Inf Syst Front (2014) 16:7–18
DOI 10.1007/s10796-013-9459-0

A cost-aware auto-scaling approach using the workload prediction in service clouds

Jingqi Yang · Chuanchang Liu · Yanlei Shang · Bo Cheng · Zexiang Mao · Chunhong Liu · Lisha Niu · Junliang Chen

Published online: 6 October 2013
© Springer Science+Business Media New York 2013

Abstract Service clouds are distributed infrastructures which deploys communication services in clouds. The scalability is an important characteristic of service clouds. With the scalability, the service cloud can offer on-demand computing power and storage 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 novel service cloud architecture is presented, and 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 auto-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 Service Level Agreement (SLA) while keeping scaling costs low.

Keywords Service cloud · Scalability · Workload prediction · Cost-aware

1 Introduction

Web service is the major standard to realize SOA, and it encapsulates business processes into services. Based on this technology, the service platform has become a very popular way to provide services to users. But service platforms have disadvantages such as they have long construction periods, low resource utilizations and isolated constructions. At the initial stage of a service platform, it is difficult to estimate the traffic of services to deploy physical infrastructures. Moreover it may be too late to add infrastructures to deal with sudden surge requests of a service platform. In these cases, cloud computing is a good choice which liberates service providers from deploying physical infrastructures.

Cloud computing is an attractive technology, and it has achieved great commercial success in recent years. Many cloud providers, such as Amazon EC2 (http://aws.amazon.com/ec2/) and Google App Engine (http://ec.europa.eu/research/fp7/index_en.cfm), provide on-demand computing power and storage capacities dynamically which is regarded as scalability.

Service clouds are distributed infrastructures which are designed to facilitate rapid prototyping and deployment of adaptive communication services in clouds (Samimi et al. 2007), and they may be good choices when service platforms' workloads are dynamic or they need a lot of resources. With the scalability of service clouds, service providers don't need to be concerned for over-provisioning for a service, thus wasting costly resources, or under-provisioning, thus missing potential customers and the revenue (Armbrust et al. 2009).

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, when resources are used appropriately, leasing more virtual resources may leads to a performance improvement if the cloud is resource-constrained, but also bears a higher cost. Catering to the user 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.

J. Yang (✉) · C. Liu · Y. Shang · Z. Mao · B. Cheng · C. Liu · L. Niu · J. Chen
State Key Lab of Networking and Switching Technology, Beijing University of Posts & Telecommunications, Beijing, 100876, China
e-mail: yangjingqi88122@gmail.com

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 (Dutta et al. 2012), and this one is the most popular method in the virtual resource scaling. Though this method provides a larger scale resource, but it incurs a considerable waste of computing 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 (Wang et al. 2012). Modern hypervisors support online 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. The vertical scaling can be further divided into the self-healing scaling and the resource-level scaling in some researches (Han et al. 2012). The horizontal scaling and the vertical scaling are suitable in different situations, and an intelligent combination of these two scaling methods may help us find an 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 a novel service cloud architecture which offers different services and an auto-scaling approach in service clouds 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. Finally, we conduct experiments to show that our auto-scaling approach can scale service clouds with less costs and better performance.

The remainder of this paper is organized as follows: Section 2 shows challenges of the service cloud scaling and provides solutions accordingly; Section 3 proposes our cloud auto-scaling approach; Section 4 introduces the cloud scaling architecture to support our approach; Section 5 evaluates our approach; Section 6 presents an overview of related work on the workload prediction and scaling techniques used for clouds, and it provides a comparison between related work and our approach; and finally Section 7 concludes this paper and describes future work.

2 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. We present it using a service cloud as an example.

Figure 1 shows the logical architecture of the server cluster in our service cloud. There are several different applications in the server cluster. Each application is composed of one or more services which achieve the function of an application together. 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.

Figure 2 shows the physical architecture of the server cluster, and it abstracts how VMs of each service are allocated in cluster nodes. A cluster node is a physical machine. VM workers of a service may run on the same cluster node, and they also can be deployed on different cluster nodes. VM workers have a variety of processing capabilities, e.g. small, medium and big in Amazon EC2 (http://aws.amazon.com/ec2/). 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. But we will not discuss that scenario in this paper.

