Cluster Comput (2016) 19:1515–1526
DOI 10.1007/s10586-016-0588-3

Adaptive application-aware job scheduling optimization strategy in heterogeneous infrastructures

Seoyoung Kim[1] · **Jieun Choi**[2] · **Yoonhee Kim**[2]

Received: 29 February 2016 / Accepted: 20 June 2016 / Published online: 29 June 2016

Abstract As the rapid advancement and diversity in the computing systems, it is demanding to take the most robust scheduling algorithm that guarantee an optimized performance to execute manifold applications on large-scale heterogeneous computing environments. This paper present an adaptive application-aware job scheduling optimization strategy for large-scale high throughput computing in heterogeneous infrastructures. The proposed scheduling optimization method is built on two main concepts. First, it provides application-aware job distribution weights through empirical data in large-scale heterogeneous infrastructures. Here we adopt the concept of *weight*, which represent the ratio of tasks that will be computed on each resource. The weights can vary in terms of application type and are optimized until the system get in steady status. Secondly, it offers an adaptive control phase that is invoked by the weight adjustment and resource scaling feature. The feedback data from monitoring module is forwarded to the control phase in order to adjust weights and over-provisioning ratio, and result in enhancing overall balance between performances and utilization of system. The experimental evaluation with the four realistic workload patterns demonstrates that, when compared to the core-based scheme which distributes tasks in proportions of each resource's number of cores, the use of our optimization method can achieve 62 % better average throughput, 43 % shorter average queueing time, and 38 % better average utilization of the entire resources in diverse infrastructure environments by harnessing our adaptive module.

Keywords Job scheduling optimization · Adaptive evaluation · Application-aware · High-throughput computing

1 Introduction

As demand for computing resources is increasing, distributed environments have evolved into versatile forms by employing various emerging computational technologies. New properties of such systems lead to seek sophisticated solutions for scheduling strategy as well as its optimization in order to overcome the performance degradation and to leverage the systems effectively. Moreover, properly scheduling scientific computing jobs to the heterogeneous resources in order to achieve good performance at low cost is still challenging since performances vary depending on where or when the applications are executed.

This paper presents an application-aware adaptive job scheduling optimization strategy in heterogeneous infrastructures, especially for large-scale high throughput computing applications. This optimization method has two main topics. First, it offers application-aware job distribution weights that are based on the empirical data in large-scale heterogeneous infrastructures. We exploit the concept of *weight* in order to represent the ratio of tasks that will be computed on each resource. The weights can vary in terms of applica-

✉ Yoonhee Kim
yulan@sookmyung.ac.kr

Seoyoung Kim
sssyyy77@kisti.re.kr

Jieun Choi
jechoi1205@sookmyung.ac.kr

[1] Division of Supercomputing, KISTI, Daejeon 34141, Korea

[2] Department of Computer Science, Sookmyung Women's University, Seoul 140-742, Korea

tion type and are optimized until that system get in steady status. Secondly, it provides an adaptive control phase that is invoked by the weight adjustment and resource scaling feature. The feedback information from monitoring module is forwarded to the control phase in order to adjust weights and over-provisioning ratio, and lead to enhancing overall balance between performances and economy of system.

We also design a service architecture model that delineates how this approach can apply and operate in general. We implement and demonstrate the algorithms along with middleware to compose our testbed. The experimental assessment exploiting the four types of realistic workload pattern proves that, when compared to the `Core-based` manner which allocates tasks in proportions of each resource's number of cores, our optimization approach can yield 62 % better average throughput results, 43 % shorter average queueing time, and 38 % better average utilization of the total resources in diverse infrastructure environments by harnessing our adaptive module. The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 details the service architecture model the proposed approaches work in and main algorithms. We present our evaluation and its result in Sect. 4. Finally, we conclude this paper in Sect. 5.

2 Related work

Regarding distributed computing systems, diverse job scheduling schemes and their optimization methods have been steadily researched and proposed in the last few decades, (such as integrated systems and heterogeneous environments). In this section, we will introduce some existing researches focused on the three approaches as follows.

2.1 Application-aware

First, we introduce the studies related with application-aware approach. Recently, there have been many researches about scheduling algorithm for several data-intensive applications in data-analysis and HPC communities [5,6,18]. They insisted on a novel approach which is based on data-locality, data size, location of storage, file transfer time and network bandwidth to optimize scheduling of data-intensive applications.

Yang et al. [20] presented an application scheduling framework that uses a uniform model to predict a kind of applications such as cpu-intensive application, i/o-intensive application, and different mixed applications. It leverages performance interference by prediction techniques which act as the core for an application scheduling in the virtualized environment. They conduct various experiments and the results show the effectiveness of the proposed method for predicting the performance interference.

Gupta et al. [11] devised an application-characteristics aware VM placement technique which is a combination of HPC-awareness and cache-sensitibity awareness. It can optimize resource allocation while being HPC-aware by applying multi-dimensional on-line bin packing(MDOBP) heuristics while ensuring that cross-application interference is kept within bounds. They designed and implemented the technique using OpenStack Nova [15] and compared it to dedicated execution without their technique. The experiment results show that it can improve the performance by up to 45 % compared to default scheduling method. However they only considered the application-characteristics which is HPC or Non-HPC in terms of cache.

For a given application, these papers analyzed its resource usage, cpu, memory and i/o, etc. and proposed a scheduling technique to optimize its performance in a single environment (i.e., homogeneous infrastructure).

2.2 Heterogeneous infrastructure

We review other approaches for building solutions to facilitate joint usage of computing resources, originating from different types of distributed computing infrastructure and addressing the need of simultaneously harnessing different computing infrastructures.

Moca et al. [13] proposed a scheduling method in distributed computing infrastructures like Grids and Clouds by suggesting a multi-criteria scheduling strategy based on the Promethee algorithm. It fully implemented the Promethee scheduler and recreated an hybrid DCI environment including Internet Desktop Grid, Cloud and Best Effort Grid based on real failure traces.

Mateescu et al. [12] suggested a complex and innovative architecture for combining the benefits of the HPC, Grid and Cloud technologies. In [12], they aim at combining the best attributes of each technology, proposing a model for how scientific workloads can be managed in such a hybrid computing environment.

2.3 Adaptive mechanism

Chang et al. [7] dealt with the adaptive scoring job scheduling algorithm to schedule compute-intensive and data intensive tasks. The Job scores are adaptably computed by selecting the most appropriate resource. The tasks are assigned to the resources according to the cluster score. Local and global updates are performed to get the newest status of resources in the grid environment.

