Workload Predicting-Based Automatic Scaling in Service Clouds

Jingqi Yang, Chuanchang Liu, Yanlei Shang, Zexiang Mao, Junliang Chen
State Key Lab of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
Email: {yangjingqi,lcc3265,shangyl,zxiangmao,chjl}@bupt.edu.cn

***Abstract*—Service platforms have disadvantages such as they have long construction periods, low resource utilizations and isolated constructions. Migrating service platforms into clouds can solve these problems. The scalability is an important characteristic of service clouds. With the scalability, the service cloud can offer on-demand capacities to different services. In order to achieve the scalability, we need to know when and how to scale virtual resources assigned to different services. In this paper, a linear regression model is used to predict the workload. Based on this predicted workload, an auto-scaling mechanism is proposed to scale virtual resources at different resource levels in service clouds. The automatic scaling mechanism combines the real-time scaling and the pre-scaling. Finally experimental results are provided to demonstrate that our approach can satisfy the user SLA while keeping scaling costs low.**

I. Introduction

Based on web services, the service platform has become a very popular way to provide services. But service platforms have disadvantages such as they have long construction periods, low resource utilizations and isolated constructions. Moreover it may be too late to add infrastructures to deal with sudden surge requests of a service platform. In these cases, migrating service platforms into clouds can solve these problems. Service clouds are proposed to quickly deploy communication services in clouds, and they may be good choices when we need a service platform. With the scalability of service clouds, service providers need not be concerned for over-provisioning or under-provisioning [1].

The cost of a service cloud will be less if we lease less virtual resources from the cloud provider, but performance will be affected when the peak load occurs. Conversely, leasing more virtual resources leads to a performance improvement, but also bears a higher cost. Catering to the user Service Level Agreement (SLA) while still keeping costs low is challenging for service clouds primarily due to the frequent variation of platform workloads. With such a problem there should be an universal method to predict the platform workload, and then virtual resources can be added and released depending on this predicted workload.

Generally, there are two kinds of scaling technologies at different resource levels in service clouds. The first one is the horizontal scaling which adjusts the amount of Virtual Machine (VM) instances. Though this method provides a larger scale resource, but it incurs a considerable waste of resources sometimes. Beyond that, the horizontal scaling takes several minutes to boot a VM, and the new VM cannot be used at once. The second one is the vertical scaling which is implemented by changing the partition of resources (e.g. CPU, memory, storage, etc.) inside a VM [2]. Modern hypervisors support on-line VM resizing without shutting it down. The vertical scaling can scale virtual resources in a few milliseconds, but it is limited by the amount of available resources on the physical machine hosting the VM. An intelligent combination of the horizontal scaling and the vertical scaling may help us find a optimal scaling strategy.

The main contributions of this work are as follows: (1) We propose a linear regression model to predict the workload of service clouds; (2) We present an automatic scaling approach which combines the real-time scaling and the pre-scaling, and we consider three scaling techniques: self-healing scaling, resource-level scaling and VM-level scaling; (3) Our experiment results show that our auto-scaling approach can meet the user SLA with less costs.

II. Related Work

Some researches are carried out about service clouds. REMICS [3] defines the methodology and tools to deploy services in clouds. It points out that for taking advantages of service cloud computing technologies, the quality requirements such as scalability become important. But there are not so many researchers working on the scalability of service clouds.

It is necessary to predict workloads of service clouds for better performance. A number of different methods are used in the workload prediction. In [4], it develops a model-predictive algorithm for the workload prediction in which a second order autoregressive moving average method filter is used. In [5], a two-step prediction approach and a hot spot detection model are proposed. PRESS [6] uses a pattern matching and state-driven approach to predict workloads. It first employs signal processing techniques to identify repeating patterns. If no signature is discovered, PRESS employs a statistical state-driven approach, and uses a discrete-time Markov chain to predict the demand for the near future.

