iSDF: an Integrated Software-defined Computing Framework for Scientific Experiments*

Seoyoung Kim, Julim Ahn, Heewon Kim,
Yoonhee Kim [0]
Dept. of Computer Science
Sookmyung Women's University
Seoul, Korea
{sssyyy77, julim8990, gmldnjs0610, yulan}@sm.ac.kr
[0] Corresponding author

Jieun Choi
Dept. of Supercomputing
Korea Institute of Science and Technology
Information (KISTI)
Daejeon, Korea
jieun1205@kisti.re.kr

***Abstract*— Cloud computing platform is becoming one of the most desired environments for the emerging applications as well as traditional application domains. In this regards, there have been lots of efforts to develop and redesign the framework to deliver solutions that meet a variety of requirements including resilience, performance, and agility.**

This paper presents a design and implementation of *iSDF*, an integrated software-defined computing framework for scientific experiments. The iSDF aims at reinforcing traditional cloud-based scheme by supporting application-aware scheduling that avoids resource starvation and contentions for software-defined computing. To demonstrate the effectiveness of the iSDF, we evaluate it by performing several experiments based on three scenarios. The results show that our framework can achieve a 51% increase on resource utilization and average makespan time a 19% improvement compared to the conditions that do not include the policy. Thus, we proved that the framework can contribute to increase utilization for shared resources and to optimize overall makespan time, eventually affecting overall throughput of jobs.

Keywords— Cloud computing; Software-defined computing; Scientific applications; Dynamic resource sharing; Workflow, Anti-interference scheduling

I. INTRODUCTION

With the rapid technological advance in cloud computing, more and more application executions have been migrating to data centers in a variety of fields. Accordingly, applications running over the clouds are getting diverse and their demands are also increasing. Such shifts could lead resource contentions in data centers, which can cause low resource utilization and make the datacenter burdensome to manage.

To address these issues with regard to software-aspect, there has been a lot of efforts to create a new paradigm: the computing environment can be redefined in a software-oriented way, which is referred as software-defined environment (SDE).

Software-defined environment[1] is an abstracted and virtualized IT infrastructures automatically managed by software where software-defined computing(SDC), software-defined network(SDN) and software-defined storage(SDS) exist and eventually compose a software-defined design of infrastructure system: Software-defined infrastructure (SDI).

Among them, Software-defined Computing is the most meaningful for most of application domains running over data centers. It allows mobility and higher resource utilization, as several virtual resources can be assigned together to the identical server, and different requirements for resources can be mitigated by being shared amongst the virtual resources.

On the other hand, scientific applications exhibit distinct properties such as heterogeneous workloads, diverse running patterns and highly dynamic demands for resources, making them still non-trivial to support and run on clouds. To achieve optimized performance as well as improvement of resource utilization for it, it is essential to design its own software-design computing framework.

In this paper, we have designed and implemented *iSDF*, an integrated Software-Defined computing Framework for scientific experiments that offers sophisticated resource allocations over diverse types of resources. The key features of our framework are described as follows:

1) *Design of Software-defined computing framework*
   : we have designed a framework for software-defined computing that enables the management to be decoupled and independent from underlying infrastructures, so that we can elastically regulate them
2) *Application-aware Anti-Interference scheduling*
   : it offers a dynamic fine-grained resource scheduling by detecting different application's needs and minimizing inteferences between concurrent jobs
3) *Workflow supporting*
   : not only a bunch of tasks, it also supports workflow execution and scheduling to ensure its performance
4) *Ease of use environment*
   : users can control this framework with graphical interface(GUI) or application program interface(API) without additional knowledgements of computing parts

* This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP)(NRF-2017R1A2B4005681)

This paper is organized as follows. Section 2 surveys related work. Section 3 details our framework by presenting architecture and execution scenarios. After introducing the scheduling strategy adopted in this framework, we evaluate it in Section 4. Finally, we conclude in Section 5.

II. Related Works

There exists some related technologies to develop the integrated system with scheduling method. In this section, we discuss these researches in detail.