Berman et al. [3] introduced the resource selection problem with the AppLeS project, which provides an adaptive capability to grid environment and considers a novel methodology for adaptive scheduling by allowing the deployment of adaptive distributed applications. In [3], they implemented

this adaptive scheme with two main objectives; Studying adaptive scheduling for grid computing and applying these results to some applications to verify their approach.

The work [4] presented an approach to improve the process of the grid resource selection through a self-adaptive capability to grid applications they designed. It can deal with the dynamic environmental conditions using artificial intelligence algorithm.

3 Proposed algorithm

3.1 Service architecture model

In this section, we begin with the description of the proposed method by discussing the service framework we have designed. Figure 1 illustrates an extended service architecture model for the proposed approach. It is composed of three classes and four major service modules, which effectively control a bunch of tasks through a middleware layer in different computing infrastructures.

The undermost layer is *System Layer*, which could be characterized as varied computing natures, such as Supercomputer, Cluster, Grid, and Cloud, etc.

Those infrastructures contained in the system layer are handled by *Middleware Layer*, which aims to harness the computing resources at best. The system can be either of middleware solutions, like Globus toolkit [8], HTCondor [9], HTCaaS [16], and so on. The adopted middleware solution needs additional features for it to effectively support several services of the topmost class *Service Layer*.

Service Layer consists of four services that are Job Profiling, Monitoring, Weights Adjustment, and Resource Scaling services, where the latter two services belong to the Adaptive module.

Job profiling service is in charge of three roles described as follows. This service is mainly responsible of identifying the type of applications (or `jobtype`). Then it decides and calculates the initial distribution weight value for each of the identified `jobtype` using accumulated job records. Finally, it archives and manages the records of tasks once a job is finished.

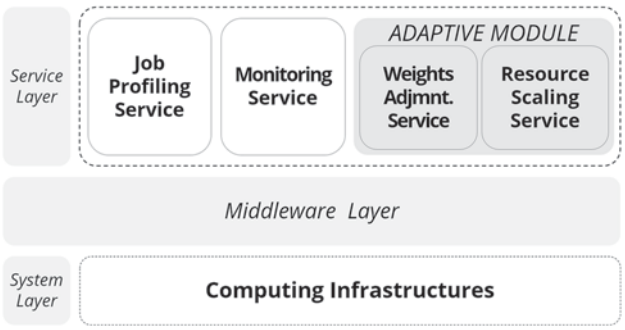

Fig. 1 Service architecture model

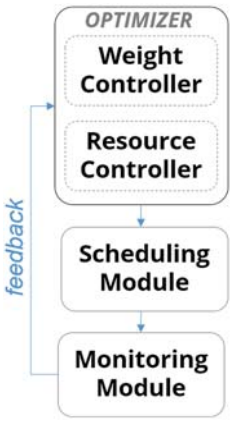

Fig. 2 Abstract flow of the major modules

Monitoring service examines and collects all metadata about jobs, workload, and resources. The monitoring service continuously observes overall workloads of the system, whilst taking into account both dynamically changing workload status and application type. This enables the system to improve utility and capacity of computing resources. The monitored information is sent to the '*Adaptive Module*' and used to control the degree of resources as well as the weight.

Adaptive module contains two services:*Weights Adjustment* and *Resource Scaling*. *Weights Adjustment service* regulates weight values depending on the feedback from *Monitoring Service*. *Resource Scaling service* mainly has in charge the control of a degree for over-provisioning metric(*i.e.*, `vCPU /pCPU` ratio). It is adaptably controlled depending on the error rate monitored by system.

Figure 2 shows the abstract flows of the major modules based on the aforementioned model. The optimizer, that has one dedicated controller for both weight and resource scaling, is exploited to balance system by adjusting distribution rates and degree of over-provisioning. When the scheduling module is carried out, it utilizes the optimized values and let the monitoring module watch the status in pertinent aspects to the above two controllers. The feedback results are forwarded to the optimizer so as to tune the weights and the degree of over-provisioning.

3.2 Proposed algorithm

In the following section, we will introduce the detailed procedure of our adaptive job scheduling optimization. The whole processes of the optimization are as follows.

Step 1: For each metajob M in which multiple subjobs are contained, system identifies two properties that are job type and the count of tasks.

Table 1 Notations

Notaion	Property	Description
$\mathbb{M}$	$\{ j_0, \ldots, j_m \}$	A set of metajobs
$M[id]$	$< jobtype$ **T**, $cnt(tasks)$ **N** $>$	A metajob
$W_{j \times r}$	j: the num. of jobtypes	A matrix for
	r: the num. of resources	a group of weight
	each row is a set of	sets
	weights for specific *jobtype*	
w_{jr}	r: *id* of Job type	A weight for
	r: *id* of resources	resource distribu-
	*: all elements	tion
R_r	r : *id* of Resource	A resource
$r_{v/p}$	$0 < r_{v/p} \leq C$	Ratio of *vCPU* to
	C : Maximum capacity	*pCPU*
ER	$ER \geq 0$, $ER \in \mathbb{Q}$	An error rate
	$\mathbb{Q}$ is rational number	
λ	$\lambda \geq 0$, $\lambda \in \mathbb{Q}$	Threshold of workloads
ε	$\varepsilon \geq 0$, $\varepsilon \in \mathbb{Q}$	Threshold of error rates
Δ	$\Delta \in \mathbb{Q}$	Adjustment value for weights
P	$P \in \{ LOAD, ERROR \}$	Policies of adaptive weight adjustment

Step 2: The *system* takes its distribution weights (w_{j*}) by identifying the application type (*i.e.,* job type), and allocates tasks into resource queues regarding their weights.

Step 3: It figures the number of tasks to be submitted for each resource through $w_{*r} * cnt(tasks)$ and counts the number of resources where its quantity of workload exceeds the threshold system made. After then, the system controls weight values for all resources in accordance with its workload.

Step 4: Meanwhile, resource monitor module periodically watches error rate *ER*. If $r_{v/p}$ is changed (where *policy* is *ERROR*), then the system also regulates weights properly.

The below sections are about the algorithms which are adopted in the aforementioned steps. The key notations that the algorithms employs are listed in Table 1.

3.2.1 Job scheduling optimization algorithms

Step 1 and *Step 2* correspond to following Algorithms 1 and 2.

