An Approach to Optimized Resource Allocation for Cloud Simulation Platform

Haitao Yuan[1], Jing Bi[2], Bo Hu Li[1,3], Xudong Chai[3]

[1] School of Automation Science and Electrical Engineering, Beihang University, 100191 Beijing, China
[2] Department of Automation, Tsinghua University, Beijing, China
[3] Beijing Simulation Center, 100854 Beijing, China
buaahaitao@gmail.com

Abstract. Resource allocation for simulation applications in cloud simulation environment brings new challenges to infrastructure service providers. In order to meet the constraint of SLA and to allocate the available virtualized resources optimally, this paper first presents autonomic resource management architecture, and then proposes a resource allocation algorithm for infrastructure service providers who want to minimize infrastructure cost and SLA violations. Our proposed algorithm can maximize the overall profit of infrastructure service providers when SLA guarantees are satisfied or violated in a dynamic resource sharing cloud simulation platform. The experimental evaluation with a realistic workload in cloud simulation platform, and the comparison with the existing algorithm demonstrate the feasibility of the algorithm and allow a cost-effective usage of resources in cloud simulation platform.

Keywords: Cloud Simulation; Service Level Agreement (SLA); Resource Allocation; Virtualization.

1 Introduction

Cloud computing can be classified as a new paradigm for resource allocation of computing services supported by state-of-the-art data centers that usually employ virtual machine (VM) technologies for consolidation and environment isolation purposes [1]. An increasing number of organizations (e.g., research centers, enterprises) benefit from cloud computing to host their applications [2,3]. Different from cloud computing, cloud simulation is a new network-based and service-oriented simulation model [4,5]. The cloud simulation virtualizes different types of simulation resources and further constitutes the "resource pools". Hence consumers can acquire services of the simulation resources at anytime and anywhere using the networked cloud simulation platform.

To fully realize the potential of cloud simulation, infrastructure service providers (ISPs) of cloud simulation platform have to ensure that they can be flexible in their service delivery to meet various simulation requirements, while keeping the simulation consumers (SCs) isolated from the underlying infrastructure. In such a

dynamic environment where SCs can join and leave the cloud simulation environment at any time, ISPs should be able to provide their SCs with the required services according to a given service level agreement (SLA). ISPs should ensure those QoS requirements using a minimal amount of computational resources. Consequently, an efficient and dynamic resource allocation strategy is mandatory in the infrastructure layer.

The current works in cloud computing [6,7,8] focused mostly on the problem of resource allocation management and maximizing the profit of ISPs in cloud simulation environments. Many works do not consider the SCs-driven management, where resources have to be dynamically rearranged based on SCs' requirements. Fu Y. et al [9] proposed an SLA-based dynamic scheduling algorithm of distributed resources for streaming. Moreover, Yarmolenko V. et al [10] evaluated various SLA-based scheduling heuristics on parallel computing resources using resource (number of CPU nodes) utilization and income as evaluation metrics. Nevertheless, our work focuses on scheduling multi-tier simulation applications based on VMs in cloud simulation environments, where the heterogeneity and inaccuracy of job requests may result in resource under-utilization. Lee et al [11] investigated the profit driven service request scheduling for workflow. In contrast, our work focuses on SLA driven QoS parameters from both the SCs' and the ISPs' point of view, and solves the challenge of dynamic changing SCs' requests to gain profit.

In order to meet the constraint of SLA and allocate the existing virtualized resources optimally to minimize SLA violations, this paper presents the autonomic resource management framework based on virtualization techniques and provides an effective virtualized resource allocation strategy for existing cloud simulation infrastructure environment and allocates VMs to serve SCs' requests in the resource management middleware of cloud simulation platform. The proposed VMs allocation algorithm which allows a cost effective usage of resources in cloud simulation platform is proposed and compared with the existing algorithm. Results show that the proposed algorithm can ensure to maximize the overall profit of ISPs when SLA guarantees are satisfied or violated.

The rest of this paper is organized as follows. Section 2 describes the autonomic resource management and present proposed system profit model. Section 3 presents the VM allocation strategy and algorithm. Section 4 demonstrates the results of prototype experiments. Concluding remarks and discussion about future work are given in Section 5.

2 System Overview

This section first provides an overview of our autonomic computing approach for the resource allocation problem in multitier virtualized environments of cloud simulation platform. Then, we present the performance and profit models of the system.

