Cloud machines’ specifications

	CPU	Core	RAM
Controller	Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20 GHz	40	96 GB
Compute 1	AMD Opteron(tm) Processor 6378	64	256 GB
Compute 2	AMD Opteron(tm) Processor 6378	64	256 GB

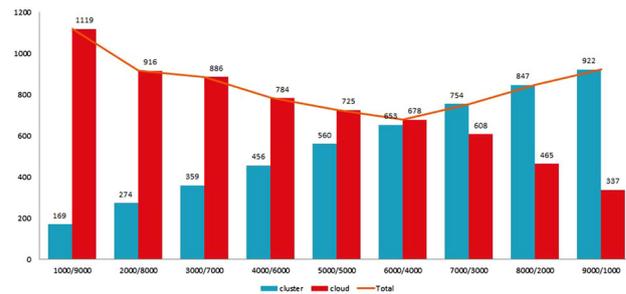


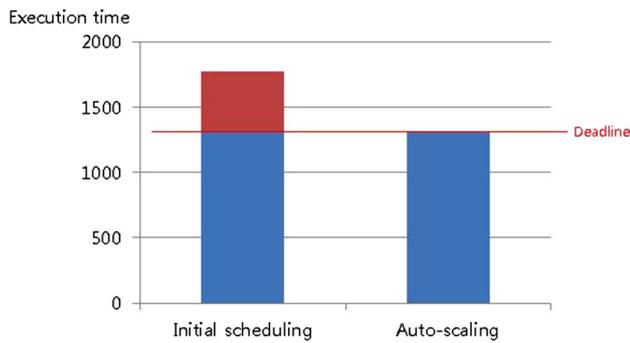
Fig. 3 Empirical combining cloud and cluster resources

### 5.2 Auto-scaling for bag-of-tasks applications

The proposed auto-scaling algorithm for bag-of-tasks applications was simulated using CloudSim [23] which has the environment identical to Tables 2 and 3. Figure 4 shows the result of the simulation for bag-of-tasks applications, comparing auto-scaling and initial scheduling. The same application from the experiment above was used in the proposed hybrid computing infrastructure. The application has ten thousand ( $10^4$ ) tasks: 6000 tasks in cluster resources and 4000 tasks in cloud resources. During the simulation, there is a delay from  $-20$  s to  $+20$  s at 400 tasks. During initial scheduling, tasks are executed regardless of deadline violations since initial scheduling does not include rescheduling of tasks. In contrast, the auto-scaling algorithm tries to reschedule tasks that may violate the deadline. When a deadline violation occurs, the proposed auto-scaling algorithm works better than the initial scheduling. In the initial scheduling, 212 tasks fail while all tasks were executed successfully within the given deadline using the auto-scaling approach. So, the proposed auto-scaling algorithm for bag-of-tasks application can prevent task failure compared to initial scheduling.

### 5.3 Workflow applications

The concept of the proposed workflow algorithm is demonstrated in a simulation in CloudSim [23] using a protein anno-



**Fig. 4** Auto-scaling for bag-of-tasks

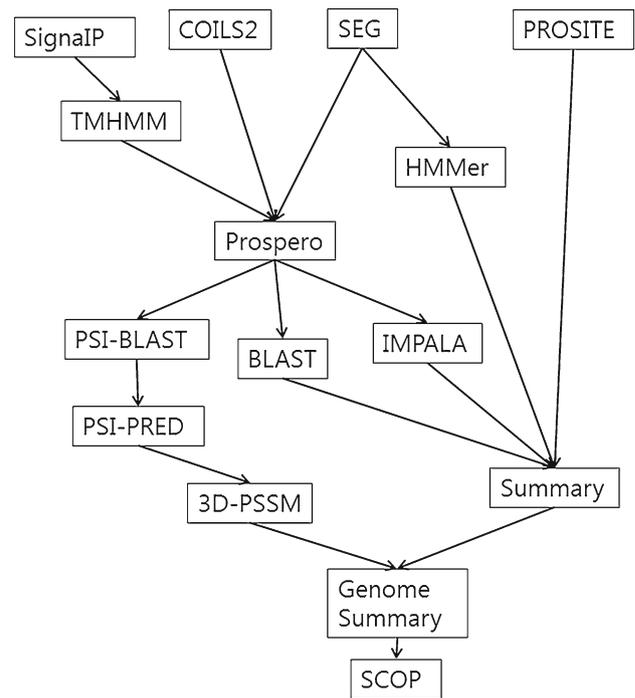
tation workflow. The simulation of the proposed auto-scaling algorithm is conducted in a hybrid computing infrastructure. Section 5.3.1 shows test applications and discusses workflow auto-scaling with data transfer times. Section 5.3.2 compares algorithm 3 to deadline-MDP [12] and Sect. 5.3.3 discusses our auto-scaling performance when a deadline violation occurs.