Once a user submit a Metajob, it is indicated as a Metajob data structure (line 1) where the job type could be defined as various forms such as *cpu* or *i/o-intensive*, and so on (line 1). Regarding the jobtype ***T*** the user submitted, the weight set is decided so that it can dispatch tasks in the rate of weights it is given (lines 1–1). Since we have defined a group of weights as j by r Matrix W where j is the count of jobtypes and r is the number of resources, the proper row is returned according to the jobtype (line 1). By using the returned weight set, the system allocates resources to tasks (line 1) and here calls Algorithm 2, which describes the assigning tasks into resources in keeping with the weight values. At this point, the system figures the number of tasks which will be dispatched to each resource ($w_{Tr} * cnt(N)$) and sends it to weight controller module in order to be used for overload measurement. After the submission, the system controls weights W by sending the policy '*LOAD*', which will be handled in the following subsection (line 1).

Algorithm 1 Job Scheduling Optimization Algorithm

Require: a set of MetaJob $\mathbb{M}$
1: **while** $\mathbb{M}$ **do**
2: MetaJob M = ($jobtype$ T, $cnt(tasks)$ N)
3: //j = 0, 1, ..., T, ..., $n-1$, $\jmath_j$ = { cpu, i/o, mem, ... }
4: **Switch** (T)
5: **case** j**:**
6: $w_{T*} \leftarrow W$.getWeightSet(j);
7: **default:**
8: T $\leftarrow$ *detectJobType(M)*;
9: $w_{T*} \leftarrow W$.getWeightSet(T);
10: **end switch**
11: ***SubmitJobstoResources***(w_{T*});
12: $W \leftarrow$ ***WeightsAdjmt***($LOAD$);
13: **end while**

Algorithm 2 Job Submission Algorithm

Require: a set of Weights $W_{1 \times r}$
1: i : resource ID
2: **for all** $w \in W_{1 \times r}$ **do**
3: **if** $i == 0$ **then**
4: **M** [0 : (N * w_{ji}) -1] $\leftarrow R_i$
5: **else**
6: **M** [N * $\sum_{t=0}^{i-1} w_{jt}$: (N * $\sum_{t=0}^{i} w_{jt}$) -1] $\leftarrow R_i$
7: **end if**
8: **end for**

3.2.2 Adaptive weights adjustment algorithm

Algorithm 3 is a procedure which adjusts the weight values considering workload status. It is divided in two main parts. One part is that policy is in '*LOAD*'(line 3–3), where the monitoring module observes the quantity of workload for each resource, and regulates the weights W. Initially, the system measures the workload threshold (λ, line 3), as shown on the following Eq. 1,

$$threshold(m) = \begin{cases} 0 & \text{if} \quad n = 0 \\ \dfrac{\sum_{i=n-m+1}^{n} M[i].N}{m} & \text{if} \quad n > 0 \end{cases} \tag{1}$$

Algorithm 3 Adaptive Weights Adjustment

Require: Policy P
1: **Switch** (P)
2: **case** $LOAD$**:**
3: $\lambda \leftarrow$ ***threshold***(m);
4: **if** $cnt(R_{load>\lambda}) == 0$ **then**
5: Stay_Current_weight( );
6: **else if** $cnt(R_{load>\lambda}) == cnt(\text{Total } R)$ **then**
7: *InitializeWeights(W)*;
8: **else if** $cnt(R_{load>\lambda}) > 0$ **then**
9: $\mathbb{X} \leftarrow$ set of $R_{load>\lambda}$
10: $W_{j\times r} \leftarrow$ matrix of w ; // $w_{(0,0)}, ..., w_{(j-1,r-1)}$
11: **for all** $x \in \mathbb{X}$ **do**
12: $w_{*x} \leftarrow w_{*x}$ - Δ
13: **end for**
14: **for all** $y \in \mathbb{X}^c$ **do**
15: $w_{*y} \leftarrow w_{*y} + \dfrac{\Delta \cdot \mathbb{X}.size}{\mathbb{X}^c.size}$
16: **end for**
17: **end if**
18: **case** $ERROR$**:**
19: $\mathbb{C} \leftarrow$ set of $R_{i\in cloud}$;
20: $W_{j\times r} \leftarrow$ array of w ; // $w_{(0,0)}, ..., w_{(j-1,r-1)}$
21: **for all** $x \in \mathbb{C}$ **do**
22: $w_{*x} \leftarrow w_{*x} + \Delta$
23: **end for**
24: **for all** $y \in \mathbb{C}^c$ **do**
25: $w_{*y} \leftarrow w_{*y}$ - $\dfrac{\Delta \cdot \mathbb{C}.size}{\mathbb{C}^c.size}$
26: **end for**
27: **end switch**
Ensure: $W_{updated} \leftarrow setWeightSet(W)$

where m is the number of recent metajobs and n is the latest *id* of the metajob in the system. This λ keeps reflecting the most recent status of workload by analyzing the freshest records of jobs. After deciding λ, the system counts the resources($R_{load>\lambda}$), where workload exceeds λ. It keeps the current weight values (line 3–3) if $R_{load>\lambda}$ does not exists, while the weight set will be initialized if the count is identical to the R number of total resources (line 3–3). It is because we consider the present weight values to be untrustworthy. In case the count becomes higher than zero (lines 3–3), it decreases the weight values of all the resources where workload exceeds the threshold. In the meantime, it also increases them for the rest of the resources. Δ, which is the amount to be adjusted, can vary depending on the the number of resources.

The second part concerns the '*ERROR*' policy's state (lines 3–3) and is triggered by *Resource Scaling Service*, which conducts adaptive virtual machine scaling and thus causes weight adjustments. That is, if the *policy* is set as *ERROR*, it is led by the fluctuation of over-provisioning ratio, and thus weight adjustment is required so that system can maintain the overall balance. In this method, the adjustment is carried out only when the ratio of scaling increases. The system elevates the weights of the resources where the scaling up is made (line 3–3), while reducing weight values for the other resources (line 3–3). The above weights tuning are invoked for the weights of the entire application types.

Algorithm 4 Adaptive Resources Scaling

Require: ER, $r_{v/p}$
1: **while** *True* **do**
2: **if** $ER > \varepsilon$ **&&** $r_{v/p} > 1$ **then**
3: $r_{v/p}--$;
4: **else**
5: $r_{v/p} \leftarrow$ **MIN[** $r_{v/p}++, MAX(r_{v/p})$ **]**
6: **if** $r_{v/p}$ *increases* **then**
7: $W \leftarrow$ ***WeightsAdjmt***$(ERROR)$;
8: **end if**
9: **end if**
10: Return $r_{v/p}$;
11: sleep(*Interval*) ;
12: **end while**
Ensure: $r_{v/p}$

3.2.3 Adaptive resources scaling algorithm

Algorithm 4 corresponds to the *Step 4* we explained earlier. It is responsible of scaling the resources(which are virtual machines) up & down, relying on the error rates they incur. Here, the most essential parameter is $r_{v/p}$, which is obtained from Eq. 2,

$$r_{v/p} = \frac{vCPU}{pCPU}, \quad 0 < r_{v/p} \leq C \tag{2}$$

where $r_{v/p}$ denotes the number of virtual cores ($vCPU$) mapping on to a physical core ($pCPU$). A constant C is the maximum capacity of virtual cores that cloud solution offers.