High Throughput Computing-as-a-Service(HTCaaS) [2] is an integrated multi-level scheduling and job submission system. It aims to enable for scientists to carry out large-scale and complex scientific computations by integrating computing resources over heterogeneous infrastructures such as Grids, Clouds and Supercomputers. This system, however, is focused on the large-scale task executions rather than on allocating fine-grained resources. Moreover, it is required to develop an adaptor to connect and expand new infrastructures. Our framework, iSDF is improved further upon the HTCaaS system by redesigning job processing part and complementing resource management parts.

In the aspect of job scheduling and resource allocation research for cloud environment, [3] proposed a controlling system which allocates appropriate resources through monitoring and analyzing current workloads of applications. In this work, a virtual server is operated on a group of physical machines and each server is responsible for particular. However, this study mainly focused on the proper resource management and utilization with less considering task performance.

Other work [4] suggested a scheduling scheme considering deadline as well as power consumption in a cloud computing environment. However, this scheduling has limitations that it is not applicable to scientific application such as workflow because it is a target of Bag of tasks application which does not have dependency between tasks.

The paper [5], which proposed a scheduling scheme for interdependent workflow applications without limiting the scheduling object to the Bag of tasks application, has limitations in the grid environment, not in the cloud environment.

In this paper, we propose a SDC framework offering the application-aware anti-interference scheduling strategy where it schedules tasks according to the needs of various types of resources and weights by retrieving the profiling information containing the application's job characteristics information in the cloud computing environment.

III. An Integrated SDC Framework

We begin the description of the proposed framework by introduting its arcitecture. We then explain how the jobs and computing resources are effectively managed by our framework with *execution scenario*.

A. Architecture

Fig. 1 illustrates the architecture of our integrated SDC framework, *iSDF*. Its components are categorized into three groups according to job or resource aspect and others.

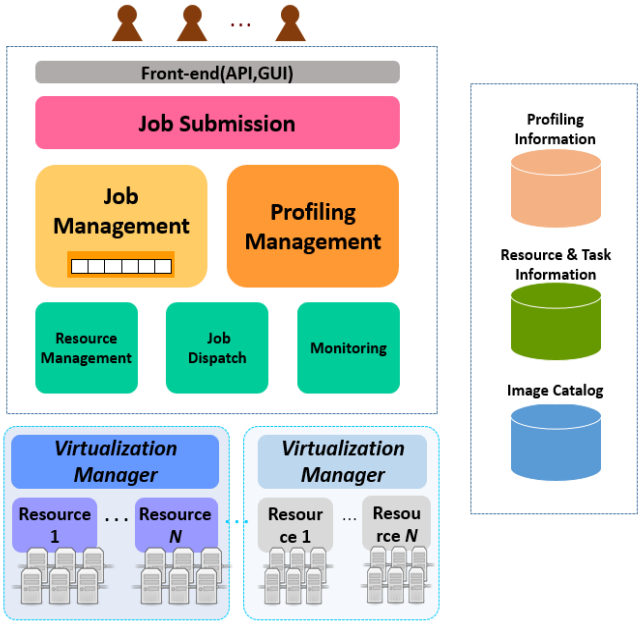

Fig. 1 Architecture of the integrated framework

First group related with job management consists of *Job Submission*, *Job Management* and *Job Dispatch* module.

Job Submission module manages and parses job description files, JSDL, specified by user(s). Job Submission Description Language (JSDL) [6] is an extensible XML specification to describe requirements of computational jobs. JSDL includes job name, file staging (in/output, executable files), commands to execute and arguments with their range. The multiple tasks can be generated according to the range of the arguments by parameter sweeping. This file also can be specified differently depending on the characteristic of the application to run. In case of workflow, for instance, it can include the dependency information and additional file staging steps as well as the basic parameters such as application name, file staging, etc. The module then loads it in a form of the dependency-based job graph.