### 5.3.1 Extension of protein annotation workflow

The London e-Science Center developed the Protein annotation workflow [2] which has dependent tasks and is able to deal with large amounts of information. Also, it provides 15 services that require several steps to fulfill each. These services are performed sequentially, and each service in a workflow generates output data required by the child services as inputs, as shown in Fig. 5. Among the services, HMMer, IMPALA, BLAST, PSI-BLAST, and PSI-PRED are considered in particular in this experiment. Each services explanation can be found in [18–22]. In the experiment, input data and output data are key factors in considering data transfer times. Because HMMer, IMPALA, BLAST, PSI-BLAST, and PSI-PRED services require large input data, the values of each data transfer time are different. Especially, the PSI-BLAST service requires not only large input data, but also large output data as following PSI-PRED service requires all the output data from PSI-BLAST.

Figure 6 shows an extension of a protein annotation workflow. The number of tasks is 57 and the length of tasks is featured in parentheses in Fig. 6. Protein annotation workflow [2] applications services are performed sequentially and generate output data required by the child services as inputs. The I/O data of the workflows range from 1.2 to 29.2 GB. The available network bandwidth between services is 100 Mbps. In this experiment (Figs. 7, 8, 9), four private clouds having 330 MIPS and public clouds having 600 and 2400 MIPS are used.

In Fig. 7, an extension of the protein annotation workflow using only hybrid cloud resources is used. In Fig. 7, the



**Fig. 5** E-protein service structure [2]

two graphs show the number of VM used when the workflow pattern includes I/O data and when it has no I/O data. The dot pattern graph in Fig. 7, represents the result of auto-scaling when considering I/O data. A diagonal pattern graph shows the result of auto-scaling without considering I/O data, where the data transfer time is zero. The graph without considering I/O data executes all tasks within 7200 s. However, with considering I/O data, the graph shows finishing within 8500 s. The auto-scaling algorithm successfully perform dynamically allocation tasks actually needed in workflow considering task dependency and data transfer time. At the beginning of the execution time, both graphs used the same number of VMs. Tasks are not affected by data transfer time, but only influenced by dependency. So, tasks are allocated to VMs in parallel. After 1800 s, considering the I/O data graph using less VM than without considering the I/O data graph does because of waiting tasks. In 4200 s, with considering the I/O data graph shows that waiting tasks can execute at the same time.

Figure 8 presents the percentage of tasks completed at each monitoring interval using only hybrid cloud resources. The graph without I/O data finishes at 7200 s. However, the graph with I/O data accomplishes all tasks within 8500 s, because tasks are influenced by data transfer times. In the case of executions with I/O data, tasks wait to be allocated until parent tasks are finished. From the graph, before 1800 s, the lines show no difference. After 1800 s, however, tasks cannot be allocated to VM and wait for data transfer time