2.1 Autonomic Resource Management

This section first provides an overview of our autonomic computing approach for the resource allocation problem in multitier virtualized environments of cloud simulation platform. Then, we present the performance and profit models of the system.

Due to such the large variation in the simulation tasks, it is difficult to estimate workload requirements in advance, and planning the capacity for the worst-case is either infeasible or extremely inefficient. In order to dynamically allocate resources for multitier virtualized simulation application execution environments, the most common approaches are based on Monitor, Analyze, Plan, and Execute (MAPE) control loops [12, 13]. Autonomic systems maintain and adjust their operations in the face of changing components, demands or external conditions and dynamically allocate resources to applications of different customers on the base of short-term demand estimates. The goal is to meet the application requirements while adapting IT architecture to workload variations.

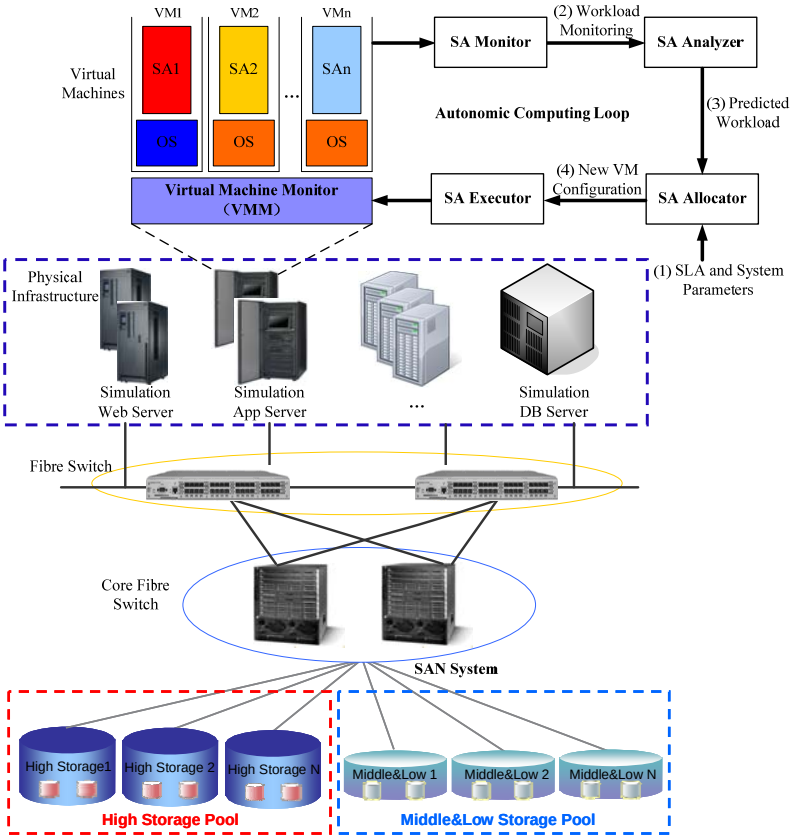

Fig. 1. Autonomic computing architecture for cloud simulation platform.

The architecture of a high-level autonomic computing overview is shown in Fig.1, includes a set of heterogeneous physical servers which run a Virtual Machine Monitor (VMM), shared by multiple independent application environments, hosting simulation applications (SAs) from different professions or departments. Modern SAs are usually designed using multiple tiers, which are often distributed across different servers. Server physical resources are partitioned among multiple VMs, each one hosting a single SA. Among many servers' resources, we choose CPU as the key resource in the allocation problem. The resource allocator exploits the VMM API to dynamically partition CPU capacity among multiple VMs and their hosted SAs.

Autonomic computing architecture provides mechanisms to automate the configuring of VMs and to tune the virtualized simulation multi-tier application, therefore the response time requirements of the different SCs can be guaranteed. It determines the run-time allocation for cloud simulation platform. It mainly includes four components described as follows:

- SA Monitor: collects the workload and the performance metric of all running SAs, such as the request arrival rate, the average service time, the CPU utilization, etc.
- SA Analyzer: receives and analyzes the measurements from the monitor to estimate the future workload. It also receives the response times of different SCs.
- SA Allocator: sets up allocation strategy for each tier of the SAs, and uses optimization algorithms to determine resource allocation.
- SA Executor: assigns the VM configuration, and then runs the SAs to satisfy the resource requirements of the different SCs according to the optimized decision.

2.2 System Performance and Profit Models

To solve the problem of resource allocation in the cloud simulation platform, this section describes how we establish the analytic model to estimate the performance in terms of SLA.