Job Management module mainly takes care of tasks scheduling and queue management. It manages the entire queue where tasks are categorized by their job id before splitting. In the workflow case, the same procedure is performed after checking its dependency, which is supported by Chronos[7]. One of the main role of this module is making the plan for task scheduling. Here, diverse scheduling policies can be applied to the framework by system administrator. As a default, however, this framework adopts the fine-grained resource allocation method exploiting the application characteristics analyzed from the system which also refers to *Application-aware Anti-interference scheduling* method. This method will be discuss in details later.

Job Dispatch module exams the input data, the execution file, and launches the tasks into the allocated resources (*a.k.a.* virtual machine).

The second category related with resource consists of *Resource Management* and *Monitoring* modules.

Resource Management module principally takes charge of allocation for multiple types of resources and management of virtual machines. This module is based on Mesos [8] which helps the system control multiple computers as one computer and allocate resources in dynamic way. It launches the virtual machines based on the plan made by the *Job Management* module and also regulates the particular resources (*e.g., cpu, mem, disk, etc.*).

Monitoring module collects the status of CPU, Memory, Disk, virtual resources and Job status periodically, and shares them with other modules. The monitored information is stored in the Resource & Task Information DB.

The rest contains only one module: *Profiling Management*. This module manages the simulation history based on the submitted jobs. Whenever each job is completed, meaningful information is extracted in the form of a profile schema, and then its profile information is stored. When the *Job Management* module asks characteristic information, this *Profile Mgmt.* module returns it if it exists, otherwise just keeps the profile as the new application. The job profiles keep being accumulated every time it is executed. The profiling method applied in this framework is based on the algorithm of this paper [9] which proposed the methodology for analyzing and finding the characteristic (*i.e.,* factors) of the jobs through profiling. The factors to be determined from profiling are categorized according to the following two characteristics; Application and Resource. Application factors can be several and input arguments used during the runtime that are likely to directly affect the execution time of the application. Resource factors include the parameters of physical elements that have a significant effect on the duration of the job. This may consist of CPU count, job reliability (rate of the number of all successful tasks), maximum parallel throughput, and the average waiting time (waiting time from submission of a job to beginning of the first task). At this time, such profile information is stored in Profiling Information DB.

B. Execution Scenario

The overall steps of job execution and its processes in i*SDF* are depicted on Fig. 2 and described as follow:

1) A JSDL file specified by user is submitted to the *iSDF* through the front-end, either API(Application Programming Interface) or GUI (Graphic User Interface).
2) The Job Submission module parses the JSDL file and extracts information such as application type, executable file path, parameters, dependency, etc. For each job, the module generates one or multiple tasks (sub-jobs) depending on the range of parameters specified in the file, and then sends them into the queue of Job management module with their affiliation info. (*i.e., job id*).
3) The Job management module asks the job characteristics to the Profile Management module by sending the list of affiliation information for the tasks. The job profile data is used for the upcoming scheduling and allocation step.

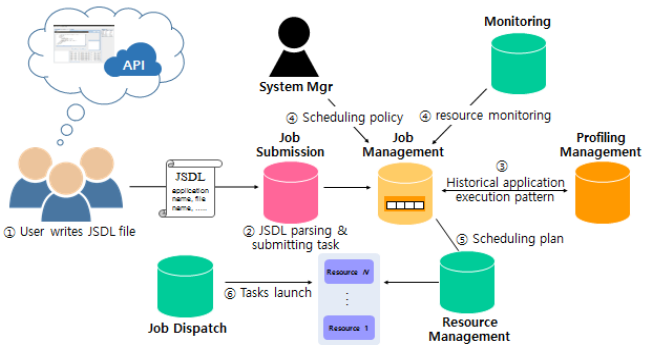

Fig. 2 Execution Scenario of the integrated framework

4) After collecting the quantitative information of available resources via the Monitoring module, the Job Management module decides a scheduling plan for the tasks based on the scheduling policy (*e.g.,* Application-aware anti-interference method) and sends it to the Resource management module.
5) Resource Management module launches the virtual machines, according to the pre-established plan which depends on the multiple type of the resources allocated.
6) Job Dispatch module prepares to launch the tasks by checking input and execution file and, finally, sends the tasks into the VMs.