2.1 Predicting workloads

In order to achieve scalability the server cluster should add or release virtual resources dynamically according to

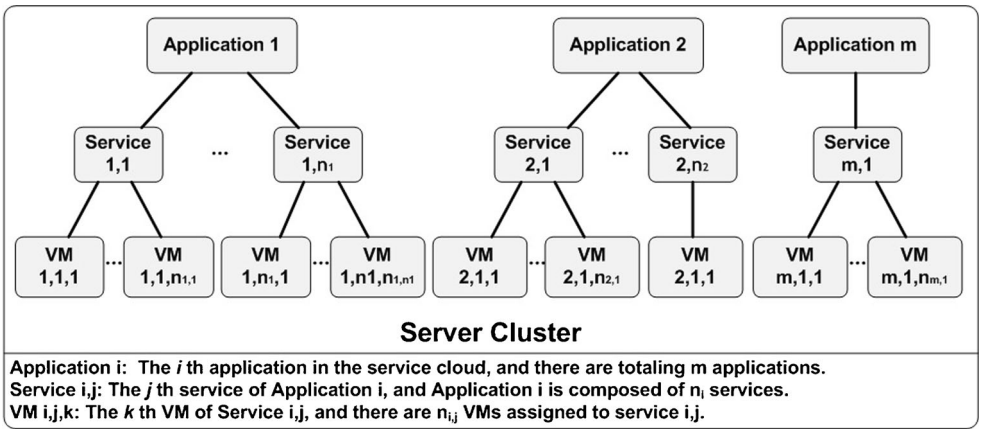

Fig. 1 Logical architecture of server cluster

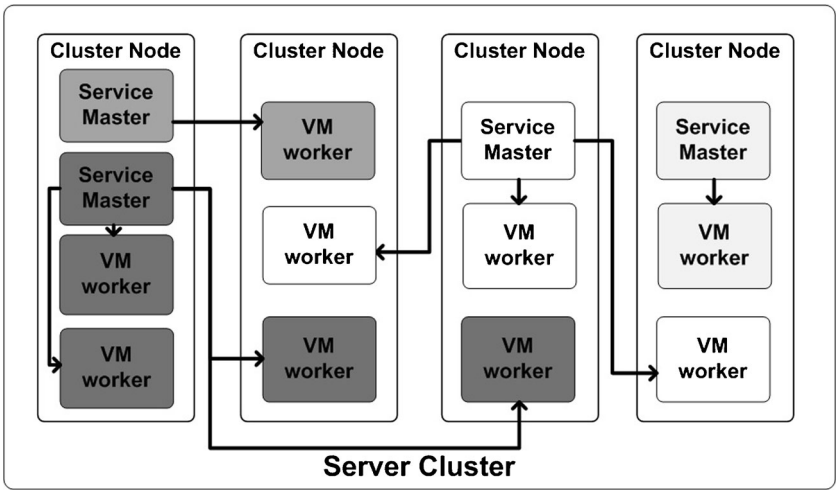

Fig. 2 Physical architecture of server cluster

platform running states. The general approach of scaling up platform is to add virtual resources when the system detects a higher system utilization exceeding the upper threshold. We can execute the horizontal scaling or the vertical scaling. The horizontal scaling is not subject to the resource constraints, but there is a problem that it will take a while to complete the horizontal scaling. This may lead more user SLA violations. The scaling time taken by the vertical scaling can be ignored, but 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. According to the architecture in Fig. 1, we can see that there will be different types of requests which are produced by different services, and each type of requests consumes a different amount of resources. Thus workloads of different services are predicted independently in service clouds. Section 3.1 describes our workload prediction model.

2.2 Dealing with the predicted deviation

Because of the abnormal and burst of workloads in service clouds, there may be a predicted deviation. The prediction workload may be lower than the actual workload, and then resources scaled at the last interval may be not enough to handle requests. 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.

2.3 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 as far as possible. Given a resource request, how to choose the most suitable virtual resource to add becomes an issue. 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 waste and reducing computational overheads, we use a greedy approach to achieve the optimization goal. Section 3 shows how we solve this complex problem with low complexity.

3 Cloud auto-scaling approach

In this section, we explain our cost-aware auto-scaling approach in more detail. 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.

In our approach 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. 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. Because the self-healing scaling is free of the scaling cost (*SC*), it is usually executed first, and then the resource-level scaling is conducted. All these methods have their advantages and disadvantages, and they could be applied to different occasions.

3.1 Workload prediction model

In our strategy, the workload prediction is used to predict the request number of services at the next time interval. There are a number of techniques existing in literature that can be applied for forecasting the workload. 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 shown in Fig. 3. As a result of these, in this paper, we use a linear regression model (LRM) to solve this problem.

The linear regression model is the principal style of econometric models (Baltagi 1998). It is used to study the relationship between a dependent variable and an independent variable. The generic form of the linear regression model is

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

Where Y is the dependent or explained variable, X is the independent or explanatory variable, and i indexes the sample observation. In our scenario, Y is the workload, and X is the time. 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.

