A VM Vector Management Scheme for QoS Constraint Task Scheduling in Cloud Environment

Kyung-no Joo(✉), Seonghwan Kim, Dongki Kang, Yusik Kim, Hyungyu Jang, and Chan-Hyun Youn

Department of Electrical Engineering, KAIST, Daejeon, Korea
{eu8198, s.h_kim, dkkang, yusiky326, chyoun}@kaist.ac.kr

Abstract. To reduce operational costs in computing service, there have been many researches on resource utilization improvement. In cloud environment, virtualization technology, coupled with virtual machine migration, can improve utilization of physical machines by server consolidation. Cloud service providers will consolidate virtual machines in order to reduce the number of physical machines running, therefore reducing their operational cost. Capacity of resources used by virtual machines can be set by users who schedule their tasks, minimizing resource waste by underutilization. However, it is difficult for a user to find the optimal virtual machine with respect to the resource capacity in minimal cost. To solve this problem, cloud service broker is required between users and cloud service providers. Task scheduling in cloud service broker solves finding virtual machine with lowest cost while satisfying SLA. Previous methods using mixed integer programming have showed difficulties in complexity and as system got larger and more complex, they could not solve the problems effectively. In this paper, with preliminary experiment, we propose vector modeling on virtual machine types and tasks can be applied and used in VM management. The allocated computing resources for each task components showed low complexity in operation of VM managements and effectiveness in task consolidation.

Keywords: Cloud computing · Scheduling workloads · SLA

1 Introduction

Cloud computing service provides computing resources such as CPU, RAM, storage and network through internet as much as users want to use, as long as they are willing to pay. This model is called "pay-per-use model" and it is one of the major characteristics of cloud computing service. Cloud computing provides scalable, theoretical infinite computing resources to users with low risk of resource waste because they can always stop using the service with no penalty or investment made whenever demand for resources has disappeared [1, 2].

Users who are willing to use cloud service to process their requests must decide which resource to use. There are many cloud service providers providing many different types of virtual machines (VM) with different computing power and price. Not only that, there is no standard to compare performance of different VM types across different cloud service providers. As a result, users find it difficult to compare performance across different cloud service providers. Moreover, SLAs guaranteed by cloud service providers [3, 4] are limited to resource capacity and availability with given prices. Since performance level for each specific application is not guaranteed, users find it difficult to make optimal decision in choosing which resource to buy. One solution to this problem is to have a brokering service layer to solve resource allocation decision problem instead. The broker needs to translate user requesting SLA consisting of time and cost constraints to SLA which cloud service providers provide to users: resource capacity, price and availability. Then, users can simply request jobs with time and cost constraints and cloud broker service can decide whether or not the job can be processed within constraints.

In cloud computing, with virtualization technology [5], one physical machine (PM) runs more than one virtual machines (VM). Therefore, through virtualization, multi tenants can be served by a single physical machine. This approach is called server consolidation and it allows efficient usage of computing resources by running as many as VMs possible in a single PM as long as performance is not affected [6]. Addition to server consolidation, scheduling multi-requests to a single VM is also introduced [7]. Resource capacity of VM types provided by cloud service providers are relatively coarse-grained and there are not many tasks which can fully utilize given computing power with a single task alone. However with task consolidation, brokering service can improve resource utilization [7, 8]. There are two issues in task consolidation: how many tasks can we consolidate execute with a given VM type's resource capacity? And which tasks should be consolidated together to maximize resource utilization?

We want to solve a decision problem: choose a VM type provided by multi cloud service providers to schedule a QoS constraint application to minimize operational cost for cloud service broker. To maximize brokering service's profit, maximizing resource utilization is necessary. Some previous works [6, 9] used linear programming (LP) optimize the scheduling problem. However, using LP solver as optimizer has complexity issue: if system is complex, such as hybrid cloud environment, optimizing process takes too long to schedule tasks on-line. Also, they do not consider how the services are placed taking into account load balancing constraints in terms of individual resource components. Others [7] scheduled multi requests to a single VM in order to fully utilize a VM resource as long as SLA violations do not occur. However, they do not consider balance between resource components. If any resource component is fully utilized by scheduling specific resource-intensity tasks into same VM, other resources can be waste because no more task can be scheduled due to fully utilized single re-source component. Memory is fully almost fully utilized while CPU is underutilized. In this case, no more tasks can be scheduled because there is no sufficient memory resource left although CPU is underutilized. CPU resource becomes resource waste.