C. Application-aware Anti-interference scheduler

It has been known that lots of computational jobs running on datacenters tend to vary, covering CPU, Memory-intensive or I/O-intensive [10][11]. Such kind of jobs may be allocated inappropriate resource (for instance, memory) instead of the actual its desire (for example, CPU), eventually causing overall performance degradation as well as low throughput for all executing jobs. In order to guarantee ideal performance with limited resources, it is important to offer sophisticated and balanced scheduling by figuring what the application really needs and providing it so that whole applications are able to be run in anti-interference way.

The application-aware anti-interference scheduler, our *iSDF* is adopting, is developed to support aforementioned issue and delivers the following three features: first, it offers a fine-grained resource allocation exploiting resource needs and its degree (*i.e.,* CPU/Mem/Disk/Port weights) from the profiling management module in this framework. The second feature is 'anti-interference'-based scheduling service which is scheduling tasks according to the needs for the multiple types of resources and weights resulting in well-balanced resource utilization and letting the tasks be oblivious of interference. The last feature is a workflow-supportive scheduling which offers resource reservation using Maximum Estimated Resource requirements (*MER* is induced from the resource

requirement of each task and the maximal concurrent degrees of a workflow where the degree of concurrency in a level is the number of tasks in the level). Suppose that a workflow job is submitted where *MER:{cpu=5, Mem=1}*. Our scheduler gives priority to this workflow job by reser*ving (allocating) resources (cpu=5, mem=1)* and then reschedules other tasks with remained resources toward the way to minimize interference between all jobs.

Fig. 3 is a flowchart of our scheduling algorithm. If the current job is workflow, it allocates resources using its MER value. When a task has dominant resources, it schedules to maximize resource consumption (minimize remaining resources) by putting resources into tasks that have opposing dominant resource requirements.

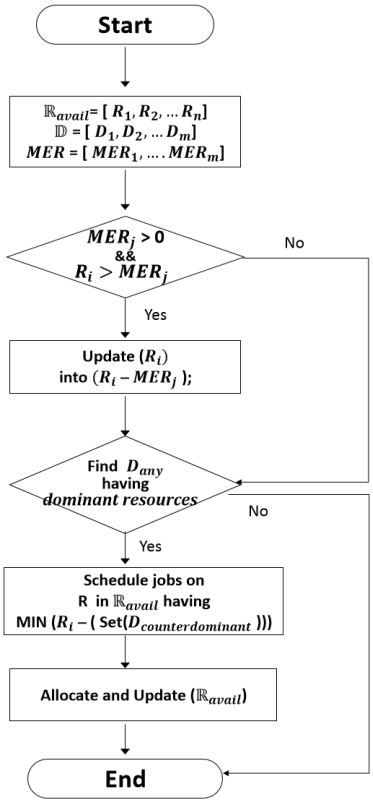

Fig. 3 A Flowchart of the Application-aware Anti-interference Scheduling

Fig. 4 illustrates a simple example to offer better understanding on the mechanism of our scheduling method. Resource management module gets reported the total amount of the available resources regularly (Step 1). Monitoring module updates the resource capacities and in the meantime, Job management module acquires the resource availability from the Monitoring (Step 2). Given the list of submitted jobs, their resource demands are *Job*1:{2cpu, 2gb, MER=null}, *Job2*:{1cpu, 5gb, MER=3} which are offered by Profiling Management module (Step 3). The job management module gives priority to task2 since it (belonging to Job2) has *MER* value, and reserves resources as follow: 3*{1cpu, 5gb}→[3cpu, 15gb]. After, task 1 and task 3 are scheduled on the remained resources in ascending order of the resource needs (Step 5).