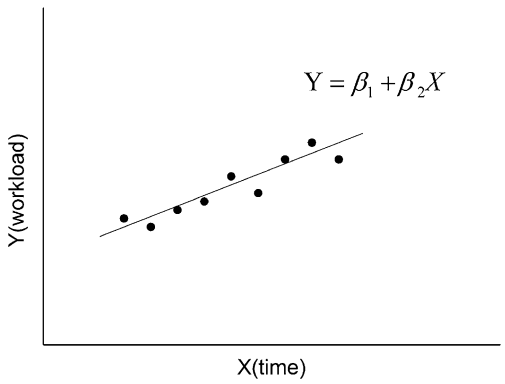

Fig. 3 Linear regression model

We use the Ordinary Least Squares to solve this equation, then we can get the results Eq. 2 and 3.

$$\sum Y_i = n\beta_1 + \beta_2 \sum X_i \tag{2}$$

$$\sum X_i Y_i = \beta_1 \sum X_i + \beta_2 \sum X_i^2 \tag{3}$$

According to the Cramer's Rule, we can obtain the solution of linear simultaneous equations of β_1, β_2, as shown in Eq. 4 and 5.

$$\beta_1 = \frac{\sum X_i^2 \sum Y_i - \sum X_i \sum X_i Y_i}{n \sum X_i^2 - (\sum X_i)^2} \tag{4}$$

$$\beta_2 = \frac{n \sum X_i Y_i - \sum X_i \sum Y_i}{n \sum X_i^2 - (\sum X_i)^2} \tag{5}$$

β_1, β_2 can be calculated by substituting known previous workloads into Eq. 4 and 5, and then we can calculate the workload at the next interval using Eq. 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. Three comparison method are listed, as follows.

A second order autoregressive moving average method filter (ARMA) (Roy et al. 2011). The equation for the filter used is given by

$$Y_{t+1} = \beta * Y_t + \gamma * Y_{t-1} + (1 - (\beta + \gamma)) * Y_{t-2} \tag{6}$$

The value for the variables β and γ are given by the values 0.8 and 0.15, respectively.

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.

LRM can obtain different results depending on the length of the sliding window, and the sliding window is set to four intervals for better performance based on experiment results in this scenario.

We collect workloads of a video service for 6 hours every time interval in a service cloud, and the workloads accord with regular rules in Dilley (1996). 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 Fig. 4 and Table 1. We can see that in most situations our linear regression model outperforms other methods. Our LRM has a lowest predicted deviation among these four methods. 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

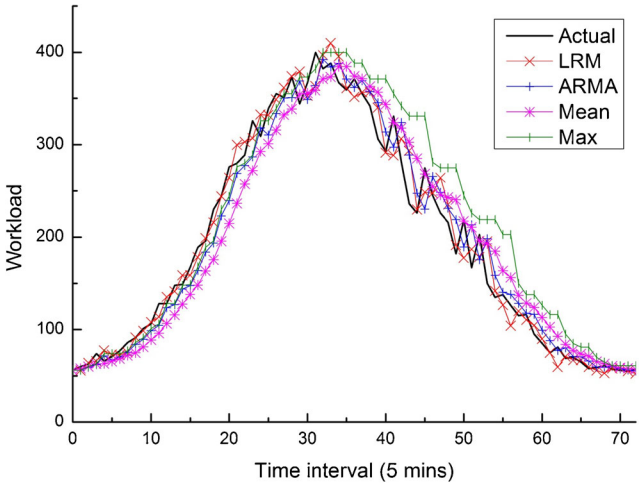

Fig. 4 Predicted and actual workloads

is less sensitive of workload variations, and Max prediction is higher than the actual workload most of the time.

3.2 Auto-scaling algorithm

In order to help understand the algorithm, Table 2 lists the main parameters used throughout this approach.

When the service cloud platform needs to be scaled, the ideal strategy is releasing the higher cost virtual resource, or adding the lower cost virtual resource. A cost model is proposed to measure the scaling cost of a operation. A *SC* is composed of a virtual resource cost (*VRC*) and a license cost (*LC*), as follows.

$$ScalingCost(SC) = VirtualResourceCost(VRC) + LicenseCost(LC) \quad (7)$$

VRC is the cost of CPU, memory and other resources used by services. So as shown in Eq. 8, *VRC* is the sum of resource costs of scaled virtual resources. 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. Thus we will take no account of *VRC* if the platform only performs the resource-level scaling. There may be different types of VMs which have different capacities, and these VMs' costs are dependent on their sizes. Hence we take *VRC* into account in the VM-level scaling.

Another consideration needs to be mentioned is that we must pay for license fees if there are business softwares running on VMs. Thus *LC* is added to *SC*. In most cases one software installed in a VM requires only one license no matter the VM's capacity, 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 the VM-level scaling, and it is shown in Eq. 9.

$$VRC = \sum_{k=1}^{n_{i,j}} cost(r_{i,j,k}) \quad (8)$$

$$LC = n_{i,j} * cost(license_{i,j}) \quad (9)$$

Pseudo code for the auto-scaling approach is given below. The algorithm 1 is started after the service cloud platform is deployed, 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). This is a system initialization stage. Services of our service cloud are scaled individually, and this can simplify the scaling problem (line 4). At every interval 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 based on this information (line 6). Finally it pre-manages virtual resources for the next interval (line 7)

The real-time virtual resource management manages resources at this interval, and it 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 to reduce the cost of the platform. This algorithm is shown in algorithm 2. In real-time environment scaling methods should be quick