In this paper, we propose a VM vector management scheme which consolidates tasks to improve resource utilization of cloud resources. We define vectors representing VM types and tasks. We used vector sum and vector dot products to define scheduled total tasks, latency factor and balance factor. Using this information, we propose a scheduling scheme with low complexity. We deployed the proposed VM vector management scheme into scheduler in the brokering system

We will discuss the performance of the proposed system with experimental result in Sect. 3.

2 VM Vector Management Scheme in Task Scheduling

2.1 Modeling Task Consolidation in Cloud Computing Environment

To schedule incoming requested tasks online while consolidating them to minimize number of VMs running, we need an efficient way to calculate expected overall utilization of a VM after scheduling. If the VM is expected to have overutilization after scheduling the task, then the task must be scheduled to different VM. Otherwise, every task on the VM has a risk of violating SLA due to context switch delays.

To test how resource utilization changes when tasks are consolidated, we performed preliminary experiments in a cluster of cloud computing environment with OpenStack cloud plat-form installed. Detailed explanation on experimental environment is in Sect. 3.1. For simplicity, we focused on a single VM type with 1 VCPU, 1 GB RAM, and 10.0 GB disk resource capacity. We monitored resource utilization of RAM using System Information Gatherer And Reporter (SIGAR) [10]. MapChem task, which is a CPU-intensive sequencing task used in bioinformatics, is consolidated in a VM during the experiment. Table 1 shows how resource utilization changed as tasks are consolidated in the same VM.

Table 1. Memory utilization changes when tasks consolidate

	Number of tasks consolidated			
	0	1	2	3
Memory utilization	0.183593	0.023664	0.278868	0.328247
Change	–	+0.0530	+0.0422	+0.0494

Memory usage when no task is scheduled comes out to be about 0.18. This is because the VM is running Ubuntu 12.04 Server OS and other daemon processes such as Tomcat server to receive MapChem application requests from the broker. As one additional task requested and executed in parallel in the same VM, memory utilization increased by 0.048218 in average (Table. 2).

Table 2. CPU utilization changes when tasks consolidate

	Number of tasks consolidated		
	1	2	3
Execution time (s)	40.992	92.706	128.243
CPU utilization	0.20496	0.46353	0.641215

CPU utilization was difficult to measure with just SIGAR because it approaches 1.0 when VM is executing any CPU-intensive task. Therefore, we redefined CPU utilization for **QoS constraint applications** which have required deadline to execute and if deadline is not met, the broker loses profit. In other words, CPU Utilization for QoS Constraint Applications are defined as

$$U_{CPU} = \frac{(Used\ CPU\ Cycles\ by\ Given\ Task)}{(Total\ CPU\ Cycles\ Available\ \ Until\ \ Deadline)} \tag{1}$$

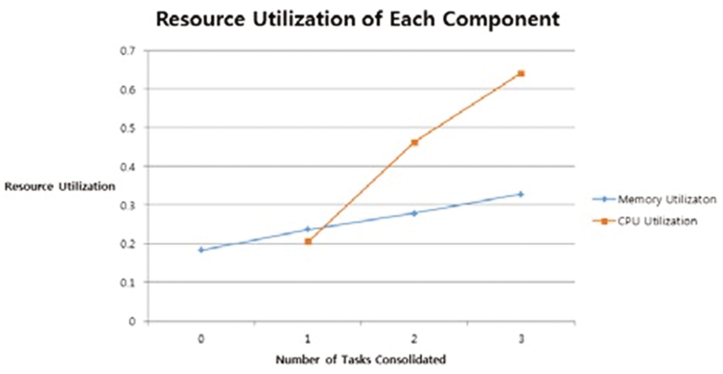

Fig. 1. Resource utilization when tasks consolidate

Both memory and CPU utilization changes come out to be linearly proportional to number of tasks consolidated in the VM currently. Linearity is shown in Fig. 2.

From this preliminary experiment, we found that resource utilization changes have linearity when tasks are consolidated. Each resource component follows different amount of change as shown in the Fig. 1: different resource component has different slope. We model multiple resource components' utilization changing with linearity with vector model. It is described in detail in Sect. 2.2.