Researchers have investigated virtual resource scaling methods at different levels, such as the horizontal scaling and the vertical scaling. There are several projects have been

978-0-7695-5028-2/13 $26.00 © 2013 IEEE
DOI 10.1109/CLOUD.2013.146

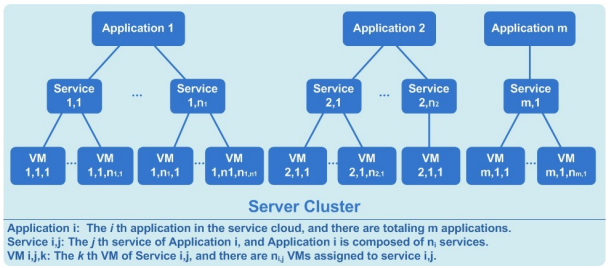

Fig. 1. Logical architecture of server cluster

produced based on the horizontal scaling. In [4], it proposes an efficient auto-scaling method based on the horizontal scaling to help cloud service providers in operating data centers. WebScale which is developed for Web applications to provide on-demand resources in form of VM [7]. Along with the development of the virtualization technology, more and more researchers are aware of advantages of the vertical scaling. SmartScale [8] uses a combination of the vertical scaling and the horizontal scaling to optimize both the resource usage and the reconfiguration cost.

A lightweight approach [9] is proposed to enable the elasticity for cloud applications. In this work scaling methods are divided into three categories: self-healing scaling, resource-level scaling and VM-level scaling. The first two methods are vertical scaling methods, and the last one is a horizontal scaling method. Our work also scales virtual resources at these three different levels. The general ides of the self-healing scaling is that if two VMs of a service are allocated in the same cluster node, resources of VMs may complement each other. For example, one or more CPUs allocated to a VM with a low CPU utilization can be removed to another VM which has already saturated existing CPU resources. The resource-level scaling up uses unallocated resources available at a particular cluster node to scale up a VM executing on it. So for example, a cluster node with low CPU and memory utilizations can allocate these types of resources to one of VMs executing on it thus scaling it up. Both of them can scale up and down virtual resources within milliseconds.

III. Auto-Scaling Challenges and Solutions

In this section we present a scenario of a service cloud, and illustrate why we use the workload prediction and a combination of the horizontal scaling and the vertical scaling to achieve auto-scaling. Our approach is generic, and it can work well in most cloud environments.

Figure 1 shows the logical architecture of the server cluster in a service cloud. There are several different applications in the server cluster. Each application is composed of one or more services. Services are loosely coupled, and each of them runs on one or more VMs. It should be stressed here that services rather than applications run on VMs directly, so we consider the scaling approach of services rather than applications in the remainder of this paper. VM workers have a variety of processing capabilities, e.g. small, medium and big in Amazon EC2 [10]. There is only a service running on a VM for clarity in this service cloud, and it can be extended to run more than one services on a VM.

A. Predicting Workloads

The server cluster should scale resources dynamically according to platform running states. When the system detects a higher system utilization exceeding the upper threshold, the horizontal scaling or the vertical scaling can be executed. But the horizontal scaling will take a while to complete and the vertical scaling is limited by scale, these two situations all produce a bad effect on system performance. As a result, it is desirable if the platform can be scaled up earlier than the time when the workload actually increases. So we need an approach to predict workloads of services, and workloads of different services are predicted independently in service clouds. Section 4 describes our workload prediction model.

B. Dealing with the Predicted Deviation

Because of the abnormal and burst of workloads in service clouds, there may be a predicted deviation. If the predicted deviation is not handled in time, the platform performance will be affected. We execute the real-time scaling in order to minimize the impact of this deviation. In our approach, when the platform utilization goes beyond the threshold, the vertical scaling will be applied to add virtual resources every interval. Profiting from the short time the virtual scaling takes, this approach is effective.

C. Scaling with Lower Costs

Service providers want to keep cloud costs low besides satisfying the user SLA, so we must consider carefully how to scale the service cloud with a lower cost. The vertical scaling cost and the horizontal scaling cost differ in both the unit price and the license fee, and there are an exponential number of possible strategies which may combine the vertical scaling and the horizontal scaling. For the purpose of avoiding resource wastes and reducing computational overheads, we use a greedy approach to achieve the optimization goal.