Table 1 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

Table 2 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 $s_{i,j}$

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
```

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 Han et al. (2012) for detailed pseudo code of the self-healing scaling and the resource-level scaling) which scale resources within a short time. 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). 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.

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
```

The algorithm 3 performs the resource scaling down until releasing resources will cause the utilization higher than the upper threshold. It first executes VM-level scaling down (line 3). Because the resource-level scaling is limited with VM resources, it usually cannot meet the demand of released resources. If the resource-level scaling is conducted first, releasing a VM after that is more likely to lead to the utilization exceeding the upper threshold. Then the VM-level scaling down will not be executed, and it may cause a low utilization of VMs. Thus when the platform needs to release resources, first the algorithm releases a VM with the largest *SC* every time in the VM-level scaling, and then it releases as many resource-level resources as possible under the upper threshold (line 4 and line 5).

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
```

The approach completes the real-time scaling so far, then it pre-manages virtual resources for the next interval based on the workload prediction in algorithm 4. Algorithm 4 does not execute the virtual resource scaling down, because there may be prediction deviations. The pre-scaling down may cause a high utilization which is higher than the upper threshold, so 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 linear regression workload prediction model introduced in Section 3.1 (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 the smallest *SC*. How to find such a strategy will be expounded in detail in Section 3.3.

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
```

3.3 Cost-aware pre-scaling

The objective of pre-scaling up is to find a combination of virtual resources at different resource levels, which have the smallest *SC* and enough resources to handle increased requests of a service at the next interval, and then scale up virtual resources. The self-healing scaling correlates with the real-time state of VMs closely, so it is not be used in the pre-scaling step which manages virtual resources at the next interval. The VM-level scaling and the resource-level scaling are used in the pre-scaling.

This optimization problem can be transformed into an integer programming problem based on the analysis of *SC* components and constraints. It is formally defined as: minimize

$$SC = VRC_r + VRC_v + LC_v, \tag{10}$$

where

$$VRC_r = n_r * cost(resource) \tag{11}$$

$$VRC_v = \sum_{i=1}^{n_v} nVM_i * cost(VM_i) \tag{12}$$

$$LC_v = \sum_{i=1}^{n_v} nVM_i * cost(license), \tag{13}$$

subject to

$$n_r + \sum_{i=1}^{n_v} nVM_i * wVM_i > n \tag{14}$$

$$n_r < a_r \tag{15}$$

Table 3 describes the components of this problem. In the pre-scaling step, a *SC* is the sum of three parts (10). The first part VRC_r is *VRC* in the resource-level scaling, and it is the product of the number of resource units and the cost of a unit resource (11). A unit resource is the resource consumed by each request of this service. There is no *LC* in the resource-level scaling as described in Section 3.2. The second part VRC_v is *VRC* in the VM-level scaling, and it is the sum of costs of every type of VMs (12). Costs of different types of VMs vary with their capacities, and usually costs increase in proportion to capacities. The third part LC_v is *LC* in the VM-level scaling, and it is the sum of *LC* of every VM (13). Constraint (14) guarantees that resources to be increased can satisfy user requests. Constraint (15) indicates resources to be increased must be available in this cluster node.

Table 3 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
n_v	Number of VMs' types
VM_i	Type i VM
nVM_i	Number of increased VM_is
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

This problem is an NP-hard problem, and it cannot get the optimal solution in polynomial time. In order to reduce the computational overhead, we can calculate the near-optimum solution with a heuristic algorithm. Then a greedy approach is used to achieve the optimization goal, as shown in the algorithm 5.