2.2 Vector Models

There are many VM types provided by multi cloud service providers. They are categorized with different capacity of resources each VM type uses. Different VM types which different cloud service providers provide as service make it difficult for users to decide optimal choice of VMs online. To ease solving such complex decision problem online, we use vector modeling to represent each VM type and tasks.

We define a VM type vector as a unit vector: (1.0, 1.0…). Each value represents utilization value when the entire resource component is used. Therefore, task vectors differ as the task. A task vector is relative to which VM type it is going to be executed

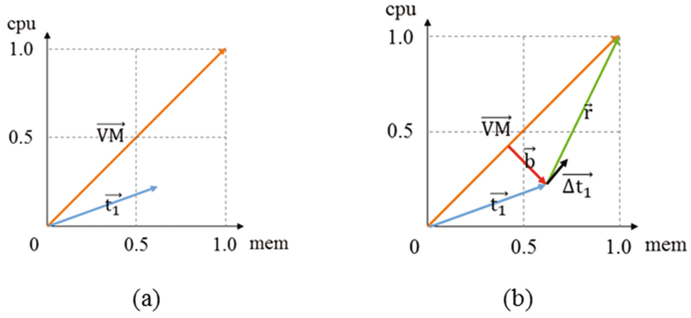

Fig. 2. (a) A VM vector, task vector, and balance factor in a vector model (b) latency factor considered in scheduling

with. Magnitudes of each component represent average utilization of the resource component. The values can be calculated based on historical data of each task. Figure 2 describes relationship between a VM vector and a task vector.

We defined the The task vector $\vec{t}$ of task T as following.

$$\vec{t} = (c, m) \tag{2}$$

Where c is the average cpu utilization and m is the average memory utilization while executing T. Axises may be added or deleted elastically such as network I/O axis and storage axis. The values can be acquired by monitoring the VM while executing tasks. Let $\mathrm{cpu_T(t)}$ and $\mathrm{mem_T(t)}$ are obtained by monitoring the VM which runs the task T for time τ. Then, the c and m value can be calculated by

$$c = \frac{1}{\tau}\int_0^\tau \mathrm{cpu_T}(t) \tag{3}$$

$$m = \frac{1}{\tau}\int_0^\tau \mathrm{mem_T}(t) \tag{4}$$

When there are some tasks running on the same VM, the total task vector can be calculated by simply adding the all task vectors.

$$\vec{\mathbf{t}}_{\mathbf{total}} = \sum \vec{\mathbf{t}}_{\mathbf{i}} \tag{5}$$

$\vec{t}_{\mathbf{total}}$ is the total task vector where each component of the vector is representing resource utilization of each resource components. $\vec{t}_{\mathrm{i}}$ is a single task vector consolidated in the VM. When a task has finished execution, the task vector is removed from total task vector for corresponding VM. The total task vector represents overall resource utilization of all tasks executing on the VM.

We define a balance vector which is the vector perpendicular to a VM vector to a tack vector. We used vector dot product to get the balance vector. And the magnitude of this vector is defined to be balance factor. Balance factor represents level of unbalance of resource utilization between different resource components. Even an application which uses one resource component intensively requires at least some minimum amount of other resource types. If a VM's resource usage is unbalanced and any of resource component utilization goes near 1.0, no task can be scheduled on the VM and other underutilized resource components cannot avoid being wasted until fully utilized resource components get freed. Therefore, we use balance factor as task scheduling criteria to avoid such case of resource waste.

Definition 1 Balance Vector ($\vec{b}$) and Balance Factor ($\left|\vec{b}\right|$)

$$\vec{b} = \vec{t}_{total} - \left(\vec{t}_{total} \cdot \overrightarrow{VM}\right)\overrightarrow{VM} \tag{6}$$

We obtain balance vector and balance factor, which are presented in Fig. 2(a). The magnitude of balance vector is described in Eq. 7. (c, m) is set of CPU and memory resource utilization respectively.

$$\left|\vec{\boldsymbol{b}}\right| = \frac{|\boldsymbol{c} - \boldsymbol{m}|}{\sqrt{2}} \tag{7}$$

Also, we should consider a performance degradation when multi tasks run in parallel in a single VM. Such variation has several causes. One of them is interference effect caused by other VMs running on same PM [6]. Also, context switching between multi tasks running on same VM can cause performance degradation. Since our paper does not focus on causes of such degradation, we focus on dealing with performance degradation during scheduling more than one task to a single VM. We defined latency factor and when sum of task vectors and sum of latency factors have any resource component greater than 1.0, we scale the same VM type to avoid SLA violation.