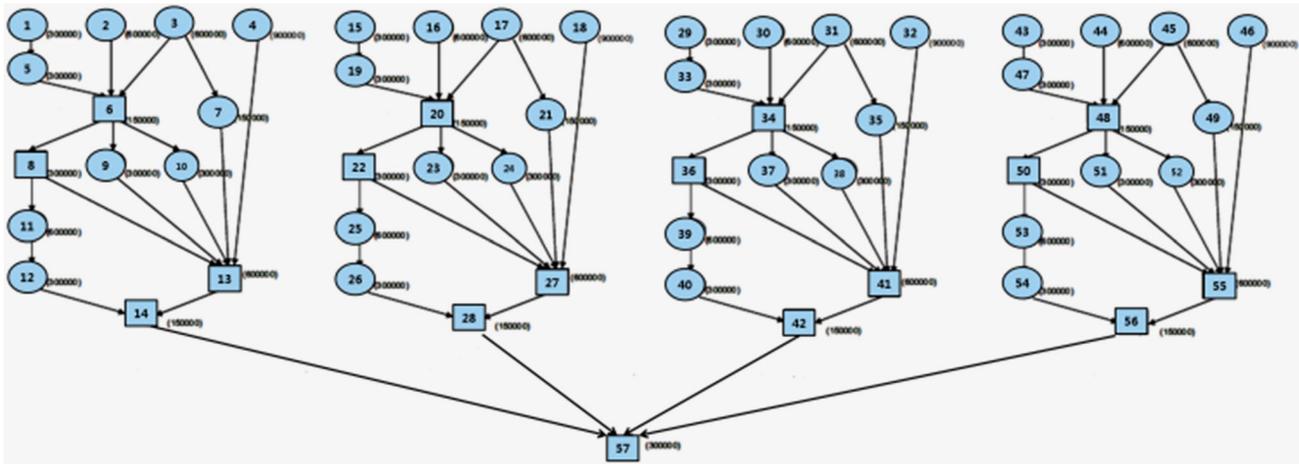


Fig. 6 Extension of protein annotation workflow

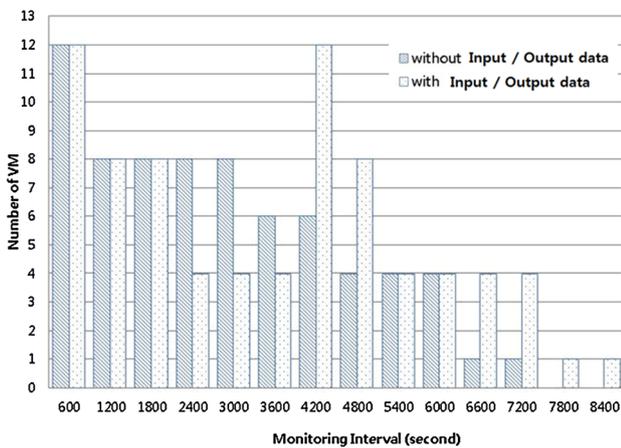


Fig. 7 The result of extension of protein annotation workflow

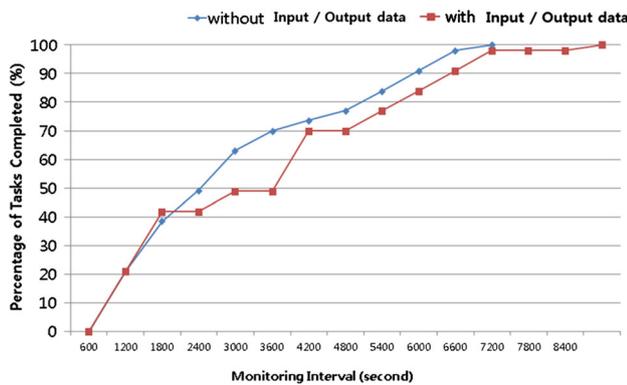


Fig. 8 The percentage of tasks completed

with I/O data. In 3600 s, the graph without I/O data shows that 70 % of tasks are finished. On the other hand, the graph with I/O data reveals that less than 50 % of tasks are allocated to VM.