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
```

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 of a VM is larger. So first the algorithm finds an 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(line 4), the algorithm boots a VM with the smallest capacity (line 5). Conversely, if available resources are sufficient (line 6), the cost of the VM-level scaling may be higher or lower than the resource-level scaling. The algorithm calculates costs of the resource-level scaling which scales resources to handle remaining requests and the VM-level scaling which scales the smallest VM, and then it makes a comparison of these two costs. The algorithm chooses a cheaper strategy to execute (line 7).

4 Scaling framework

To implement the cost-aware auto-scaling approach based on the workload prediction in our service cloud, we design and develop a cloud scaling architecture, as shown in Fig. 5. Here we focus on the management of user requests and VMs, so we give no more explanations on the management of network and so on. Fig. 5 shows all main components which achieve the scalability of our service cloud.

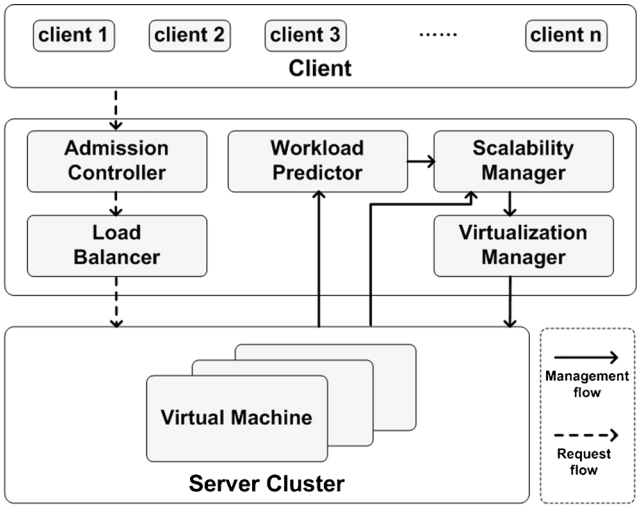

Fig. 5 Cloud scaling architecture

Admission Controller and Load Balancer. User requests generated by clients are sent to services through the admission controller. First the admission controller filtrates out invalid login requests, then requests come into the load balancer. As more than one VM exists in the server cluster, there are needs in how to dispatch requests to different VMs dynamically. Load balancer is the enabling force for this work. The load balancer collects server statuses from the server cluster, and then it forwards requests to a suitable VM according to the load-balancing strategies.

Workload Predictor. Based on the information collected from the server cluster's history log, the workload predictor predicts the workload of a service at the next interval. This component uses the linear regression model which is shown in Section 3.1 to predict workloads. The system decides whether to scale up resources according to prediction results.

Scalability Manager. The scalability manager is the core component of this scaling architecture, and the auto-scaling approach is implemented here. It decides when and how to scale the service cloud. In the real-time scaling or pre-scaling, the scalability manager proposes a scaling strategy according to platform running states or prediction results.

Virtualization Manager. The virtualization manager deals with virtual resources in the server cluster directly. It executes strategies proposed by the scalability manager, and then it adds or releases virtual resources respectively.

There are three flows in the cloud scaling architecture. The first flow is the distribution of requests of different services through the admission controller and the load balancer. The other two flows are management flows about scaling: the pre-management flow and the real-time management flow. The pre-management flow starts with collecting information from the server cluster in the workload predictor, and then the scalability manager proposes a pre-scaling strategy. Which is slightly different to the pre-scaling, the real-time management flow collects information in the scalability manager directly, and then it proposes a strategy. Finally, strategies are executed by the virtualization manager in these two flows.

5 Evaluation

5.1 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 (Calheiros et al. 2011) which is a framework for modeling and simulating the cloud computing infrastructures and services.

Using Amazon EC2 (http://aws.amazon.com/ec2/) as a reference, three types of VMs are provided in our experiment: small, medium and big, and they have different capacities and different virtual resource costs, as shown in Table 4.

There are two applications in the experimental service cloud, and each of them is composed of more than one services. These two applications are developed in our laboratory. Table 5 lists all services of these two applications. The first application is a Social Network Sites (SNS), and it is used to share pictures and blogs with other people, post comments to some news, send E-mail and so on. The other one is a video site which can be used to upload and download videos.