Latency vector of task t is defined to be the standard deviation of utilization multiplied by κ

$$\overrightarrow{\boldsymbol{\Delta t}} = \kappa\left(\sigma_t^{cpu}, \sigma_t^{mem}\right) \tag{8}$$

Where $\sigma_{\mathrm{t}}^{\mathrm{cpu}}$ presents the standard deviation of $\mathrm{cpu_T(t)}$, $\sigma_{\mathrm{t}}^{\mathrm{mem}}$ denotes the standard deviation of $\mathrm{mem_T(t)}$ and κ represents the latency factor. The meaning of the latency vector is the maximum value that the task vector can cover. When we monitor the VM resource while running tasks, we can see that resource utilization varies in time. In order to avoid degradation, we should take the maximum value of resource utilization into account.

κ is determined based on 68-95-99.7 rule since we assumed that the resource utilization follows the normal distribution of $\mathrm{N}(\vec{\mathrm{t}}, \vec{\sigma})$. If we take κ = 1, resource utilization underlies $\vec{\mathrm{t}} + \vec{\sigma}$ with probability of 84 %. This value may not be correct since the resource utilization do not follow the normal distribution. However, we can notice that almost all values are covered if we set κ near 2. The more κ value we set, the more probability of $\vec{\mathrm{t}} + \kappa\vec{\sigma}$ coverage. In our paper, we set basic κ value 2.

We defined total latency vector by adding all latency vectors of each task as in Eq. (9). Figure 4 shows the graphical representation of latency vectors. We can see there are additional black vectors presented in Fig. 4. Blue lines stand for the task vector which is the average value of resource utilization. Also, by adding black latency vector to task vector, it shows the estimated maximum coverage of the task.

$$\overrightarrow{\Delta t} = \sum \overrightarrow{\Delta t_i} \tag{9}$$

2.3 Task Scheduling Algorithm with VM Vector Management Scheme

The proposed scheduling scheme is presented as heuristics. There are four steps:

Step 1. Determine VM type candidates which can satisfy SLA requirement user request task based on historical data.
Step 2. Generate task vectors for each VM type generated in Step 1.
Step 3. Remove VM type if adding new task vector will have latency factor with any component greater than 1.0 and can violate SLA requirement due to performance degradation.
Step 4a. Find VM type which has minimum balance factor
Step 4b. If no VM type is available, VM type with minimal cost in Step 1 is allocated.

In Step 1, Based on task profiles and VM type profiles from historical data stored in repositories, available VM types which expect to satisfy SLA requirement are retrieved. Already allocated resources in resource pool are considered first. If there are no available allocated resources and auto-scaling process must proceed, cheapest VM type which can satisfy SLA requirement is chosen. In Step 2, task vectors are generated. Resource utilization differs by which VM type the task will execute on; therefore, each task generates many task vectors. Task vectors include both average utilization and standard deviation of the utilization for each resource components. Average utilization is used to calculate balance factor and the standard deviation of the utilization is used to calculate latency factor. In Step 3, VM types which already allocated in the resource pool and have high resource utilization are removed from the candidate list using latency factor. If any resource component utilization becomes greater than 1.0 after the scheduling it is expected to violate SLA requirement. Therefore, we decide not to schedule any more tasks until any of already executing task terminates. In Step 4, since any of remaining VM types in the available VM list can be used to execute the requested task we choose the VM type which has minimum balance factor. In this way, we manage all VMs utilized with balance between all resource components. We avoid resource waste because resource usage is unbalanced. If there is no VM type which satisfies by now, scaling number of VMs is necessary. We allocate new VM for the VM type which has lowest cost and satisfies SLA requirement of the given task. Algorithm 1 shows the entire procedure of vector-based balance scheduling algorithm (Fig. 3).