ER refers to the rate of $Error\ Occurrence$. The errors may be arose when virtual machine is in the unreliable or failure status, or has more workloads than it can accept during the run-time. Moreover, their amount can be different depending on the type of jobs (applications) that are running on the virtual machine. If the error rate is higher than ε and the ratio of $vCPU$ to $pCPU$ is not the minimum available capacity, the system diminishes the ratio by 1 (lines 4–4). In contrast, the ratio would rise until C (lines 4–4) and *Weight Adjustment* would be carried out by calling the policy '*ERROR*' (lines 4–4). This adaptive resource scaling process is periodically performed in the cycle of *Interval*.

3.3 Example

To understand the proposed approach, let us consider by presenting a simple example where collective jobs with three kinds of job types (*e.g.,* `cpu, i/o, mem.`) are submitted into two kinds of resources that are cluster and cloud. Suppose that an initial weight array $\boldsymbol{W}$ which is three by two

array according to the above conditions is described as the following Eq. 3,

$$\boldsymbol{W}_{init} = \begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{bmatrix} = \begin{bmatrix} 0.55 & 0.45 \\ 0.70 & 0.30 \\ 0.55 & 0.45 \end{bmatrix} \tag{3}$$

where the elements included in the same row have same type of job, and the ones that are contained in equal column are the weights for the identical physical resource. In this example, from the first to third rows (w_{0*}, w_{1*}, w_{2*}) is a set of weights when job type is `cpu`, `i/o` or `mem`-intensive, respectively. The columns(w_{*0}, w_{*1}) correspond to the resources that are *cluster, cloud*, respectively.

Consider that users submit three number of metajobs sequentially and that system have to distribute the tasks onto the existing resources. According to the steps we explained before, the system goes in to identify type of metajobs and the number of tasks each metajob has. Let the *id*s of all metajobs be consecutive numbers from `1` to `3`. Let each metajob have the following properties: $M[1]$ =<1,30000>, $M[2]$ = <2,150000>, $M[3]$ =<0,70000>. The system takes one row of W that corresponds to the jobtype to be submitted in order to allocate tasks into resource queues. The system would adopt w_{1*} for the first metajob and, dispatch tasks onto cluster and cloud at a rate of .55 and .45, respectively. For the second metajob, it would follow a rate of .70 and .30. Assumed that all metajobs are submitted, and then the number of tasks queueing in the resources would be 169 and 81 M, respectively. If new metajob is arrived to the system at this point of time, the weight value will be controlled, since $\boldsymbol{\lambda}$ is 83333.3333 (see Eq. 1), thus the overload in cluster is occurred as the condition that $cnt(R_{load>\lambda}) > 0$ (Algorithm 3, line 3) is true. $\mathbb{X}$ has one element that is `0` (which is *id* of cluster) and $\mathbb{X}^c$ would be 1 (which is cloud). If we suppose that system set Δ, which is adjustment value for weights, as `.1`, all weights included in the `0`th column(w_{*0}) decrease by Δ and every values of the `1`th column(w_{*1}) would be augmented by $\frac{\Delta \cdot \mathbb{X}.size}{\mathbb{X}^c.size}$ that is `.1`, since both size of $\mathbb{X}$ and $\mathbb{X}^c$ is `1`. Hence, the updated W would result as below.

$$\boldsymbol{W}_{updated} = \begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{bmatrix} = \begin{bmatrix} 0.45 & 0.55 \\ 0.60 & 0.40 \\ 0.45 & 0.55 \end{bmatrix} \tag{4}$$

In the same manner, W would be updated until that system is in stable status. That is, it adapts to current status of dynamic environments and contribute to leading to overall system steady.

4 Evaluation

We evaluate our optimization method through the next experiments. We first start by introducing the details of experimental setup, and go on to show our assessments using four practical applications and realistic workload models in order to verify how better performance we can present.

4.1 Experimental environments

HTCaaS: HTCaaS [16] aims to facilitate exploring large-scale HTC or MTC problems for various computing resources such as Supercomputers, Grids, Clusters and Clouds. It can hide diversity of integrating different resources from users, and allow users to efficiently submit a large number of tasks at once. It employs a *pilot-based* multi-level scheduling mechanism which supports the decoupling of resource allocation from resource binding. The general agent itself is a regular batch job which is submitted by HTCaaS system and is assigned into the resources by the local batch scheduler. Then it performs '*pulling and executing*' sub-jobs as well as coordinating the launch and monitoring processes.

OpenStack module was recently supplemented to HTCaaS with the purpose of ameliorating the system, as shown on Fig. 3. In this module, each virtual machine corresponds to an agent (pilot-job). Once a virtual machine is launched, the agent which is built in the VM starts to *pull & launch* the job it is given. The algorithms in this paper are mostly working with *Agent Manager* module.

Computing environment: The computing environment for experiments is organized with local cluster and private cloud resources, where those are integrated with *HTCaaS*. The local cluster uses a Sun Grid Engine (SGE) [10], and it consists of 1 master and 2 slave nodes. They are three Intel(R) Xeon(R) CPU E5630@2.53GHz with 8 cores & 24GB RAM as master and slaves. The cloud infrastructure adopts Openstack that consists of 1 master and 3 slaves. They are Intel(R) Core(TM) i7-4930K @3.40 GHz with 12 cores & 4 GB RAM for master, and three of Intel(R) Core(TM) i7 950@3.07 GHz for Slaves, respectively. Each virtual machine is generated as the distinctive form on the condition of the over-provisioning rate, $r_{v/p}$, but is basically formulated using Ubuntu 12.04 Server image (the default capacity is 1 GB ram and 1 vCPU).

4.2 Target application

In this paper, we are aiming for high-throughput computing (HTC) and many task computing (MTC) application, which generally have millions or billions of tasks to be processed, each with relatively short execution time. A wide range of scientific domains, such as High-energy Physics, Pharmaceutics, Chemistry, etc., belong in those application fields.