IV. Cloud Auto-Scaling Approach

In this section, we explain our cost-aware auto-scaling approach. This approach implements the auto-scaling of service clouds with a lower cost, and the scaling is based on workload prediction results and platform running states.

A. Workload Prediction Model

In our strategy, the workload prediction is used to predict the request number of services at the next interval. Because workloads of a service cloud are irregular, we need a method which can adjust its model quickly according to the variation trend of workloads. We also can observe that the workload trend is linear in a relatively short period of time. As a result of these, in this paper, we use a linear regression model (LRM) to solve this problem [11].

The linear regression model is the principal style of econometric models. It is used to study the relationship between a

TABLE I
RESULTS OF WORKLOAD PREDICTION

Metric (%)	LRM	ARMA	Mean	Max
Min predicted deviation	0	1	1	0
Max predicted deviation	72	99	166	288
Avg predicted deviation	9	12	17	20

dependent variable and an independent variable. The generic form of the linear regression model is

$$Y_i = \beta_1 + \beta_2 X_i \tag{1}$$

In our scenario, Y is the workload, X is the time, and i indexes the sample observation. The coefficients β_1, β_2 are determined by solving a linear regression equation based on previous workloads Y_{i-1}, Y_{i-2}, Y_{i-3} and so on. β_1, β_2 change with different previous workloads, that is to say this model can change with the workload trend. We use the Ordinary Least Squares to solve this equation, then we can get the results (2)(3).

$$\beta_1 = \frac{\sum X_i^2 \sum Y_i - \sum X_i \sum X_i Y_i}{n \sum X_i^2 - (\sum X_i)^2} \tag{2}$$

$$\beta_2 = \frac{n \sum X_i Y_i - \sum X_i \sum Y_i}{n \sum X_i^2 - (\sum X_i)^2} \tag{3}$$

β_1, β_2 can be calculated by substituting known previous workloads into (2) and (3), and then we can calculate the workload at next interval using (1). We can see that this method is also easy to calculate.

We compare predicted workloads with actual workloads, and a set of alternative workload prediction algorithms are also implemented for comparing. All of these methods use a sliding window of previous workloads monitored by the platform.

A second order autoregressive moving average method filter (ARMA) [4]. The equation for the filter used is given by $Y_{t+1} = \beta * Y_t + \gamma * Y_{t-1} + (1 - (\beta + \gamma)) * Y_{t-2}$. The value for the variables β and γ are given by the values 0.8 and 0.15.

Mean. The predicted workload is the mean workload over the workloads in the window.

Max. The predicted workload is the maximum workload over the workloads in the window.

We collect workloads of a video service for 6 hours every time interval in a service cloud, and workloads accord with regular rules in [12]. The time interval is set by the time spending on booting a VM, and the time interval is 5 minutes in our approach. This actual workload is compared with the other four predicted workloads including our approach LRM. Experimental results are shown in Figure 2 and Table I. We can see that in most situations our linear regression model outperforms other methods, and our LRM has a lowest prediction error. LRM can adjust its function with the workload trend, so it can predict the workload timely and accurately. By contrast, ARMA prediction is always dragging a little behind, Mean prediction is less sensitive of workload variations, and Max prediction is higher than the actual workload most of the time.

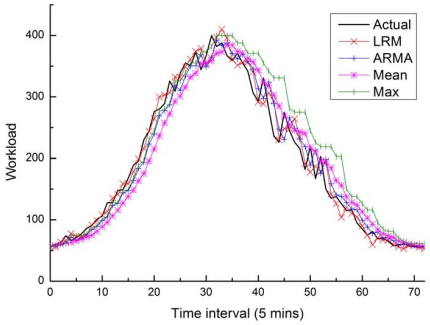

Fig. 2. Predicted and actual workloads

TABLE II
PARAMETERS OF THE AUTO-SCALING APPROACH

Parameter	Description
$n_i, j(t)$	Number of VMs of $s_{i,j}$ at the tth interval
$w_{i,j}(t)$	Number of requests of $s_{i,j}$ at the tth interval
$u_{i,j}(t)$	Utilization of $s_{i,j}$ at the tth interval
$u_{i,j,lower}$	Lower threshold of $s_{i,j}$
$u_{i,j,upper}$	Upper threshold of $s_{i,j}$
$license_{i,j}$	License of $service_{i,j}$
$r_{i,j,k}$	Resources allocated to the kth VM of $s_{i,j}$