In cloud simulation environment, different SCs with different performance requirements may use simulation services supported by ISPs. What's more, the ISPs' objective is to serve requests with different requirements such that the ISPs' profit can be maximized and the SCs' performance requirements can be also guaranteed. In this section, we explain the system profit model. The properties defined in the SLA are as follows:

- Request Type: It defines the SCs' request types, which are 'Gold', 'Silver' and 'Bronze'. The corresponding performance requirements are different for different request types.
- Response Time: It is defined as the actual time taken for a SA request to go through the three-tiered system. The value of response time is different in each type, and specified in SLA.
- VM Type: There are three VM types, which include 'Large', 'Medium' and 'Small'. They are corresponding to three kinds of SCs.
- VM Price: Price of VM is decided by the number of served requests per time unit. It includes the physical equipment, power, network and administration price.

- VM Penalty: The corresponding penalty caused by violations of SLA. Violation occurs when actual run time exceeds pre-defined response time in SLA. For each request *r*, a linear utility function specifies the per request penalty $Penalty^r_{i,k,m}$ incurred by the corresponding average end-to-end response time *Rr*. If services provided by ISPs violate SLA terms, it has to pay for the penalty according to the clauses defined in the SLA.

Let *R* denote the number of SCs and *r* denotes SC's id. Let *M* denote the number of tiers for a typical multi-tier simulation application and *m* denotes tier id. Let *I* be the number of initiated VMs, and *i* indicates the VM id. Let *K* denote the types of VMs. Let n_m denote number of initiated VMs in tier *m*. $\Pr ofit^r_{i,k,m}$, $VMCost^r_{i,k,m}$ and $Penalty^r_{i,k,m}$ denote the profit, VM cost and penalty cost for serving request *r* using VM_{ikm} respectively. $useTime^r$ denotes the duration time used by SC *r*. $PriceServ^r$ denotes ISPs' charge from SC *r*. Our goal is to maximize the profit of ISPs. The global profit function can be formulated as:

$$\sum_{r=1}^{R} Profit^r_{i,k,m} = \sum_{r=1}^{R} PriceServ^r \times useTime^r - \sum_{r=1}^{R} Cost^r_{i,k,m} \quad (1)$$

where, $\forall r \in R, 1 \le i \le n_m, \forall k \in K, \forall m \in M$

Let $Cost^r_{i,k,m}$ indicate the cost for serving SC request *r* with $VM_{i,k,m}$. It depends on the VM cost ($VMCost^r_{i,k,m}$) and penalty ($Penalty^r_{i,k,m}$). It can be formulated as:

$$Cost^r_{i,k,m} = VMCost^r_{i,k,m} + Penalty^r_{i,k,m} \quad (2)$$

where, $\forall r \in R, 1 \le i \le n_m, \forall k \in K, \forall m \in M$

The VM cost depends on the VM type *k*, the price of VM *i* with type *k* in tier *m* ($PriceVM_{i,k,m}$), the initiation Time ($iniTimeSA^r_{i,k,m}$) and duration time of SC request *r* ($useTime^r_{i,k,m}$). The VM cost is defined as:

$$VMCost^r_{i,k,m} = PriceVM_{i,k,m} \times \left(iniTime^r_{i,k,m} + useTime^r_{i,k,m} \right) \quad (3)$$

where, $\forall r \in R, 1 \le i \le n_m, \forall k \in K, \forall m \in M$

The SLA violation penalty model is similar to other related utility functions [14, 15]. The penalty $Penalty^r_{i,k,m}$ function penalizes the ISPs by reducing the utility.

The resource allocation problem in question is how to dynamically allocate the CPU among VMs with the goal of maximizing the global profit function. In this paper, we adjust VM allocation strategy for virtualized simulation environments.

3 Virtual Machine Allocation Strategy

The main objective of our work is to maximize the profit for ISPs by minimizing the cost of VMs using an effective virtualized resource allocation strategy. To achieve this objective, we propose and examine a 2-step virtualized resource allocation algorithm (*HoriVerScale*) which allows a cost effective usage of resources in cloud simulation environment to satisfy dynamically changing SCs' requirements in line with SLAs.

3.1 Allocation Strategy

In cloud simulation environment, SCs with different performance levels may request services of the same simulation application. In this case, SCs in each tier of a typical multi-tier simulation application have different performance levels. In order to serve these different requests in each tier, VM with different capacities should be allocated to requests from different performance levels, e.g., requests with the highest level should be served by VM with the highest capacity. In this way, the performance requirement of corresponding requests can be satisfied. What's more, SCs with the same performance level may request services in the same tier of a multi-tier simulation application. In this case, the same VM with enough capacity can be reused and shared by different SCs of this level.