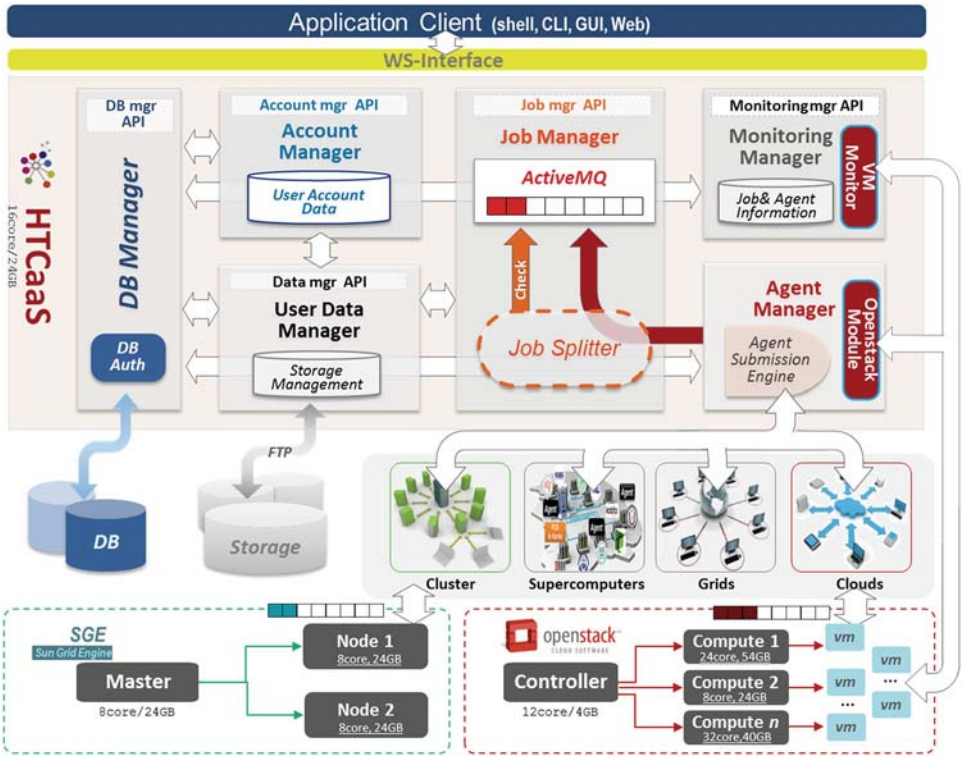

Fig. 3 Experimental architecture using HTCaaS

In the experiments, we adopted four different kinds of practical HTC and MTC applications, which are AutoDock [1], Montage [14], Pythia [17] and Condensed Matter Physics (denoted as C.M.S in this paper) simulation [2]. These applications can be characterized as either CPU-intensive or I/O-intensive.

AutoDock [1] is a molecular modeling simulation software to predict how small molecules, such as substrates or drug candidates, bind to receptor of known 3D structure. For our experiments, we select autodock3 which is used for performing the docking of ligands to a set of target proteins in order to discover potential new drugs for several serious diseases such as SARS and Malaria, from a wide range of scientific computing applications to represent CPU-intensive jobs having small input data(6.1MB)/output data(3.0KB) files with considerable size of memory usage, as shown on the Table 2. Autodock has 0.7 % memory utilization while keep 100.0 % CPU utilization.

Montage [14] is an astronomical image mosaic engine for creating composite Flexible image transport system (FITS) mosaics using multiple astronomical images. In our experiments, we use a set of five data cubes, released as part of the Galactic AreciboL-band Feed Array HI(GALFA-HI) survey [19], into a mosaic. It is deemed an data-intensive application having not only relatively lots of input (5.5 GB) and output (3.2 GB) data files but also intermediate files. Montage has average memory utilization and CPU utilization, 90.8 and 9.1 %, respectively.

Pythia [17] is a suite of standard tool for Monte Carlo(MC) simulations, for events generation in high-energy physics domain. It is mainly a CPU-intensive application with small size of input (8.0 MB)/output (122 KB) files. Average CPU utilization of pythia is 10× higher than memory utilization.

The last application we introduce is an in-house code based on Monte Carlo simulation in Condensed matter physics for the formation of a monopoles and anti-monopoles lattice in short-period chiral magnets [2]. This simulation is an I/O-intensive application having large size of input (2.68 GB) and output (19.6 GB) files, as described in the Table 2. C.M.S gets about 80 % average memory utilization but it keeps 40 % average CPU utilization.

4.3 Workload model

Figure 4 depicts four types of realistic workload patterns, which are *Stable*, *Growing*, *Bursting*, and *On-and-Off*, respectively. *Stable* pattern (Fig. 4a) is a baseline for comparison with other patterns. It has stable trend without any bursting and is almost impossible to be occurred in real computing environment. *Bursting* pattern (Fig. 4b) represents the case having sporadic spikes. As the workload spikes intermittently, it tends to create intense overloads on certain resource(s). *Growing* form (Fig. 4c) can be the scenario that high throughput applications starting with a few parameters but then adding many arguments. *On-and-Off* workloads (Fig. 4d) have spikes at regular intervals, and are typically

Table 2 Properties of applications

Application	Input	Output	Avg. Mem. (%)	Avg.CPU util (%)
AutoDock	6.1 MB	3.0 KB	0.7	100.0
Montage	5.5 GB	3.2 GB	90.8	9.1
Pythia	8.0 MB	122 KB	10	100.0
C.M.S	2.68 GB	19.6 GB	80	40