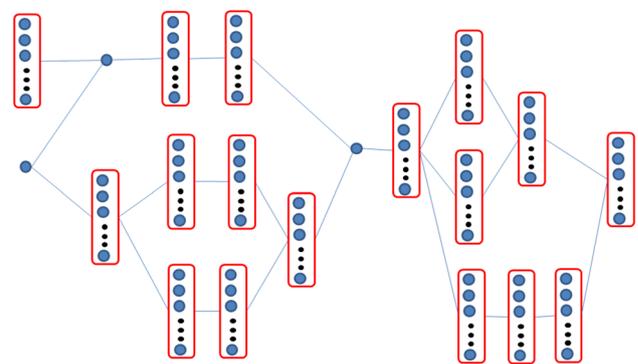


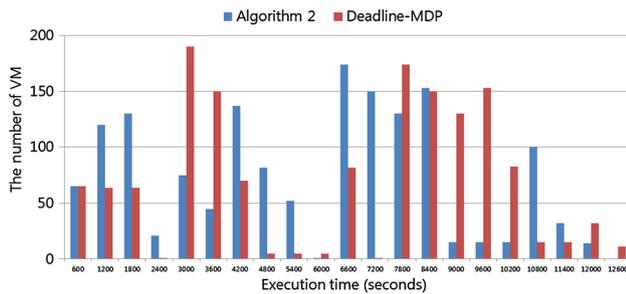
Fig. 9 Simple workflow pattern

Thus, data transfer times influence execution times and VM usage. The proposed auto-scaling method can complete the tasks as soon as possible by automatically allocating VMs, considering data transfer time and dependency.

### 5.3.2 Comparing algorithm 3 with deadline-MDP

The same workflow presented in Fig. 9 was generated to compare algorithm 3 with deadline-MDP [12]. Figure 9 presents a simple workflow pattern. In Fig. 9, some tasks merge with other tasks and the number of dependencies is 2150. The number of tasks is 1000 and the length of tasks ranges from 15,000 to 90,000. The monitoring interval is 600 s.

Figure 10 shows the result of comparing algorithm 3 with deadline-MDP using the workflow presented in Fig. 9. Bittencourt et al. [12] proposed deadline-MDP which is a workflow scheduling algorithm using sub-deadlines. Deadline-MDP divides deadlines with synchronization tasks to meet the whole deadline, whereas algorithm 3 finds the critical path to arrange deadlines and allocates VMs with



**Fig. 10** Comparing algorithm 3 to deadline-MDP [12]

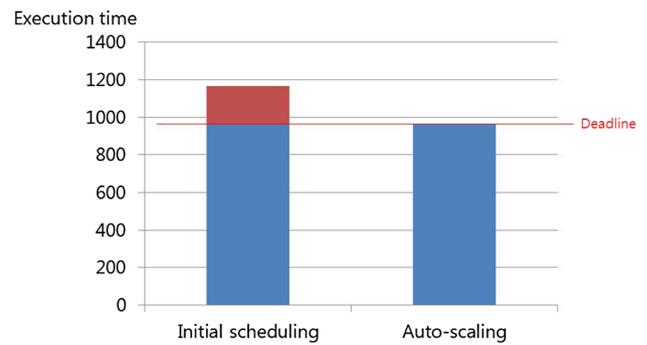
consideration of dependency and data transfer time; sub-deadlines are not considered in algorithm 3.

Some tasks may be restrict to allocated VMs using sub-deadlines in special cases. Thus, sub-deadline may be used for a number of VMs momentarily after meeting a sub-deadline. Deadline-MDP extends total execution time because deadline-MDP cannot sometimes allocate VM to keep sub-deadlines. Deadline-MDP also uses an inefficient amount of VMs after keeping a sub-deadline. For example, after 7200 s, deadline-MDP uses 174 VMs to execute tasks that were waiting for the sub-deadline time. This is an inefficient way to use resources. Algorithm 3 appropriately allocates VMs by using 82 VMs at 4800 s and 150 VMs at 7200 s. The results show that the proposed algorithm finishes all tasks within 12,000 s while deadline-MDP completes within 12,800 s. Algorithm 3 runs faster than deadline-MDP and uses the resources with maximum efficiency.

Additionally, deadline-MDP may not be appropriate for allocating many tasks in the initial stages because they impacts on sub-deadlines. Allocating an amount of tasks to VMs at initial stages is efficient by following initial scheduling, because delay can occur in run times and then rescheduling would be needed.