For ISPs, initiation of a new VM is more costly than reuse of initiated VM. And this paper considers that resizing of VMs is much cheaper than starting a new VM. There are existing possible solutions for vertical scaling including migrating a virtual machine to another physical server or dynamically increasing the virtualized resource allocated to the virtual machine. So we assume that the available number of VMs in each tier is limited, denoted as C_m. Let C_{mg}, C_{ms} and C_{mb} denote upper limit of gold type, silver type and bronze type in mth tier respectively. Note that C_m is the sum of C_{mg}, C_{ms} and C_{mb}.

The next problem for ISPs comes that how to accurately determine C_{mg}, C_{ms} or C_{mb} in mth tier for three kinds of SCs. As shown in Fig. 2, when the number of initiated VMs for requests from specific consumer type in mth tier reaches corresponding limit, e.g. C_{mg} for gold type, to guarantee corresponding performance requirements while minimizing VMs costs. The VM with the maximum remaining capacity will be scaled vertically by a certain amount, denoted as $sv\%$, which assures that the vertically scaled VM can exactly satisfy requirement of that requests. Note that when the sum of remaining and scaled resources in a VM reaches the threshold (85%) of physical machine, the VM should be migrated to another physical server. In this way, new arrival requests can always be served by a certain VM.

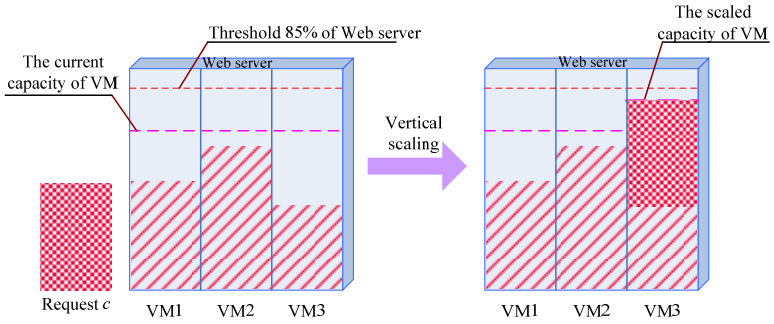

Fig. 2. An example of VM vertical scaling for gold consumers in Web tier, where C_{mg} is set 3.

In the proposed 2-step algorithm *HoriVerScale*, we firstly propose the algorithm to specify the upper limit of available VMs number for a specific SC in mth tier including C_{mg}, C_{ms} and C_{mb}. Then in order to serve new arrival requests, we propose

the algorithm to choose a suitable VM and specify the amount of the added virtualized resource. In this way, a specific VM can be vertically scaled instead of initiating a new VM. Altogether, the 2-step algorithm firstly horizontally scales VMs to initiate more VMs until the corresponding limit of number of VMs for a specific kind of SCs in *m*th tier is reached. And secondly the algorithm vertically scales the selected VM by increasing virtualized resources or migrating to another physical server instead of initiating new VMs.

3.2 Proposed Algorithm

The algorithm firstly determines the upper limit of initiated VMs. The requests arrival rate of three kinds of SCs can be determined respectively by online monitoring. Together with known different capacities of VMs, the algorithm can determine minimum number of VMs to assure that total processing capacity of VMs is no less than total requests arrival rate. Then the algorithm checks whether the performance requirement specified in SLA is violated. If so, new VMs will be initiated until SCs' performance requirement can be satisfied. The final initiated number of VMs for specific SCs in *m*th tier can be set as C_{mg}, C_{ms} and C_{mb} respectively. Algorithm 1 describes the proposed *HorizontalScaleLimit* algorithm, which is the first step of 2-step algorithm.