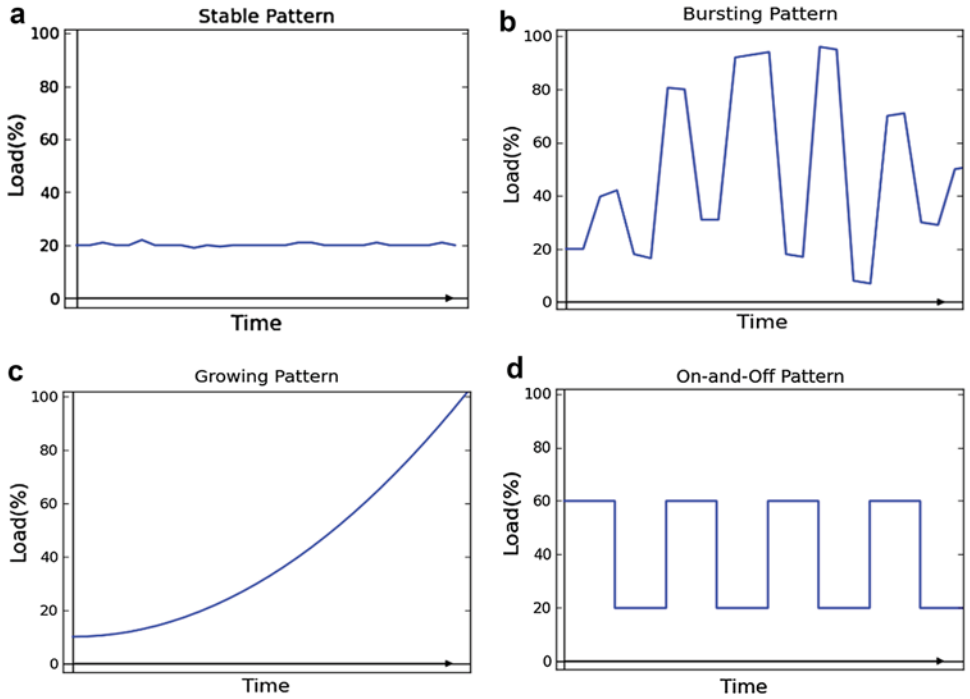

Fig. 4 Workload patterns. **a** Stable, **b** bursting, **c** growing and **d** on-and-off

seen with applications that are needed for a short period of time and later shut off. As an example, it can be one-time large-scale job submissions which will be taken by thousands of users in one day.

The latter three patterns may represent the general environment for large-scale computing service having unexpected or irregular peaks for several times.

4.4 Experiments and results

We run a series of experiments using the addressed mentioned applications and workload patterns. We carry out a total of one-hundred thousand (10^5) number of tasks (subjobs) three times for each scenario, and an average measurement of the results is adopted for performance evaluation.

There exist four scenarios in terms of workload patterns, where each scenario is conducted under the condition of various application combination which are *CPU-intensive*, *I/O-intensive*, *CPU+I/O-intensive*.

For performance metrics that are used for comparisons, we take into account *throughput*, *queueing time*, and *resource utilization*. In these experiments, we compare the proposed scheduling optimization algorithm to a `Core-based` policy as a baseline that system distributes tasks in proportions of the number of cores each resource has. Our main findings are summarized as below.

Throughput should be considered as a primary performance metric, since we are aiming for HTC and MTC applications. Figure 5 illustrates the results of the proposed method and core-base one for all workload patterns. Firstly, comparing the suggested method with the baseline relative to *Stable* workload pattern, the improvements are 30.4, 6.3, and 30.2 % respectively as shown in Fig. 5a. In the second scenario that is *Bursting* pattern, it has better enhancements which are 52.3, 34 and 63.5 % respectively (see Fig. 5b), where the results indicate our method can cause better performance on multiple applications environment by offering application-aware weights with its optimization. In the *Growing* pattern, it results in maximum improvements that are 72, 50.4, and 77.7 % correspondingly (Fig. 5c). Besides, in *On-and-Off* pattern (Fig. 5d), there exist 38.9, 27.4 and 38.4 % of improvements, correspondingly, compared to `Core-based` policy. To conclude, these throughput outcomes eventually point out that our method improves overall

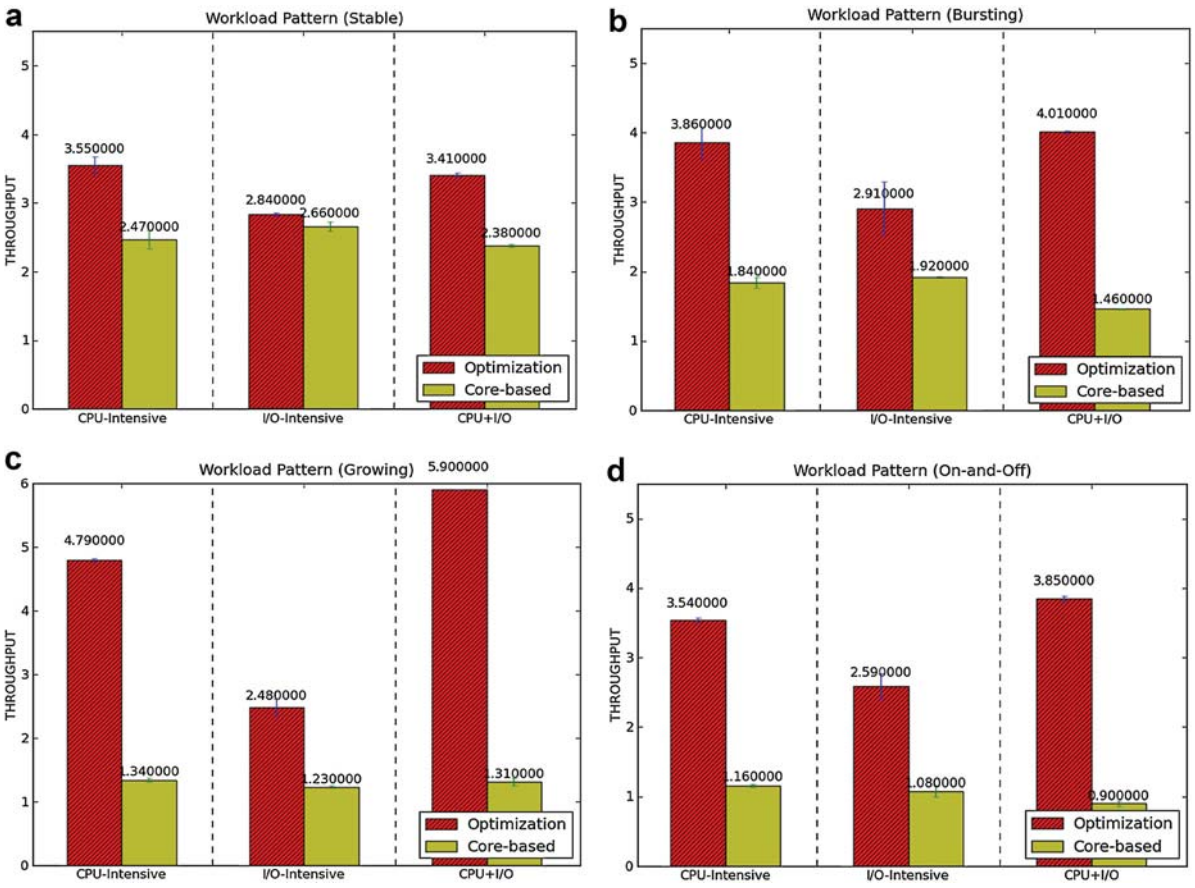

Fig. 5 Throughput results. **a** Stable, **b** bursting, **c** growing and **d** on-and-off

throughput of the system, and particularly more effective on the environment having multiple-applications and dynamically changing or irregular workload patterns.