3 Experiments and Evaluation

The user request consists of workflow topology W and the service level agreement SLA. We used colored Petri-net model in order to represent the workflow topology W. In our experiments, all Physical Machines (PMs), controller node and compute nodes have two quad-core processor with hyper-threading (Intel Xeon Processor E5620), 14 GB of RAM and 1000 GB of disk. Ubuntu 12.04 server is installed in all PMs and OpenStack computing environment is installed on top of the operating system.

Algorithm 1. Vector-based Balance Scheduling Algorithm
Input: user request task $\mathbf{t_{req}}$, deadline $\mathbf{DL_{req}}$
Output: scheduled VM

```
01: Get available VM types and put into AvailableVMList
02: Calculate task vector $\mathbf{t_{req}}$ for all possible VM types
03: ProposedType = VM Type which is available and the cheapest
04: for all VMs $\in$ AvailableVMList do
05:     Let $\{\overrightarrow{t_1}\}$ be the scheduled task vector list inside the VM where $i \geq 0$
06:     if $\sum(\vec{t}_1 + \overrightarrow{\Delta t_1}) + \vec{t}_{req} + \overrightarrow{\Delta t_{req}}$ has element which is larger than 1.0 then
07:         continue
08:     $\vec{t}' = \sum \vec{t}_1 + \vec{t}_{req}$
09:     Calculate $|\vec{b}|$ using eq. 7
10:     if $|\vec{b}|$ is smaller than $\min_b$ then
11:         $\min_b = |\vec{b}|$
12:         scheduledVM = VM
13: if scheduledVM is null then
14:     scheduledVM = new VM with type = proposedType
15: schedule and run $\mathbf{t_{req}}$ onto scheduledVM
16: $\overrightarrow{\mathbf{scheduledVM}} = \overrightarrow{\mathbf{scheduledVM}} + \overrightarrow{t_{req}}$
```

Fig. 3. Vector based balance scheduling algorithm

3.1 Performance Metric

We defined two performance metrics: total cost and SLA violation rate. Each VM type has its own VM cost. Therefore, we define total cost as the sum of each VM's cost.

$$\mathrm{TC} = \sum_{i \in K} C^i \cdot m^i_{BTU} \tag{10}$$

We also defined SLA violation rate. User requests with time constraints and we choose which VM type to execute the task with based on the constraint. However, if SLA is not satisfied, we count it as SLA violation case. SLA violation rate is the ratio of such case to all user requests. Therefore, SLA violation rate is defined as follows:

$$\mathrm{VR} = \frac{\text{number of requests not satisfying deadline}}{\text{number of requests}} \tag{11}$$

3.2 Application Service Scenario

To evaluate our proposed scheme, we experimented with other schemes as well to compare as following:

Single-Request Single VM (SRSV) scheme [8] – Only a single request is scheduled to a single VM. This scheme rarely violates SLA requirement, however it has lowest utilization compared to other schemes.

Multi-Request Single VM (MRSV) scheme [8] – Multi requests are scheduled to a single VM and executed in parallel. This scheme does not consider balance between usages of different resource components (CPU, memory, and disk I/O etc.).

Vector-based Balanced Scheduling (VBS) – Multi requests are scheduled to a single VM and executed in parallel. This scheme uses balance factor as scheduling criteria in order to balance between usages of different resource components. It also considers performance degradation with latency factor as auto-scaling criteria, to reduce SLA violation rate.

All of above schemes are tested with same set of request inputs. We generated random requests which are combination of CPU-intensive task and memory-intensive task. The amount of workload for each task also differs and randomly selected. As we assumed with more resource capacity, performance increases. For example, VM type with 2 CPU cores finishes task execution with only half of time, which VM type with 1 CPU core would use. We experimented multiple times with average inter-arrival time being different starting from 10 s to 20 s. (λ = 10, 12, 14, 16, 18, 20) Input request follows Poisson arrivals. To be fair, we used same input sets for all schemes.

We evaluated the proposed scheme for three times with different pricing models. First scenario charges \$0.03 per CPU core and \$0.03 per Gigabyte RAM. Therefore, the price ratio between CPU and memory units is 1:1. Second scenario charges \$0.03 per CPU core and \$0.02 per Gigabyte RAM. The price ratio between CPU and memory units is 3:2. Third scenario charges \$0.03 per CPU core and \$0.04 per Gigabyte RAM. Therefore, the price ratio between CPU and memory units is 3:4. Total cost and violation rate measurement experiment result is shown in Fig. 4.