Algorithm 1. *HorizontalScaleLimit*
Input: requests arrival rate for a specific kind of SCs in *m*th tier (e.g. $\lambda_{m,g,i}$ for gold type) and processing capacity of corresponding VM, $\mu_{m,g,i}$, i=1,2,…,$C_{m,g}$
Output: The upper limit of initiated VMs for a specific kind of SCs in *m*th tier, e.g., C_{mg} for gold type.
Function:
HorizontalScaleLimit ($\lambda_{m,g}$, $\mu_{m,g}$){
1 for m=1 to M do
2 C_{mg}=1
3 end for
4 for m=1 to M do
5 $\mu_{m,g} = \sum_{i=1}^{C_{m,g}} \mu_{m,g,i}$
6 while($\frac{\lambda_{m,g}}{\mu_{m,g}} \geq 1$) do
7 $C_{m,g} = C_{m,g} + 1$
8 $\mu_{m,g} = \sum_{i=1}^{C_{m,g}} \mu_{m,g,i}$.
9 end while
10 end for
11 for m=1 to M do
12 while(performance requirement can **NOT** be satisfied)

```
13          C_{m,g} = C_{m,g} + 1
14      end while
15  end for
```

Then the second step of 2-step algorithm is described as follows. The algorithm checks the request type of SC *c*. According to the request type, the algorithm finds the VM_i with type *t* (VM_{it}) that can satisfy the performance requirement of the request. Then, it checks whether there is already initiated VM_{it} as SC *c* requests. If there is an initiated VM_{it}, then the algorithm checks whether this VM_{it} has enough capacity to serve the request of SC *c* according to requested resource on this VM. If there are more than one VM_{it} with enough available capacity to serve the request *c*, then the request *c* is assigned to the machine with minimum available capacity. If there is no initiated VM with type *t*, a new VM of corresponding type is initiated.

When the number of initiated VMs for requests from a specific SC' type in *m*th tier reaches corresponding limit, e.g. C_{mg} for gold type, in order to serve new arrival requests, no more VMs will be initiated and the VM with the maximum remaining capacity will be scaled vertically by a certain amount to assure that the vertically scaled VM can exactly satisfy requirement of that requests. Note that when the sum of remaining and scaled resources in a VM reaches the threshold (85%) of physical machine, the VM should be migrated to another physical server. Otherwise, virtualized resources can be directly allocated to vertically scale the VM. In this way, new arrival requests can always be served by a certain VM. Algorithm 2 describes the *VerScaleLimit* algorithm.

Algorithm 2. ***VerScaleLimit***

Input: request *c*

Output: VM serving request *c*

Function:

```
VerScale (c){
1   if(there is no initiated VMs which can satisfy requirement of request c) {
2     initiate new VM with corresponding type as request c required
3   }
4   else if(there is initiated VM_i with type k which matches to the VM type
          requested by c){
5         if(the number of initiated VMs for requests from a specific SC' type
            in mth tier don't reach corresponding limit){
6           initiate new VM with corresponding type as request c required
7         }else{
8             for each initiated VM_i with type k (VM_{i,k}) {
9               if (VM_i has enough capacity required by request c) {
10                  put VM_i into vmList
11              }
11            }
13            if(VMmax is not empty){
14              schedule request c to VMmin, which has minimum remaining
                capacity
15            }else{ //vertical scaling
```

```
16                    select VM with maximum remaining capacity, VMvs
17                    if(scaled VM does not reach the threshold of physical
                        machine){
18                      scale VM by increasing virtualized resource and assure
                        that the VM can exactly serve request c.
19                    }else{
20                      migrate the VM to another physical machine
                          with available virtualized resources
21                    }
22                }
23            }
24    }
25 }
```

ISPs can maximize the profit by minimizing the resource cost, which depends on the number and type of initiated VMs. Therefore, this 2-step algorithm *HoriVerScale* is designed to minimize the number of initiated VMs by utilizing already initiated VMs.

4 Performance Evaluation

This section presents the performance results obtained from an extensive set of experiments. The proposed VM allocation algorithm, *HoriVerScale* have been evaluated and compared with the existing algorithm, *MinRemCapacity* [16]. *MinRemCapacity* also reuse initiated VM to serve arrival requests, but it initiates new VMs when existing initiated VMs cannot satisfy requirement of requests instead of scaling existing VMs. In this section, we firstly describe our experiment setting, and give the analysis of experimental results.

4.1 Experimental Setting

We evaluate our proposed algorithm for VM allocation in our cloud simulation platform consisting of six heterogeneous physical machines. To decrease the number of records and keep the variation characteristic, we take a sample from every 120 requests. The request arrival rates vary with time from 0 minute to 1440 minutes. All the parameters used in the experiments study are given in Table 1.

Table 1. SLA Characteristics for Simulation Application in Cloud Simulation Platform.