In the experiment, we compare our approach with two baseline methods. The first strategy is labeled "Horizontal scaling", and it adds or releases fixed size VMs when the utilization is beyond or below thresholds. The second strategy uses the method in Han et al. (2012) which is labeled "Lightweight Scaling", and it also scales virtual resources at three different levels. Our approach is labeled "Auto-scaling". Finally, the effectiveness of our approach is validated by experiment results from different aspects.

5.2 Evaluation results

Figures 6 and 7 show resource usages and cumulative resource usages of three approaches at each interval respectively.

Table 4 Different settings of VMs

	Memory	CPUs	Disk	Price-per hour
small	1.7Gb	1	160Gb	$0.065
medium	3.75Gb	1	410Gb	$0.130
big	7.5Gb	2	850Gb	$0.260

Table 5 Services of applications

Application	Service
SNS	Picture service
	Blog service
	Comment service
	E-mail service
Video site	Video service
	Cache service

From the results we can see that the horizontal scaling consumes the most resource for most intervals. Because it scales virtual resources in VM unit, it cannot scale resources in a fine-grained unit. So there are more idle resources than the other approaches. Figure 8 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 and costs. Even so, our approach costs 3 % less than the horizontal scaling. Figure 9 indicates utilizations of three approaches at each interval. Utilizations of the auto-scaling, the lightweight scaling and the horizontal scaling are 82 %, 84 % and 74 % respectively. The first two approaches are able to utilize resources more fully, and the last one wastes more resources.

Based on the above results, the lightweight scaling and the auto-scaling are similar in resource using and utilizations, 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

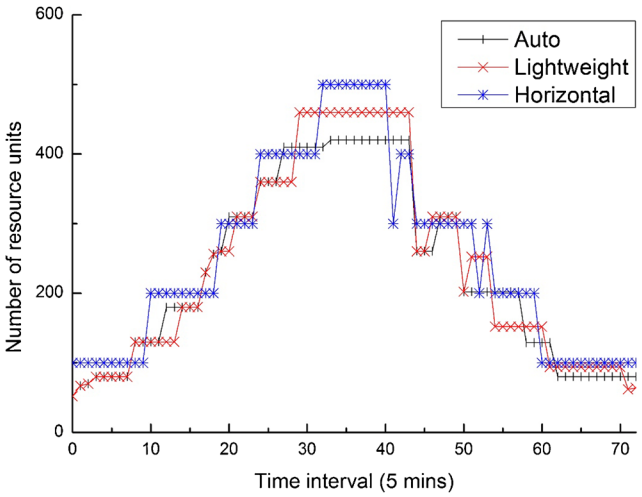

Fig. 6 Resource usages at different intervals

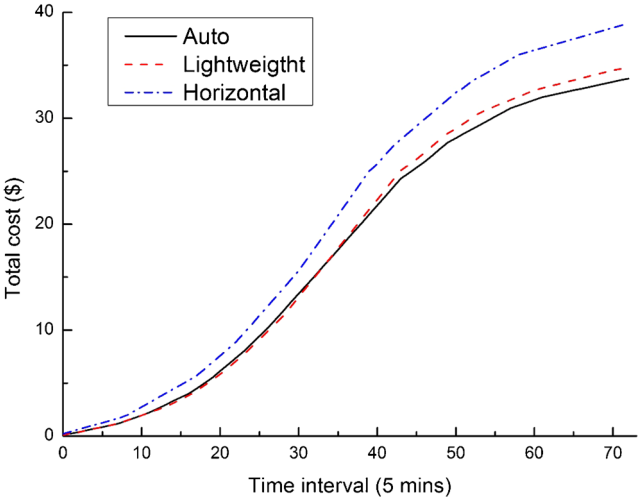

Fig. 7 Cumulative resource usages at different intervals

cause SLA violations. Comparing with the auto-scaling, the lightweight scaling incurs a higher SLA violation known from Fig. 10. Three CDFs of utilizations are shown in Fig. 10. 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 resource utilizations of the auto-scaling and two

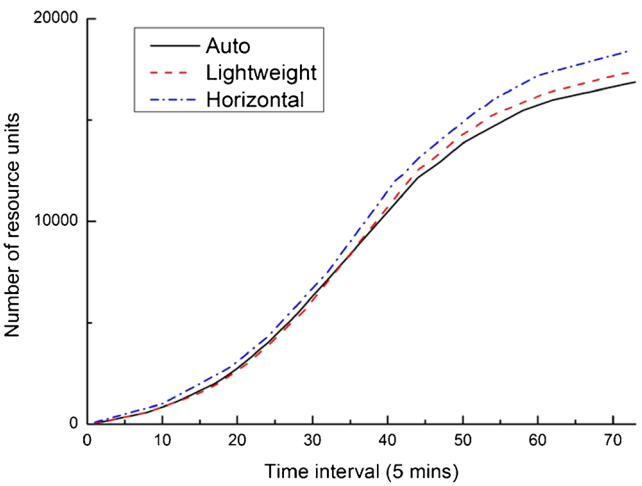

Fig. 8 Cumulative scaling costs at different intervals

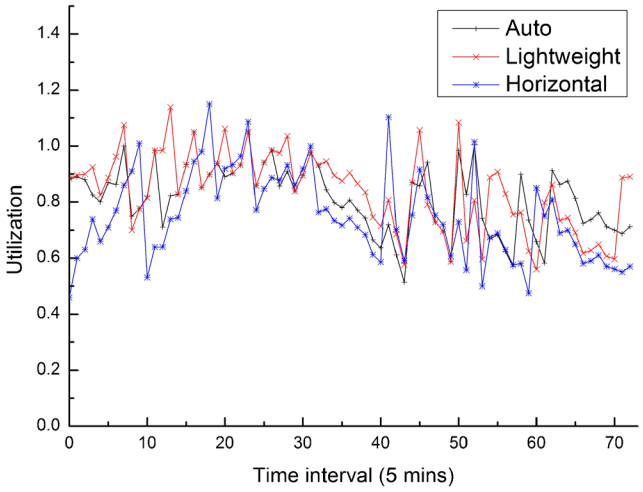

Fig. 9 Utilizations at different intervals

comparison approaches in a service cloud. We observed that the auto-scaling outperforms the others in the experiments. Based on the workload prediction, the auto-scaling approach can scale virtual resource dynamically at a lower cost and satisfy more user requests.

6 Related work

We have surveyed related work that investigates service clouds, scaling techniques and prediction methods in clouds.

Some researches are carried out about service clouds. REMICS (http://www.remics.eu/), a FP7 (http://ec.europa.eu/research/fp7/index_en.cfm) project, defines the methodology and tools to deploy services in clouds. It shows

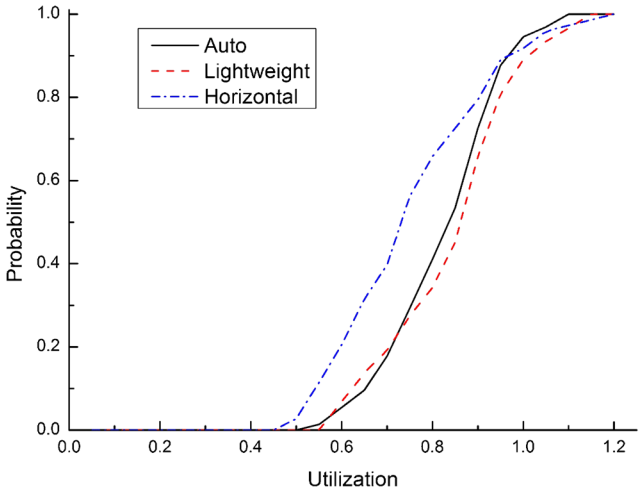

Fig. 10 CDF of utilizations

advantages of service clouds, and points out that for taking advantages of service cloud computing technologies the quality requirements such as the scalability become important (Mohagheghi and Sther 2011). But there are not so many researchers working on how to scale the virtual resources in service clouds.

In order to scale the platform properly, it is necessary to predict workloads of service clouds for better performance. A number of different methods have been used for predicting workloads. In Roy et al. (2011), it develops a model-predictive algorithm for the workload prediction in which a second order autoregressive moving average method filter is used. In Saripalli et al. (2011), a two-step prediction approach and a hot spot detection model are proposed. This prediction method aims to get a representative view of the workload trend from measured raw data, and then applies a workload prediction algorithm to workload trends. But these kinds of previous methods are static, and it cannot adjust their models with workload trends which may lead to an inaccurate prediction. PRESS Gong et al. (2010) 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. Analogously, methods which use a pattern matching or a string matching (Caron et al. 2010) are overly dependent on history workloads, and they also involve a large number of calculations in most cases. So we propose a linear regression model which can adjust its model dynamically with fewer computations.