### 5.3.3 Auto-scaling for workflow applications

The proposed auto-scaling algorithm for workflow applications is simulated within a hybrid computing infrastructure that includes local cluster and cloud resources. Figure 11 shows the result of a simulation of workflow applications compared with auto-scaling and initial scheduling. The workflow application, which extends from Fig. 6 and has 228 tasks, was used for this simulation. An assumption that the 48 tasks that are not in the critical path have a 100 s delay is made, because all tasks in a critical path scheduled cluster resources that are not scaled even if a delay is caused. Similarly, during initial scheduling, tasks are executed regardless of deadline violations since initial scheduling does not include rescheduling of tasks. In contrast, the auto-scaling algorithm tries to reschedule tasks that may violate a deadline. When a deadline violation occurs, the proposed auto-scaling algorithm works



**Fig. 11** Auto-scaling for workflow

better than the initial scheduling approach. In initial scheduling, some tasks fail while no tasks fail within a deadline in auto-scaling. Thus, the proposed auto-scaling algorithm for workflow application can protect against task failure better than initial scheduling.

## 6 Conclusions and future work

This paper proposes auto-scaling methods that effectively manage bag-of-tasks applications and workflow applications within a hybrid computing infrastructure. Our hybrid computing infrastructure consisted of local cluster and cloud resources. We analyzed the application using various empirical combinations of cluster and cloud resources in our hybrid computing infrastructure. We conducted simulations that showed performance achievements of our algorithm with the bag-of-tasks application. Also, we carried out simulations with the protein annotation workflow application and used various kinds of workflow. In the simulation, our proposed workflow scheduling algorithm automatically allocated VMs within the deadline, while considering application types, task dependency and data transfer time in a hybrid computing infrastructure. The proposed auto-scaling methods can eliminate task failure while an application is running. In the future, we will implement our auto-scaling methods for different scientific workflows based on various policies.

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

## References

1. Cirne, W., Brasileiro, F., Sauve, J., Andrade, N., Paranhos, D., Santos-Neto, E., Medeiros, R.: Grid computing for bag of jobs applications. In: Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and E-Government, 21–23 Sept 2003
2. O'Brien, A., Newhouse, S., Darlington, J.: Mapping of Scientific Workflow Within the E-Protein project to Distributed Resources. In: UK E-Science All Hands Meeting, Nottingham (2004)

3. Kang, H., Koh, J., Kim, Y.: “A SLA driven VM auto-scaling method in hybrid cloud environment. In: Network Operations and Management Symposium (APNOMS), 2013 15th Asia-Pacific, Hiroshima, Japan, 25–28 Sept 2013
4. High-Throughput Computing as a Service (HTCaaS), <http://htcaas.kisti.re.kr/>
5. Liu, C.-Y., Shie, M.-R., Lee, Y.-F., Lin, Y.-C., Lai, K.-C.: Vertical/horizontal resource scaling mechanism for federated clouds (2014)
6. Lorido-Bortran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. *J. Grid Comput.* **12**(4), 559–592 (2014)
7. Bao, J., Lu, Z., Wu, J., Zhang, S., Zhong, Y.: Implementing a novel load-aware auto scale scheme for private cloud resource management platform. In: Network Operations and Management Symposium (NOMS) (2014)
8. Yang, J., Liu, C., Shang, Y., Cheng, B., Mao Z., et al.: A cost-aware auto-scaling approach using the workload prediction in service clouds. In: 6th IEEE International Conference on Cloud Computing (CLOUD), pp. 810–815 (2013)
9. Saleh, O., GropengieBer, F., Betz, H., Mandarawi W., Sattler, K.: Monitoring and auto-scaling iaas clouds: a case for complex event processing on data streams. In: 6th IEEE/ACM International Conference on Utility and Cloud Computing, pp. 387–392 (2013)
10. Dutta, S., Gera, S., Vermam A., Viswanathan, B.: Smartscale: automatic application scaling in enterprise cloud. In: 5th IEEE International Conference on Cloud Computing (CLOUD), pp. 221–228 (2012)
11. Mao, M., Humphrey, M.: Scaling and Scheduling to Maximize Application Performance within Budget Constraints in Cloud Workflows. In: IEEE 27th International Symposium on Parallel and Distributed Processing (2013)
12. Yu, J., Buyya R., Tham, C.K.: Cost-based scheduling of scientific workflow applications on utility grids. In: 1st IEEE International Conference on E-Science and Grid Computing, Melbourne, 5–8 Dec 2005
13. Abrishami, S., Naghibzadeh, M., Epema, D.H.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *J. Future Gener. Comput. Syst.* **29**(1), 158–169 (2013)
14. Bittencourt, L.F., Madeira, E.R.: A performance oriented adaptive scheduler for dependent tasks on grids. *Concurr. Comput.* **20**(9), 1029–1049 (2008)
15. Rizos, S., et al.: Integrated Research in GRID Computing. Scheduling workflows with budget constraints. Springer, Berlin (2007)
16. Niu, S., et al.: Cost-effective cloud HPC resource provisioning by building semi-elastic virtual clusters. In: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, p. 56 (2013)
17. OpenStack, <https://www.openstack.org/>
18. HMMER, <http://hmmer.janelia.org/>
19. IMPALA, <http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html>
20. Johnson, M., Zaretskaya, I., Raytselis, Y., Merezuk, Y., McGinnis, S., Maddent, T.L.: NCBI BLAST: a better web interface. *Nucl. Acids Res.* **36**, W5–W9 (2008)
21. Bergman, N.H., Bhagwat, M., Aravind, L.: PSI-BLAST Tutorial (2007)
22. PSI-PRED, <http://bioinf.cs.ucl.ac.uk/index.php?id=779>
23. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **41**(1), 23–50 (2011)