SC Type	VM Type	Response Time Threshold(msec)	Price($/hour)
Gold	Large	200	0.6
Silver	Medium	250	0.35
Bronze	Small	300	0.2

4.2 Effectiveness of Proposed Algorithm

In this section, we compare its performance results from different perspectives. Fig. 3 shows the variation of average number of initiated VMs in three tiers including web tier, application tier and database tier of our simulation application. As illustrated, the number of initiated VMs with *HoriVerScale* algorithm is less than *MinRemCapacity* algorithm in all 3 tiers. The comparison of number of total initiated VMs shows that proposed *HoriVerScale* algorithm can reduce nearly 13% of initiated VMs. So the ISPs can reduce cost by using less initiated VMs while satisfying SCs' performance requirement.

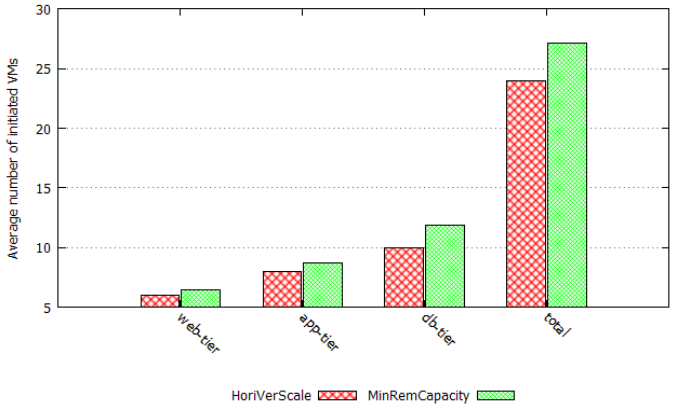

Fig. 3. Comparison of average number of initiated VMs.

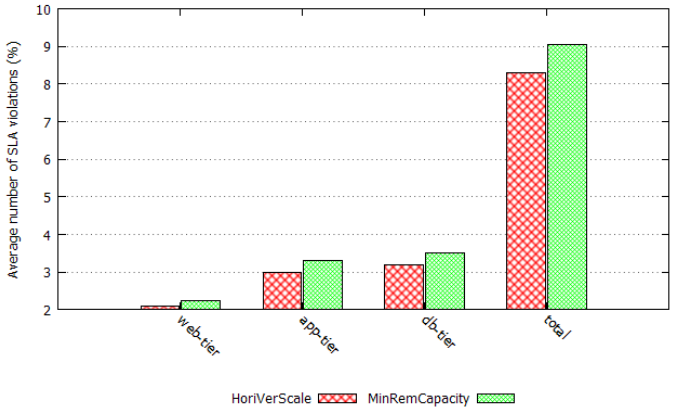

Fig. 4. Average percentage of SLA Violations.

Fig. 4 shows the average percentage of SLA Violations using *HoriVerScale* and *MinRemCapacity* algorithms. As illustrated, *HoriVerScale* algorithm shows better performance, response time of which is smaller than that of *MinRemCapacity* algorithm in average. So SLA violations in 3 tiers can be all reduced with *HoriVerScale* algorithm. In addition, the comparison of number of total SLA violations shows that proposed *HoriVerScale* algorithm can reduce nearly 9.5% of SLA violations.

Fig. 5 shows comparison of the reduced total cost using *HoriVerScale* and *MinRemCapacity* algorithms. In average, the algorithm *HoriVerScale* performs better to reduce total cost by using less VMs compared with *MinRemCapacity* algorithm. Because it costs less with less or equal number of VMs but generates less number of SLA violations. So, during the variation of arrival rate, the *HoriVerScale* algorithm can reduce the SLA violations and total cost in the context of resource sharing.

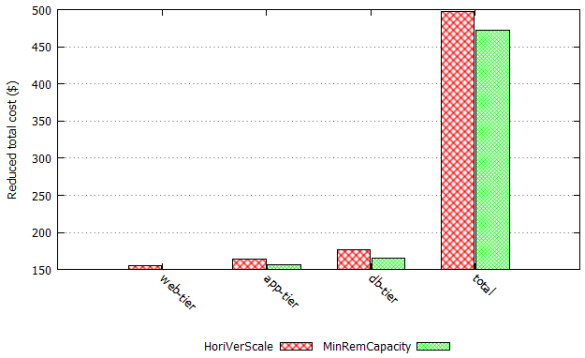

Fig. 5. Reduced total cost using *HoriVerScale* and *MinRemCapacity* algorithms.