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 produced based on the horizontal scaling. In Roy et al. (2011), it proposes an efficient auto-scaling method based on the horizontal scaling to help cloud service providers in operating modern day data centers to support the most customers. Empirical results are provided to demonstrate that resources can be allocated and deallocated by this algorithm in a way that satisfies both the application QoS (Quality of Service) while keeping operational costs lows. WebScale (Lin et al. 2012) which is developed for managing resources for Web applications provides on-demand resources in the form of virtual machines. In Lin et al. (2011), a new algorithms, Dynamic Round-Robin, is proposed for the energy-aware virtual machine scheduling and consolidation. It reduces a significant amount of power consumption. The horizontal scaling is also supported in commercial clouds (e.g., Amazon EC2 Autoscale (http://aws.amazon.com/autoscaling/)) which allow an automated way to increase the number of VM instances provisioned for an

enterprise. Along with the development of the virtualization technology, more and more researchers are aware of advantages of the vertical scaling. SmartScale (Dutta et al. 2012) uses a combination of the vertical scaling and the horizontal scaling to ensure that the application is scaled in a manner that optimizes both the resource usage and the reconfiguration cost incurred due to the scaling. In Han et al. (2012), it proposes a lightweight approach to enable the elasticity for cloud applications. This work also uses self-healing scaling, resource-level scaling and VM-level scaling, but it does not take predicted deviations and the horizontal scaling time into consideration, and we use these scaling technologies in a more suitable way.

7 Conclusion and future work

We investigate the problem of auto-scaling based on predicted workloads in service clouds in this paper. We use a linear regression model to predict the workload at the next interval based on previous workloads, and we also propose an approach to scale the service cloud 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 to reduce costs. In order to implement these mechanisms, we propose a cloud scaling architecture to support this approach, and it implements both the workload prediction and the service cloud scalability.

Our approach is compared with other methods. 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. It also can be expanded to construct a more complex environment easily.

We are currently conducting more simulations to further evaluations and improving the auto-scaling approach with data sets from a SNS system and a multimedia conference system deployed in our service cloud. In addition, we are also developing a service cloud which integrates the existing service platforms of our laboratory based on our cost-aware auto-scaling approach.

Acknowledgments The 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

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. (2009). *Above the clouds: a berkeley view of cloud computing*. EECS Department, University of Califonia, Berkeley, Technical report, UCB/EECS-2009-28.

Baltagi, B.H. (1998). Econometrics (pp. 41-69). Berlin: Springer.

Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C., Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisoning algorithms. *Software: Pratice and Experience*, *41*(1), 23–50.

Caron, E., Desprez, F., Muresan, A. (2010). Forecasting for grid and cloud computing on-demand resources based on pattern matching. In *Proceedings of 2nd IEEE international conference on cloud computing technology and science* (pp. 456–463).

Dilley, J.A. (1996). Web Server Workload Characterization, Hewlett-Packard Laboratories Report, HPL-96-160. http://www.hpl.hp.com/techreports/96/HPL-96-160.html.

Dutta, S., Gera, S., Verma, A., Viswanathan, B. (2012). SmartScale: automatic application scaling in enterprise clouds. In *Proceedings of the 2012 IEEE 5th international conference on cloud computing* (pp. 221–228).

Gong, Z., Gu, X., Wilkes, J. (2010). PRESS: PRedictive elastic ReSource scaling for cloud systems. In *Proceedings of the 2010 international conference on network and service management* (pp. 9–16).

Han, R., Guo, L., Ghanem, M.M., Guo, Y. (2012). Lightweight resource scaling for cloud applications. In *Proceedings of the 12th IEEE/ACM international symposium on cluster, cloud and grid computing* (pp. 644–651).

Lin, C.-C., Wu, J.-J., Lin, J.-A., Song, L.-C., Liu, P. (2012). Automatic resource scaling based on application service requirements. In *Proceedings of the 2012 IEEE 5th international conference on cloud computing* (pp. 941–942).