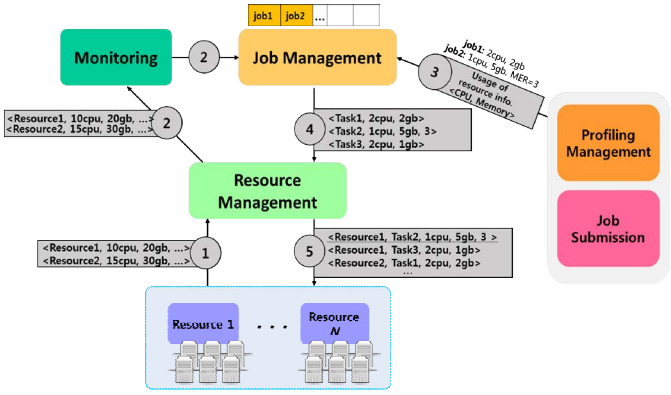

Fig. 4 An example of the Application-aware Anti-interference Scheduling Method

IV. Performance Evaluation

Here, we shows the performance evaluation of our framework.

A. Experimental environment & setting

We created an experiment environment with the features specified on Table 1. In this paper, we construct a private cloud environment based on OpenStack [12] version *Liverty* as a computing environment. They are Intel Xeon CPU E6-1650 v4 @3.60 GHz with 12 cores & 32GB RAM for master, and two of slave. We use two applications in this experiments: *Autodock* which is bag of tasks application, and *Montage which is a workflow application.*

TABLE 1 EXPERIMENTAL SETTING

	CPU	Memory(GB)
Controller	12	32
Compute1	12	32
Compute2	12	32

AutoDock [13] is a molecular modeling simulation software which is largely used for drug repositioning area. For our experiments, we choose autodock3 which aimed at performing the docking of ligands to a set of target proteins in order to find out possible new drugs for several serious diseases such as SARS and Malaria, from a wide range of scientific computing applications to act for CPU-intensive jobs having small input (6.1MB)/output data(3.0KB) files with low memory usage, as shown on the Table 2. *Autodock* has 0.7% memory utilization while maintain 100.0% CPU utilization.

Montage [14] is an astronomical image mosaic engine for making complex Flexible image transport system (FITS) 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 [15], into a mosaic. It is considered a data-intensive application having not only moderately lots of input (5.5 GB) and output (3.2 GB) data files but also intermediate files. Montage has quite high memory utilization whereas CPU utilization is relatively low, 90.8 and 9.1%, respectively. Table 2 shows a measure of resource usage of each application.

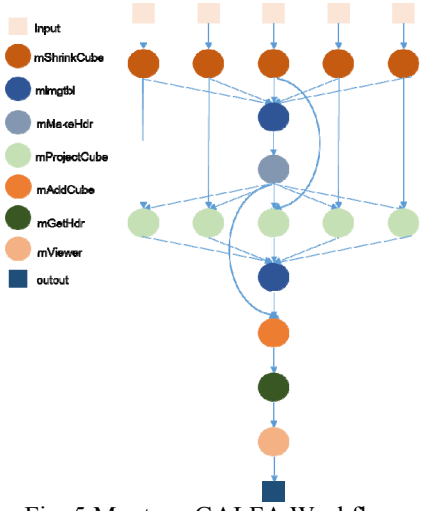

Fig. 5 Montage GALFA Workflow

TABLE 2 A MEASURMENT OF RESOURCE UASGE OF APPLICATIONS

Application	Avg. use of CPU(%)	Avg use of Mem(%)
Autodock	99.2	1.1
Montage	87.3	10.5

In this experiment, we carried out the Autodock application with one thousand of ligands, and each of tasks uses 1 core of CPU, 0.1MB of Memory. Montage application, defined as Fig. 5, consists of 16 tasks and each task of it needs 1 core of CPU, and approximately 1GB of Memory.

To demonstrate effectiveness of our framework, we consider three scenarios having difference scheduling and allocation policy each other, which are detailed as the following:

1. A scheduling with no aid of application profiling, no consideration of overall utilization
2. A scheduling using the application profiling without anti-interference control
3. A scheduling with application profiling and anti-interference control

In the first case, we assume that the default policy to allocate resources is performed as distributing resources in the ratio R/*n* % (where R is the ratio of total available resources and *n* is the count of applications to run). To validate the anti-interference function of our scheduler, the second and third scenarios are performed, both of which utilize the profiling function. This way we can note the differences with or without our scheduler precisely.