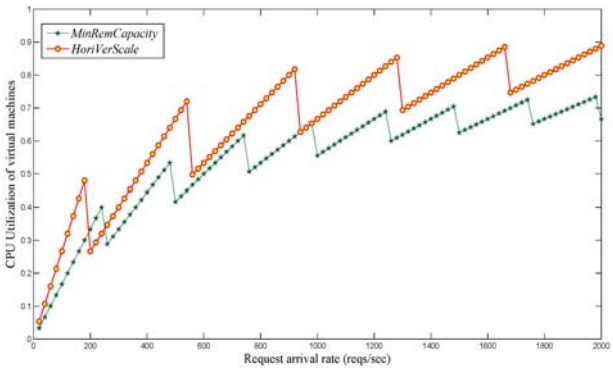

Fig. 6. CPU utilization of initiated virtual machines.

Fig. 6 shows comparison of average CPU utilization of initiated virtual machines in our simulation application. Based on better VM allocation mechanism, during the

variation of arrival rate from 0 to 2000, as illustrated, the *HoriVerScale* algorithm performs better and can assure higher average CPU utilization than *MinRemCapacity* algorithm.

As shown in Fig. 7, it shows the throughput variation of the RUBiS system as request arrival rates vary with time. Above 80% of arrival requests can be served by either *HoriVerScale* algorithm or *MinRemCapacity* algorithm at any time. At some time, 90% of arrival requests can be served. This shows that both algorithms can effectively serve most of arrival requests. However, compared with the *MinRemCapacity* algorithm, the *HoriVerScale* algorithm shows superior performance and corresponding throughout is slightly higher at most of time from 0min to 2000min. The shows that *HoriVerScale* algorithm can serve more requests and can effectively utilize VM resources.

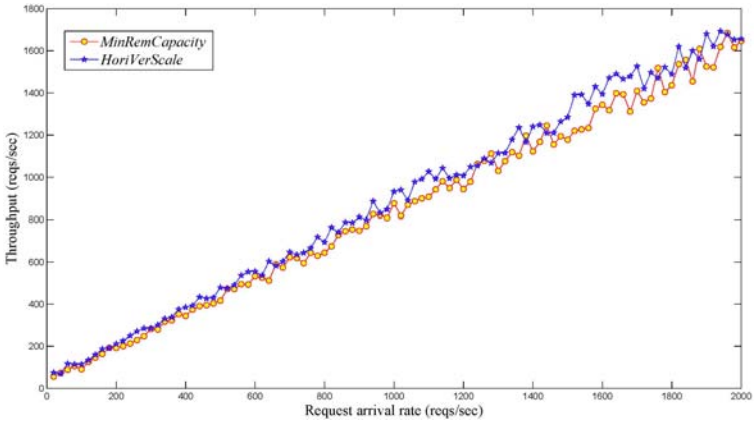

Fig. 7. Throughput of the system using *HoriVerScale* and *MinRemCapacity* algorithms.

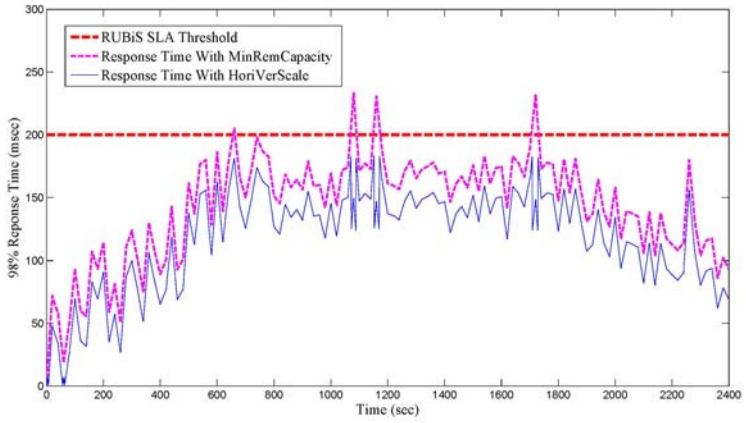

Fig. 8. 98% of the average response time using *HoriVerScale* and *MinRemCapacity*.

As shown in Fig. 8, it shows the 98% of the average response time variation using *HoriVerScale* and *MinRemCapacity* algorithms. This only shows SLA threshold for consumers with gold level, which is set 200msec. As illustrated, though *MinRemCapacity* algorithm shows great performance at most of the time, in some time, response time of *MinRemCapacity* algorithm violate SLA threshold. However, *HoriVerScale* algorithm shows better performance, response time of which is smaller than that of *MinRemCapacity* algorithm at most of the time. In addition, *HoriVerScale* algorithm can also assure that response time can be in the limit of corresponding SLA threshold.