**Jieun Choi** is currently a Masters’ student in the Department of Computer Science, Sookmyung Women’s University. She received her B.S. degree from Sookmyung Women’s University in 2014. She is also researcher at the Distributed & Cloud Computing Lab of Sookmyung Women’s University. Her research interests include cloud computing and management in distributed computing systems.



**Yoonsun Ahn** is currently a Masters’ student in the Department of Computer Science, Sookmyung Women’s University. She received her B.S. degree from Sookmyung Women’s University in 2013. She is also researcher at the Distributed & Cloud Computing Lab of Sookmyung Women’s University. Her research interests include hybrid cloud computing, ontology and intelligent systems.



**Seoyoung Kim** is a researcher in National Institute of Supercomputing and Networking (NISN) at KISTI (Korea institute of Science and Technology Information). She received her B.S. and M.S. degree from Sookmyung Women’s University in 2010 and 2012, respectively. She has worked in NISN, KISTI from 2012. Her research interests focus on meta-scheduling in distributed computing systems, particularly (on) High-Throughput Computing, Many-Task Computing.



**Yoonhee Kim** is a professor in the Department of Computer Science, Sookmyung Women's University. She received her B.S. degree from Sookmyung Women's University in 1991, her M.S. degree and Ph.D. from Syracuse University in 1996 and 2001, respectively. She was a Research Staff Member at the Electronics and Telecommunication Research Institute during 1991 and 1994. Before joining the faculty of Sookmyung Women's University in 2001, she

was on the faculty of the Computer Engineering Department at Rochester Institute of Technology in NY, USA. Her research interests span many aspects of runtime support and management in distributed computing systems.



**Jaeyoung Choi** received the B.S. degree in Department of Control and Instrumentational Engineering, from Seoul National University, Seoul, Korea, in 1984, the M.S. degree in Department of Electrical Engineering, University of Southern California in 1986, and the Ph.D. degree in School of Electrical Engineering from Cornell University in 1991. He has previously worked at Oak Ridge National Laboratory (1992–1994) and University of Tennessee, Knoxville (1994–

1995) as a postdoctoral research associate and a research assistant professor, respectively, where he had been involved with the ScaLAPACK project. He is currently a professor of School of Computer Science and Engineering at Soongsil University, Seoul, Korea. His research interests include high-performance computing (HPC), cloud computing, and distributed middleware.