We ran two applications, which are mentioned before, for 100 times respectively and calculated the average costs in terms of makespan and utilization over the aforementioned three scenarios.

B. Results and Analysis

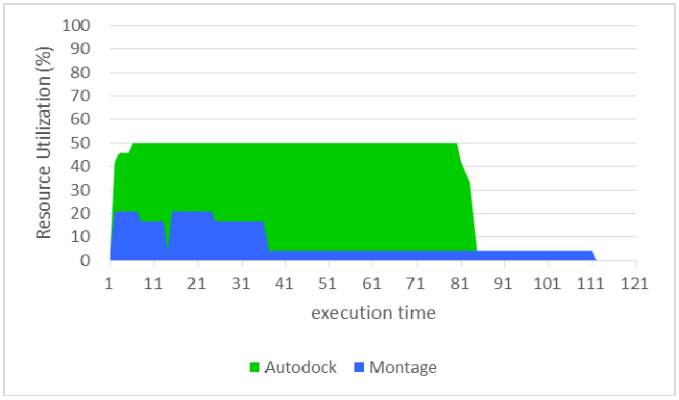

Fig. 6 Experimental result of Scenario #1

With respect to the addressed condition, Fig. 6 – 8 depict the results, respectively.

In the first case where the default scheduling policy is employed, the jobs for each applications can be given 50% of total resources, respectively, because only two applications are carried out and the initial rate of available resources is 100% (from $R/n = 100/2\%$).

As seen on the Fig. 6, it leads to low resource utilization over all, since the jobs for each application could not take enough resources they preferred and could be performed with fixed number of resources.

The second scenario generates interesting result as plotted on Fig. 7. Although each task received the proper virtual resource that furnished the desired resources using profiling module, we can notice that the resource starvation for workflow application (Montage) encountered while they are running. Thus, it incurred the long latency and makespan time for specific application such as workflow.

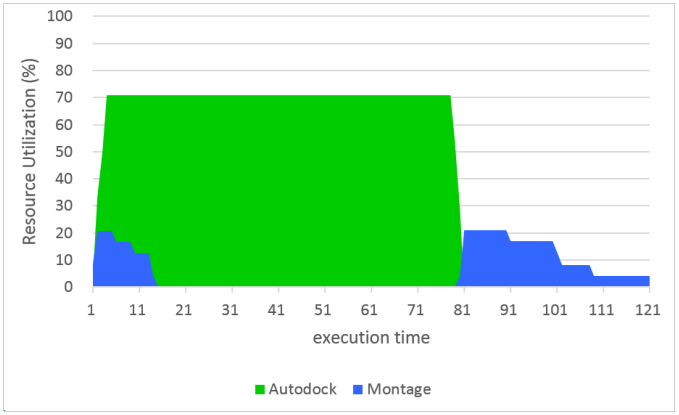

Fig. 7 Experimental result of Scenario #2

The result seems to be caused by the pursuit of resource being more concentrated on the specific one for each application (e.g., cpu for Autodock) and overall utilization for each server being not considered.