5 Conclusion

This paper focuses on the problem of virtualized resource allocation for SA with the goal of maximizing the SLAs revenue while minimizing energy costs in cloud simulation platform. According to the performance metrics specified in the SLA, the system profit model is proposed. Based on this model, in order to dynamically allocate resource to minimize SLA violations and infrastructure cost, a VMs allocation algorithm is proposed and compared with the existing algorithm. The experimental evaluation with a realistic workload in cloud simulation platform demonstrates the feasibility of the algorithm. This allows a cost effective usage of resources in cloud. In future work, we will look into the simulation resources and capabilities sharing on demand for mass users. We would also focus on the application research of cloud simulation system.

Acknowledgment

This work was supported in part by a grant from the China Postdoctoral Science Foundation (No. 2014M550068).

References

1. Barham P., Dragovic B., raser K., Hand S., Harris T., Ho A., Neugebauer R., Pratt I., Warfield A.: Xen and the Art of Virtualization. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles, SOSP 2003, pp. 177. Bolton Landing, NY, USA, (2003)
2. Armbrust M., Fox A., Griffith R., et al: Above the Clouds: A Berkeley View of Cloud Computing. Technical Report No. UCB/EECS-2009-28, University of California Berkley, USA, Feb. 10 (2009)
3. Buyya R., Yeo C.S., Venugopal S., et al: Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems, 25(6), pp. 599-616. Elsevier science, Amsterdam, the Netherlands (2009)

4. Bo Hu Li, et al.: A networked modeling and simulation platform based on the concept of cloud computing “Cloud Simulation Platform”. Journal of System Simulation. 12, pp. 5292-5299. (2009)
5. Bo Hu Li, Xudong Chai, et al: New Advances of the Research on Cloud Simulation. Proc. of AsiaSim. (2011)
6. Aoun R., Doumith E. A., Gagnaire M.: Resource Provisioning for Enriched Services in Cloud Environment. In: Proceedings IEEE CloudCom Conference, pp. 296-303. (2010)
7. Yazir Y.O., Matthews C., Farahbod R., Neville S., Guitouni A., Ganti S., Coady Y.: Dynamic Resource Allocation in Computing Clouds using Distributed Multiple Criteria Decision Analysis. In: Proceedings IEEE CLOUD Conference, pp. 91-98. (2010)
8. Bi, J., Zhu Z.L., Tian R.X., Wang Q.B.: Dynamic Provisioning Modeling for Virtualized Multi-tier Applications in Cloud Data Center. IEEE 3rd International Conference on Cloud Computing, pp. 370-377. Miami, USA, (2010)
9. Fu Y., Vahdat A.: SLA Based Distributed Resource Allocation for Streaming Hosting Systems, http://issg.cs.duke.edu
10. Yarmolenko V., Sakellariou R.: An Evaluation of Heuristics for SLA Based Parallel Job Scheduling. In: Proceedings of the 3rd High Performance Grid Computing Workshop (in conjunction with IPDPS 2006). Rhodes, Greece (2006)
11. Lee Y.C., Wang C., Zomaya A.Y., Zhou B.B.: Profit-driven Service Request Scheduling in Clouds. In: Proceedings of the International Symposium on Cluster and Grid Computing, (CCGrid 2010), Melbourne, Australia (2010)
12. White S.R., Hanson J.E., Whalley I., et al.: An architectural approach to autonomic computing. In: Proceedings of the International Conference on Autonomic Computing (2004)
13. Kephart, J.O. Chess D.M.: The vision of autonomic computing, IEEE Computer. 36 (1), pp. 41–50. (2003)
14. Walsh W.E., Tesauro G., Kephart J.O., Das R.: Utility Functions in Autonomic Computing. In: Proceedings of the IEEE International Conference. Autonomic Computing (ICAC’04), May pp. 17–18. New York, NY (2004)
15. Bennani M.N., Menasc´e D.A.: Resource Allocation for Autonomic Data Centers Using Analytic Performance Models. In: Proceedings of the IEEE International Conference on Autonomic Computing, June pp.13-16. Seattle, WA (2005)
16. Wu L.L., Garg S.K., Buyya R.: SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments. In: Proceedings of the 11th IEEE/ACM International Conference Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011), May pp. 195–204. (2011)