B. Auto-Scaling Algorithm

In order to help understand this algorithm, Table II lists the main parameters used throughout this approach. The algorithm takes the scaling cost (SC) into consideration, so a cost model is proposed. A SC is composed of a virtual resource cost (VRC) and a license cost (LC).

$$SC = VRC + LC \tag{4}$$

VRC is the cost of scaled CPU, memory and other resources used by services (5). The self-healing scaling exchanges existing virtual resources, so it costs no VRC. We consider that each request of the same service consumes the same amount of virtual resources which is regard as a unit resource of this service. So we spend the same cost no matter where a unit resource adds in the resource-level scaling. There may be different types of VMs which have different capacities, and these VMs' VRC are depending on their size.

Another consideration is that we must pay for license fees if there are business softwares. Because in most cases one software requires only one license in a VM, and the self-healing scaling and the resource-level scaling do not change the number of VMs, hence these two scaling methods are free of the license charge in our approach. So LC is the sum of license costs of scaled VMs only in VM-level scaling (6).

$$VRC = \sum cost(r_{i,j,k}) \tag{5}$$

$$LC = \sum cost(license_{i,j}) \tag{6}$$

Pseudo code for the auto-scaling approach is given below. Algorithm 1 is started after the service cloud platform is de-

TABLE III
SYSTEM SCALABILITY MANAGEMENT

algorithm 1 System scalability management
1 Begin
2 Predict an initial workload, and boot VMs;
3 while (the system is running and in the beginning of a interval)
4 for(every $s_{i,j}$ in the cloud)
5 Monitor $w_{i,j}(t)$, $n_{i,j}(t)$, $u_{i,j}(t)$;
6 Real-time scaling at the tth interval;
7 Pre-scaling at the $(t+1)$th interval.
8 End

TABLE IV
REAL-TIME SCALING

algorithm 2 Real-time scaling at the tth interval
1 Begin
2 if($u_{i,j}(t) > u_{i,j,upper}$)
3 for(every VM of $s_{i,j}$)
4 if($u_{i,j}(t) > u_{i,j,upper}$)
5 Execute self-healing scaling up;
6 if($u_{i,j}(t) > u_{i,j,upper}$)
7 Execute resource-level scaling up;
8 else if($u_{i,j}(t) < u_{i,j,lower}$)
9 Execute virtual resource scaling down.
10 End

ployed, and it keeps running until there are no services running in the service cloud. This approach manages virtual resources of each service every time interval. When the platform is deployed, the algorithm predicts an initial workload of every service based on the experience, then boots an appropriate number of VMs for them (line 2). Services of our service cloud are scaled individually, and this can simplify the scaling problem (line 4). The algorithm first collects information of the number of requests, the number of VMs running this service and the average utilization of VMs (line 5), then it performs the real-time virtual resource management (line 6), and finally it pre-manages virtual resources at the next interval (line 7).

The real-time virtual resource management here has a twofold meaning. One is to keep the effect of prediction deviation to a minimum when the actual workload is larger than prediction, and the other one is to release unnecessary resources. This algorithm is shown in algorithm 2. If the average utilization of VMs assigned to the same service beyonds the upper threshold, VMs will be scaled up one after another until the average utilization bellows the upper threshold (line 4-7). In real-time environment scaling methods should be quick enough to respond to overloaded user requests. So we use two types of vertical scaling methods, and they are the self-healing scaling and the resource-level scaling (see [9] for detailed pseudo code of the self-healing scaling and the resource-level scaling).If the utilization is under the lower threshold, the algorithm triggers a virtual resource scaling down (line 9). This step reduces the cost of platform by releasing idle resources, and pseudo code is shown in algorithm 3.

In the algorithm 3, when the platform needs to release resources, first the algorithm releases a VM with the largest SC every time, and then it releases as many resource-level resources as possible under the upper threshold (line 4 and line 5). These two steps releases resources until releasing resources causes the utilization higher than the upper threshold.