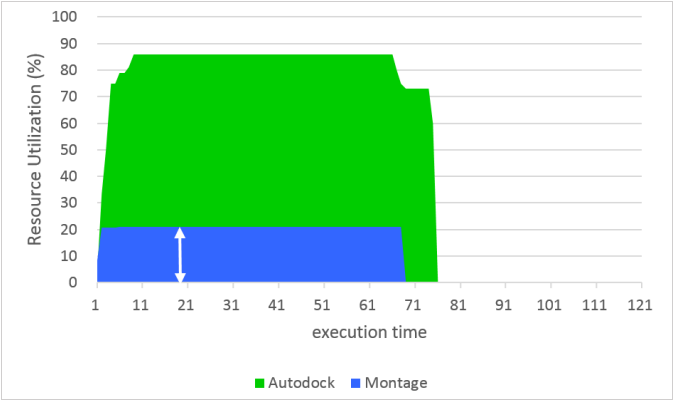

Fig. 8 Experimental result of Scenario #3

As mentioned before, the third case uses both the profiling and anti-interference control where the scheduler reserves resources up to the MER (Maximum Estimated Resource needs) if workflow exists. Its result is depicted on the Fig. 8 where the arrow indicates the reservation. By properly reserving resources for workflow, the scheduler is able to prevent the resource starvation as well as performance degradation. In addition, it is found that the anti-interference scheduling considering utilization for each server gives highly effects on improving the utilization for the whole servers, which eventually contributes to shorter makespan time.

V. Conclustion and Future work

Lots of investigations to configure the new framework for cloud computing tailored to cover diverse applications and resources are increasingly made in the recent computing industry. Due to the disruptive properties of scientific applications, it is still challenging to adapt them in virtualized computing environment.

To address this problem, this paper proposed an integrated Software-Defined computing Framework for scientific experiments which is referred as *iSDF* that provides the sophisticated allocations over diverse types of resources by considering application's properties. By supporting an integrated environment that's easy to use , in addition, users can efficiently utilize it with no supplementary knowledge on complex parts. The experimental results to evaluate our framework showed that our framework can achieve a 51% increase in terms of resource utilization, and the improve the average makespan time by a 19% compared to the conditions that do not include the policy.

Our future work is compensating this framework by adjoining a module to take care of data management that enables users not to perform duplicated simulations. Moreover, we are going to perform additional experiments on the hybrid cloud environments that include both private and public clouds.

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP)(NRF-2017R1A2B4005681)

References

[1] Li, C-S., et al. "Software defined environments: An introduction." *IBM Journal of Research and Development* 58.2/3 (2014): 1-1.

[2] 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), 2012 SC Companion: (pp. 1341-1342). IEEE.

[3] Wang, XiaoYing, et al. "Appliance-based autonomic provisioning framework for virtualized outsourcing data center." Autonomic Computing, 2007. ICAC'07. Fourth International Conference on. IEEE, 2007.

[4] J. Lee, H. Kang, Y. Kim, "Energy-Efficient Provisioning of Virtual Machines for Workflow Applications", KNOM Review, Vol. 16, No. 1, July 2013, pp. 35-42.

[5] L. F. Bittencourt , E. R. Maderia, "A Performance-oriented Adaptive Scheduler for Dependent Tasks on Grids," Concurrency and Computation: Practice and Experience, vol. 20, pp. 1029-1049, 2008.

[6] Anjomshoaa, Ali, et al., "Job sumbission description language (jsdl) specification, version 1.0" Open Grid Forum, GFD. Vol. 56. 2005

[7] Dellinger, Matthew, Piyush Garyali, Binoy Ravindran., "ChronOS Linux: a best-effort real-time multiprocessor Linux kernel." *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*. IEEE, 2011.

[8] Hindman, Benjamin, et al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." NSDI. Vol. 11. No. 2011. 2011.

[9] Kim, S., Kim, J. S., Hwang, S., Kim, Y., "Towards effective science cloud provisioning for a large-scale high-throughput computing" Cluster computing, 17(4), 1157-1169.

[10] Bikash Sharma, Ramya Prabhakar, Seung-Hwan Lim, Mahmut T. Kandemir, and Chita R. Das. "Mrorchestrator: A fine-grained resource orchestration framework for mapreduce clusters" In IEEE CLOUD, pages 1–8, 2012.

[11] R. Boutaba, L. Cheng, and Q. Zhang. "On cloud computational models and the heterogeneity challenge" J. Internet Services and Applications, 3(1):77–86, 2012

[12] Openstack, https://www.openstack.org/

[13] Autodock, http://autodock.scripps.edu/

[14] Montage, http://montage.ipac.caltech.edu/

[15] The Galfa-hi Survey: Data release 1. Astrophys. J. Suppl. 194(2), 13 (2011)