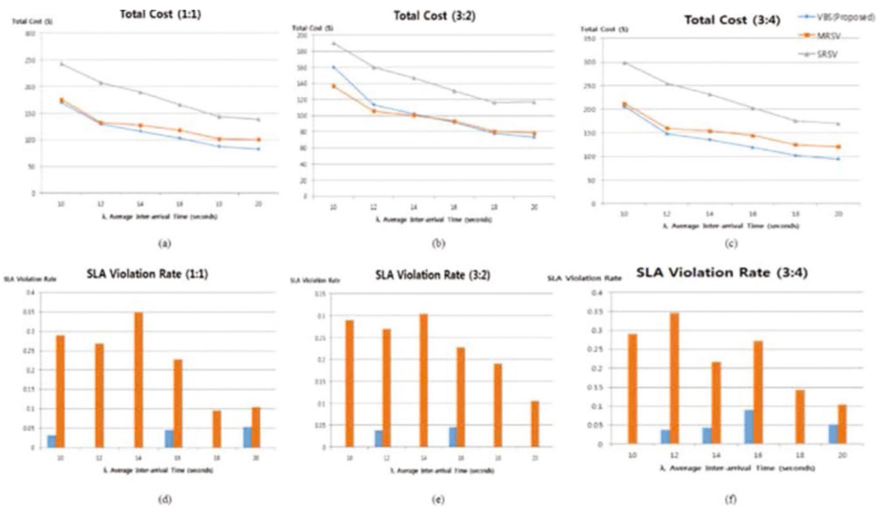

Fig. 4. Total cost and SLA violation rate as inter-arrival time varies when price ration is 1:1, 3:2, and 3:4

Both MRSV and VBS, which use task consolidation, have lower total cost spent for all inter-arrival time scenarios compare to SRSV. SRSV does not fully utilized resource of VM types which is set in coarse-grained manner. Our proposed scheme, VBS has even lower total cost compared to MRSV, because we have better resource utilization as we balance usage between different resource components. Compare to other schemes [8], total cost decreased as shown in the following figure. This is because, scheduling while considering balance between resource components increased utilization rate throughout the experiment. On the other hand, violation rate is decreased. This is because we scale number of VMs considering performance degradation due to executing too many request on a single VM causing latency by context switching. Also, interference effect also decreased because balance between resource usages reduces interference effect between tasks.

4 Conclusion

In this paper, we proposed vector models for VM types and tasks and QoS constraint task scheduling heuristics with low complexity. Task vector is a simple model to represent resource utilization made of each resource components in VM types. Based on resource utilization historical data, task vectors are generated to calculate expected resource usage before scheduling. Also, expected performance degradation is considered to minimize SLA violations due to scheduling too many tasks into a single VM.

With the experiment, we proved our scheme improved in terms of total cost and SLA violation rates compared to other schemes such as MRSV and SRSV. With balance factor as scheduling criteria, our scheme improved resource utilization of VMs because we avoid cases of being unable to schedule new task because one resource component being fully utilized and other resource components being under-utilized. This resulted low operational cost as shown in the experiment. Also, with latency factor as auto-scaling criteria, our proposed scheme minimized SLA violation which occurs during task consolidation. This can lead to increase in profit of cloud service broker because penalty fee related to SLA violation is minimized.

Acknowledgement. This work was partly supported by 'The Cross-Ministry Giga KOREA Project' grant from the Ministry of Science, ICT and Future Planning, Korea and Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. B0101-15-0104, The Development of Supercomputing System for the Genome Analysis)

References

1. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. **147**, 195–197 (1981)
2. May, P., Ehrlich, H.-C., Steinke, T.: ZIB structure prediction pipeline: composing a complex biological workflow through web services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)

3. Foster, I., et al.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Czajkowski, K., et al.: Grid information services for distributed resource sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)
5. Foster, I., et al.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov
7. Ren, Y.: A cloud collaboration system with active application control scheme and its experimental performance analysis. In: KAIST (2012)
8. Kang, D.K., et al.: Enhancing a strategy of virtualized resource assignment in adaptive resource cloud framework. In: Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication. ACM (2013)
9. Lucas-Simarro, J., et al.: Scheduling strategies for optimal service deployment across multiple clouds. Future Gener. Comput. Syst. **29**, 1434–1441 (2012)