TABLE V
VIRTUAL RESOURCE SCALING DOWN

algorithm 3 Virtual resource scaling down
1 Begin
2 if($u_{i,j}(t) < u_{i,j,upper}$)
3 Execute VM-level scaling down;
4 if($u_{i,j}(t) < u_{i,j,upper}$)
5 Execute resource-level scaling down.
6 End

TABLE VI
PRE-SCALING

algorithm 4 Pre-scaling at the $(t+1)$th interval
1 Begin
2 Predict $w_{i,j}(t+1)$;
3 Calculate $u_{i,j}(t+1)$;
4 if($u_{i,j}(t+1) > u_{i,j,upper}$)
5 Execute cost-aware pre-scaling up.
6 End

The approach completes the real-time scaling so far, then it pre-manages virtual resources based on the workload prediction. Algorithm 4 does not execute the virtual resource scaling down, because there may be prediction deviations. The resource scaling down only happens in the real-time scaling according to real-time states of VMs. Algorithm 4 predicts the workload at the next interval using the workload prediction model introduced in Section 4 (line 2), then based on this prediction it calculates the utilization of VMs (line 3). If the predicted utilization is higher than the upper threshold, the algorithm pre-scale up virtual resources (line 4 and line 5). With the purpose of reducing costs, it executes a scaling up strategy with smallest SC. In Section 4.3 how to find such a strategy will be expounded in detail.

C. Cost-Aware Pre-Scaling

The objective of pre-scaling up is to find a optimal combination of virtual resources at different resource levels to handle increased requests. The self-healing scaling correlates with the real-time state of VMs closely, so it is not be used in the pre-scaling step. Then this optimization problem can be transformed into an integer programming problem. It is formally defined as
minimize

$$SC = VRC_r + VRC_v + LC_v, \tag{7}$$

where

$$VRC_r = n_r * cost(resource) \tag{8}$$

$$VRC_v = \sum nVM_i * cost(VM_i) \tag{9}$$

$$LC_v = \sum nVM_i * cost(license), \tag{10}$$

subject to

$$n_r + \sum nVM_i * wVM_i > n \tag{11}$$

$$n_r < a_r \tag{12}$$

TABLE VII
COMPONENTS OF THE COST PROBLEM

Component	Description
VRC_r	VRC of resource-level scaling
VRC_v	VRC of VM-level scaling
LC_v	LC of VM-level scaling
n	Number of requests
n_r	Number of increased resource units
VM_i	Type i VM
nVM_i	Number of VM_i
wVM_i	Number of requests a VM_i can process
a_r	Number of available resource units
cost(VM_i)	Cost of the resource of VM_i
cost(resource)	Cost of a unit resource
cost(license)	Cost of a license

TABLE VIII
COST-AWARE PRE-SCALING UP

algorithm 5 Cost-aware pre-scaling up

```
1  Begin
2  Execute VM-level scaling up;
3  if(n > 0)
4     if(a_r < n)
5        Scale up the smallest VM;
6     else
7        Execute resource-level scaling up or VM-level scaling up.
8  End
```

Table VII describes the components of this problem.

In the pre-scaling step, a scaling cost is the sum of three parts (7). The first part is VRC_r which is the product of the number of resource units and the cost of a unit resource (8). There is no LC in the resource-level scaling as shown in Section 4. The second part is costs of every VM (9). Costs of different types of VMs vary with their capacities, and usually costs increase in proportion to capacities. The third part is LC_v, and it is the sum of LC of every VM (10). Constraint (11) guarantees that resources to be increased can satisfy user requests. Constraint (12) indicates resources to be increased must be available in this cluster node.