Queueing time refers to the latency time in each resource queue, and is to show how well the suggested method can control the weight values by detecting as well as leveraging current workload status. Figure 6 describes the outputs in terms of average queueing time, where compared to `Core-based` policy according to the four different trace patterns. The graph is normalized with the baseline policy. Comparing the our method to core-based policy with respect to CPU-intensive, I/O-intensive, and combination application, our approach can speed-up about 26, 17, and 24 % in the *Stable* pattern case and enhance by 32, 21, and 51 % in the bursting pattern through reducing latency time in the queue. In addition, the highest decline is made in *Growing* pattern, especially in the condition of combination of two types where 71.3 % of enhancement is observed. In a case of *On-and-Off* scenario, similar outcomes to the results of *Stable* scenario are made for all conditions, as resulting 21, 18, and 27.4 %, respectively. In the experiments, we could observe relatively lower performances were achieved at first when workload are flat and sinusoidal curve forms that correspond to *Stable* and *On-and-Off* pattern model. The reason of that seems to be that their flat or periodic pattern had hardly effects on optimizing weight values. Moreover, the results of the last scenario shows that more sophisticated methodology of setting Δ and *Interval*(used in *Resource controller*) are required, since these variables are capable of affecting the performance directly.

All in all, our method can reduce the average waiting time of jobs relative to baseline for the entire workload patterns.

System Utilization is measurement of the average physical cores' usage for all existing resources. This metric is to verify how proper our 'adaptive resource scaling' strategy can effectively utilize entire resources through detecting errors or failures that happened in run-time. Figure 7 depicts the average incidence that virtual machines made when over-provisioning in run-time. It is more happened on VM where CPU-intensive application ran than I/O-intensive. It causes re-execute jobs and bad influence on overall throughput. However, it is able to overcome performance degradation by conducting 'Adaptive Resource Scaling' cycle. Figure 8 presents the plot of resource utilization. The observed average resource utilization results of three conditions with adaptive scaling are 87, 74, and 91 %, respectively, which

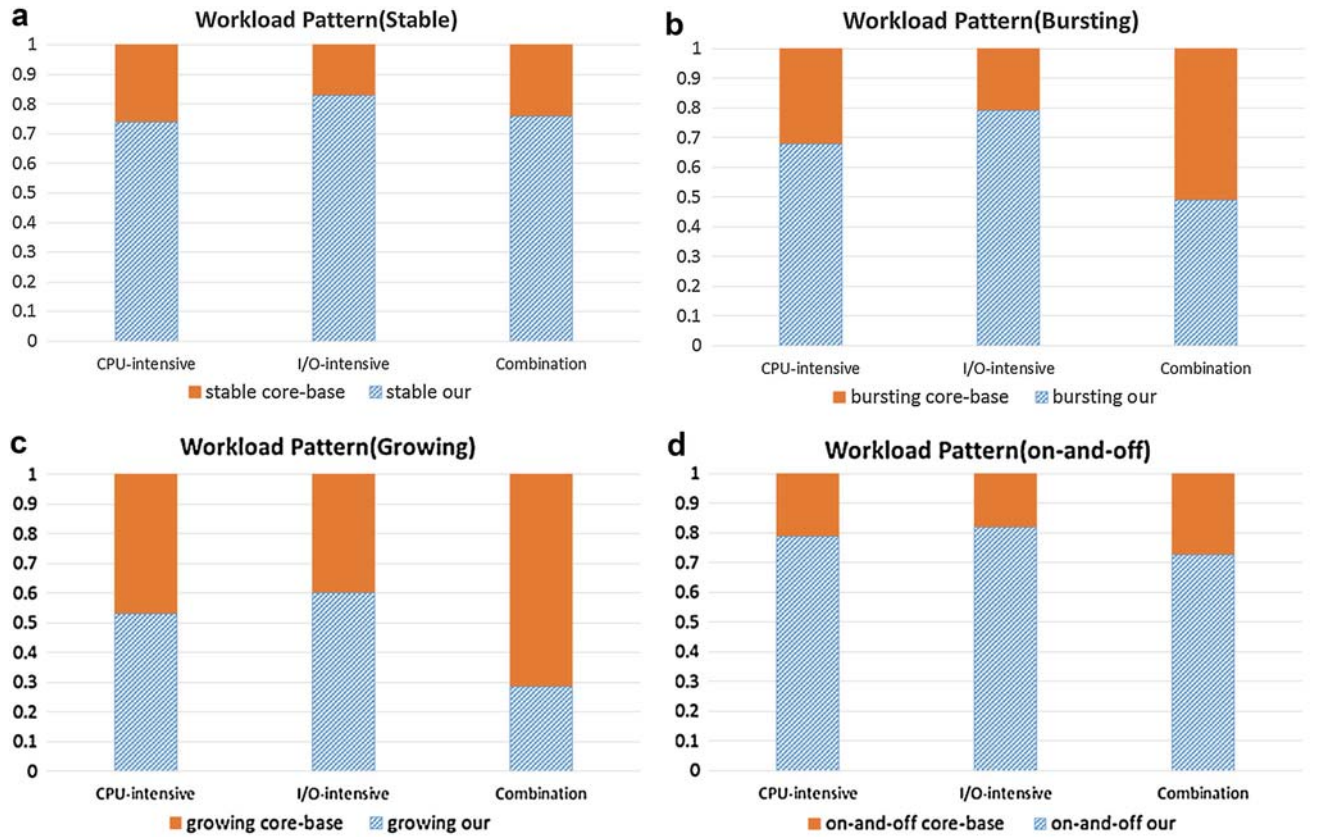

Fig. 6 Results of queuing time respect. **a** Stable, **b** bursting, **c** growing and **d** on-and-off