Lin, C.-C., Liu, P., Wu, J.-J. (2011). Energy-aware virtual machine dynamic provision and scheduling for cloud computing. In *Proceedings of the 2011 IEEE 4th international conference on cloud computing* (pp. 736–737).

Mohagheghi, P., & Sther, T. (2011). Software engineering challenges for migration to the service cloud paradigm: ongoing work in the REMICS project. In *Proceedings of 2011 IEEE world congress on services* (pp. 507–514).

Roy, N., Dubey, A., Gokhale, A. (2011). Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proceedings of the 2011 IEEE 4th international conference on cloud computing* (pp. 500–507).

Samimi, F.A., McKinley, P.K., Sadjadi, S.M., Tang, C., Shapiro, J.K., Zhou, Z. (2007). Service clouds: distributed infrastructure for adaptive communication services. *IEEE Transactions on Network and Service Management*, *4*(2), 84–95.

Saripalli, P., Kiran, G., Shankar, R., Narware, H., Bindal, N. (2011). Load prediction and hot spot detection models for autonomic cloud computing. In *Proceedings of the 2011 4th IEEE international conference on utility and cloud computing* (pp. 397–402).

Wang, W., Chen, H., Chen, X. (2012). An availability-aware approach to resource placement of dynamic scaling in clouds. In *Proceedings of the 2012 IEEE 5th international conference on cloud computing* (pp. 930–931).

Jingqi Yang graduated from the school of software engineering and received her B.Eng. degree from Beijing University of Posts and Telecommunications (BUPT), China, in 2009. She is currently a Ph.D. candidate in computer science and technology at the State Key Laboratory of Networking and Switching Technology, BUPT. Her research interests are service computing and cloud computing.

Chuanchang Liu received his B.S. degree in computer science from Harbin Institute of Technology, China, and he received his M.S. and Ph.D. degrees in computer science from BUPT, China. He is currently an assistant professor in the State Key Laboratory of Networking and Switching Technology at BUPT. His research interests include network service and intelligence, service computing, and semantic web.

Yanlei Shang received his Ph.D. degree in computer science and technology from BUPT in 2006. Then he worked in the Nokia Research Center as a postdoctoral research fellow. He is currently an associate professor in the State Key Laboratory of Networking and Switching Technology, BUPT. His research interests include cloud computing, service computing, distributed system and virtualization technology.

Bo Cheng received his Ph.D. degree in computer application technology from University of Electronic Science and Technology of China in 2006. Then he worked as a postdoctoral research fellow in BUPT. Currently, he is a professor of the Research Institute of Networking Technologies of BUPT. His research interests are web services, mobile internet, Internet of Things and cloud computing.

Zexiang Mao graduated from the school of Mathematics and received his B.Sc. degree from Shandong University, China, in 2008. He then graduated from the school of Information and Communication Engineering of BUPT with a Masters degree, China, in 2010. He is currently a Ph.D. candidate in computer science and technology at the State Key Laboratory of Networking and Switching Technology, BUPT. His research interests are service computing and cloud computing.

Chunhong Liu received her B.S. degree in computer science from Henan Normal University, China, and she received her M.S. degree from XiDian University, China. She is currently a Ph.D. candidate in computer science and technology at the State Key Laboratory of Networking and Switching Technology, BUPT. Her research interests are service computing and cloud computing.

Lisha Niu received her B.S. Degree and M.S. degree in computer science from Hebei University of Technology, China. She is currently a Ph.D. candidate in computer science and technology at the State Key Laboratory of Networking and Switching Technology, BUPT. Her research interests are service computing and cloud computing.

Junliang Chen graduated from the Department of Telecommunications, Shanghai Jiaotong University in 1955 and received the Doctor of Engineering degree from Moscow Institute of Electrical Telecommunications, former Soviet Union, in 1961. He has been working in BUPT since 1955. Currentlyhe is a professor and director of the Research Institute of Networking Technologies of BUPT. He is both a member of Chinese Academy of Sciences and Chinese Academy of Engineering. His current research interests include network intelligence and services, switching technology, communications software and fault-tolerant computing.