This problem is an NP-hard problem, and it cannot get the optimal solution in polynomial time. In order to reduce computational overhead, we can calculate the near-optimum solution with a heuristic algorithm. Then a greedy approach is used, as shown in algorithm 5. Based on the experience and information we gather from other works, the VM-level scaling costs less than the resource-level scaling per unit resource. So the VM-level scaling is conducted first (line 2). According to the analysis of VRC_v and LC_v, if a VM is full-load running, the cost per request is lower when the capacity is larger. So first the algorithm finds a VM with the largest capacity which can be made full use by user requests, it repeats this until the remaining user requests cannot make full use of any type of VMs. Then if there are still remaining user requests to be handled (line 3), the algorithm performs the next step. If there are not enough resources at resource-level to handle remaining requests, the algorithm boots a VM with the smallest capacity (line 5). Conversely, if available resources are sufficient (line 6), the algorithm makes a comparison of costs between the resource-level scaling and the VM-level scaling with the smallest capacity, and it chooses a cheaper strategy to execute (line 7).

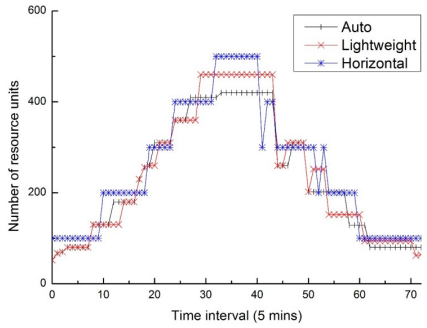

Fig. 3. Resource usages at different intervals

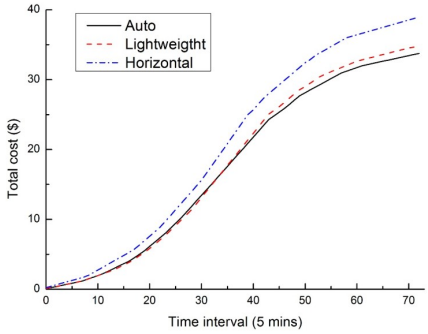

Fig. 4. Cumulative scaling costs at different intervals

V. EVALUATION

A. Experimental Setup

The experimental evaluation is designed to illustrate the effectiveness of our approach in enabling the cost-aware scaling in service clouds. The experiment is carried out on the CloudSim cloud simulator [13]. Using Amazon EC2 [10] as a reference, three types of VMs are provided: small, medium and big. They have different capacities and different virtual resource costs. There are two applications in the experimental service cloud, and each of them is composed more than one services. The first application is a Social Network Sites, and it is used to share pictures and blogs with other people, post comments, send E-mail and so on. The other one is a video site which can be used to upload and download videos.

Experiments compare our approach with two baseline methods. The first strategy adds fixed size VMs whenever the existing VMs are fully utilized, which is labeled "Horizontal scaling". The second strategy uses the method in [9] which is labeled "Lightweight scaling" which also scales resources at three levels. Our approach is labeled "Auto-scaling".

B. Evaluation Results

Figure 3 shows resource usages of three approaches. From the results we can see that the horizontal scaling consumes

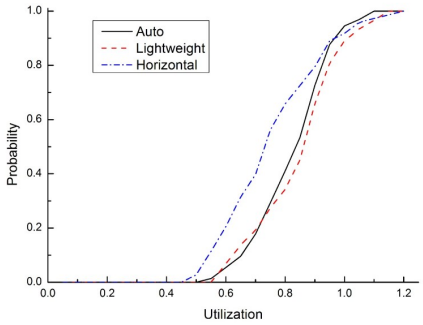

Fig. 5. CDF of utilizations

the most resource for most intervals. Because it scales virtual resources in VM unit, it cannot scale resources in a fine-grained unit. Figure 4 shows cumulative scaling costs of three approaches at different intervals. The horizontal scaling is also the worst one, and it costs more than the auto-scaling approach by about 13%. The lightweight scaling and our auto-scaling both scale virtual resources at different levels and take costs into consideration, as a result of these in most intervals these two approaches consume similar resources. Even so, our approach costs 3% less than the horizontal scaling.