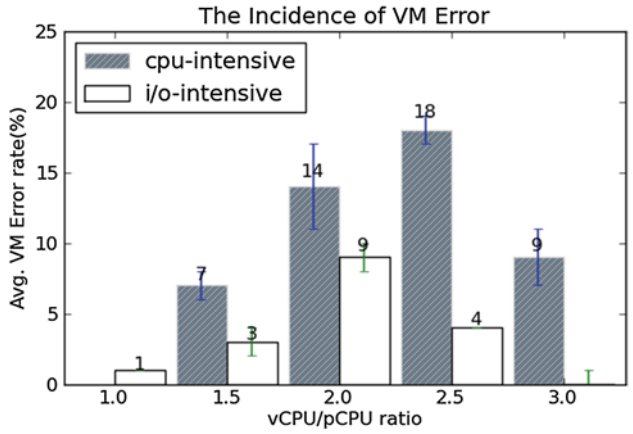

Fig. 7 The incidence of VM Error.eps

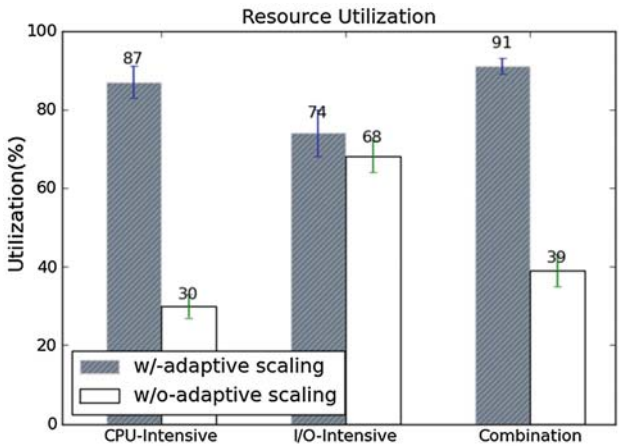

Fig. 8 Results of the resource utilization

are noteworthily improved compared to baseline particular in two conditions that are CPU-intensive and Combination. This is because the error-aware resource scaling algorithm in the our method is capable of utilizing resources better, as contributing to decreasing error occurrences.

5 Conclusion

We have presented application-aware job scheduling optimization method, which uses a weighed set meaning that proportion of tasks to be dispatched for resources in large-scale scientific applications in heterogeneous infrastructures.

The proposed method consists of three main algorithms that are *Job scheduling optimization*, *Adaptive weights adjustment*, and *Adaptive resource scaling* algorithms.

We also design a service architecture model that delineates how this approach can apply and operate in general. We have implemented and demonstrated the algorithms along with our testbed. Empirical experiment is also conducted to lead to an optimal distribution weights for jobs from different applications in heterogeneous infrastructures. The experimental assessment exploiting the diverse realistic workload pattern have proved that, when compared to the `Core-based` manner which allocates tasks in proportions of each resource's number of cores, our optimization approach can yield 62 , 43, and 38 % better performance in terms of average throughput, queueing time, and utilization, respectively. of the total resources in diverse infrastructure environments by harnessing our adaptive module. Future work will include further experiments with other widely scheduling models as well as widely known studies. Furthermore, study on identifying application types in automatic and ingenious way will be planned in the future. Finally, we would like to extend this research by adding an profiling service for diverse applications aiming for HPC as well as HTC.

Acknowledgments This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Plannig (2015M 3C 4A7065646r) and also supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. R0190-15-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development).

References

1. Autodock. http://autodock.scripps.edu/
2. Baumgärtner, A., Burkitt, A., Ceperley, D., De Raedt, H., Ferrenberg, A., Heermann, D., Herrmann, H., Landau, D., Levesque, D., von der Linden, W., et al.: The Monte Carlo method in condensed matter physics, vol. 71. Springer, Berlin (2012)
3. Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., et al.: Adaptive computing on the grid using apples. IEEE Trans. Parallel Distrib. Syst. **14**(4), 369–382 (2003)
4. Botón-Fernández, M., Vega-Rodríguez, M.A., Prieto Castrillo, F.: Intelligent self-adaptive resources selection for grid applications. Concurr. Comput.: Pract. Exp. **27**, 3539–3560 (2014)
5. Bryk, P., Maciej, M., Juve, G., Deelman, E.: Storage-aware algorithms for scheduling of workflow ensembles in clouds. Grid Comput **14**, 359–378 (2015)
6. Bu, X., Rao, J., Xu, C.: Interference and locality-aware task scheduling for mapreduce application in virtual clusters. In: HPDC'13 Proceedings of the 22nd international symposium on High-Performance parallel and distributed computing, pp. 227–238 (2013)
7. Chang, R.S., Lin, C.Y., Lin, C.F.: An adaptive scoring job scheduling algorithm for grid computing. Inf. Sci. **207**, 79–89 (2012). doi:10.1016/j.ins.2012.04.019. http://www.sciencedirect.com/science/article/pii/S0020025512002836
8. Foster, I., Kesselman, C.: Globus: a metacomputing infrastructure toolkit. Int. J. High Perform. Comput. Appl. **11**(2), 115–128 (1997)
9. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-g: a computation management agent for multi-institutional grids. Clust. Comput. **5**(3), 237–246 (2002)
10. Gentzsch, W.: Sun grid engine: towards creating a compute power grid. In: First IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE Proceedings, pp. 35–36 (2001)
11. Gupta, A., Kale, L.V., Milojicic, D., Faraboschi, P., Balle, S.M.: Hpc-aware vm placement in infrastructure clouds. In: Cloud Engineering (IC2E), 2013 IEEE International Conference on pp. 11–20 (2013)
12. Mateescu, G., Gentzsch, W., Ribbens, C.J.: Hybrid computingG where hpc meets grid and cloud computing. Future Gener. Comput. Syst. **27**(5), 440–453 (2011)
13. Moca, M., Litan, C., Silaghi, G.C., Fedak, G.: Multi-criteria and satisfaction oriented scheduling for hybrid distributed computing infrastructures. Future Gener. Comput. Syst. **55**, 428–443 (2015)
14. Montage. http://montage.ipac.caltech.edu/
15. Openstack Nova. https://wiki.openstack.org/wiki/Nova
16. Rho, S., Kim, S., Kim, S., Kim, S., Kim, J.S., Hwang, S.: Htcaas: a large-scale high-throughput computing by leveraging grids, supercomputers and cloud. In: High Performance Computing, Networking, Storage and Analysis (SCC), IEEE 2012 SC Companion, pp. 1341–1342 (2012)
17. Sjstrand, T., Mrenna, S., Pythia, P.S.: 6.4 physics and manual 050262006
18. Thaha, A.F., Singh, M., Amin, A.H., Ahmad, N.M., Kannan, S.: Hadoop in openstack: data-location-aware cluster provisioning. In: Fourth World Congress on Information and Communication Technologies (WICT) 2014, pp. 296–301 (2014)
19. The Galfa-hi Survey: Data release 1. Astrophys. J. Suppl. **194**(2), 13 (2011)
20. Yang, L., Dai, Y.: Application scheduling in cloud computing environment with the consideration of performance interference. J. Commun. **10**(8), 603–609 (2015). doi:10.12720/jcm.10.8.603-609

Seoyoung Kim is a researcher in division of Supercomputing 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 KISTI from 2012. Her research interests include big data processing, meta-scheduling in distributed computing systems and many-core processing system.

Jieun Choi is currently a Master's 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.

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.