The lightweight scaling and the auto-scaling are similar in resource using and costs, but the latter achieves a lower SLA violation which is more essential to service providers. When the utilization of a VM is in a state of full load running, it cannot handle additional requests. Then the request will be dropped, and it will cause SLA violations. Comparing with the auto-scaling, the lightweight scaling incurs a higher SLA violation than the auto-scaling. Three CDFs of utilizations are shown in Figure 5. Situations of request dropped are observed at 11% of intervals in the horizontal scaling, and the rate is 9% in the lightweight scaling. Because these two approaches scale virtual resources in real-time, sometimes it is too late to scale a VM up to satisfy increased requests. The auto-scaling pre-scales resources based on workload predictions beforehand, and it encounters the problem of request dropped at only 5% of intervals. In the worst situation, 10% of requests are dropped in our approach at an interval, and the percentage is close to 20% in the other two methods. So our approach can satisfy requests better than others.

Above all, we have study scaling costs, resource usages and utilizations of the auto-scaling and two comparison approaches in a service cloud. We observed that the auto-scaling outperforms the others. Based on the workload prediction, the auto-scaling approach can scale virtual resource dynamically at a lower cost and satisfy more user requests.

VI. Conclusion

In this paper, we investigate the problem of cost-aware auto-scaling along with predicted workloads in service clouds. We use a linear regression model to predict the workload, and we propose an approach to scale the service cloud platform in both the real-time scaling and the pre-scaling. We formulate the pre-scaling problem as an integer programming problem and present a greedy heuristic to solve it. In order to implement these mechanisms, a cloud scaling architecture is presented to support our approach. According to the experiment results, our approach predicts more accurately, costs less and has a lower user SLA violation. Our approach is also generic, and it can be used well in most service cloud scenarios.

Acknowledgment

This work is supported by 973 program of National Basic Research Program of China (Grant No. 2011CB302506, 2011CB302704), National Natural Science Foundation of China (Grant No. 61132001, 61001118, 61171102), Program for New Century Excellent Talents in University (Grant No. NCET-11-0592), the National Key Technology Research and Development Program of China "Research on the mobile community cultural service aggregation supporting technology" (2012BAH94F02), and the Novel Mobile Service Control Network Architecture and Key Technologies (Grant No.2010ZX03004-001-01).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, Above the clouds: A berkeley view of cloud computing, EECS Dpartment, Unversity of Califonia, Berkeley, Tech. Rep. UCB/EECS–2009–28, 2009.

[2] W. Wang, H. Chen, X. Chen, An Availability-aware Approach to Resource Placement of Dynamic Scaling in Clouds, Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing, pp. 930–931, 2012.

[3] REMICS. Availaible on: http://www.remics.eu/.

[4] N. Roy, A. Dubey, A. Gokhale, Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting, Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, pp. 500–507, 2011.

[5] P. Saripalli, G. Kiran, R. Shankar, H. Narware, N. Bindal, Load prediction and hot spot detection models for autonomic cloud computing, Proceedings of the 2011 4th IEEE International Conference on Utility and Cloud Computing, pp. 397–402, 2011.

[6] Z. Gong, X. Gu, J. Wilkes, PRESS: PRedictive Elastic ReSource Scaling for Cloud Systems, Proceedings of the 2010 International Conference on Network and Service Management, pp. 9–16, 2010.

[7] C. C. Lin, J. J. Wu, J. A. Lin, L. C. Song, P. Liu, Automatic Resource Scaling Based on Application Service Requirements, Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing, pp. 941–942, 2012.

[8] S. Dutta, S. Gera, A. Verma, B. Viswanathan, SmartScale: Automatic application scaling in enterprise clouds, Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing, pp. 221–228, 2012.

[9] R. Han, L. Guo, M. M. Ghanem, Y. Guo, Lightweight Resource Scaling for Cloud Applications, Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 644–651, 2012.

[10] Amazon elastic compute cloud. Availaible on: http://aws.amazon.com/ec2/.

[11] B. H. Baltagi, Econometrics, pp. 41–69. Springer Berlin Heidelberg, Berlin, 1998.

[12] J. A. Dilley, Web Server Workload Characterization. Hewlett-Packard Laboratories Report, HPL–96–160, 1996, Availaible on: http://www.hpl.hp.com/techreports/96/HPL-96-160.html.

[13] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisoning algorithms, Software: Pratice and Experience, Vol. 41, No. 1, pp. 23–50, 2011.