Profit-Driven Resource Scheduling for Virtualized Cloud Systems

Shiyang Ye, Tao Wang, Wenbo Zhang, Hua Zhong
Institute of Software, Chinese Academy of Sciences
Beijing, China, 100190
ye_shi_yang@163.com, {zhangwenbo, zhongh}@otcaix.iscas.ac.cn

Abstract—**Virtualized resource renting is a key issue in IaaS (Infrastructure-as-a-Service) cloud systems. Suitable resource allocation improves resource utilization and increases profit for application providers. The application provider will obtain better revenues according to the Service Level Agreement (SLA), if they rent more virtual resources. However, they will invest much more capital for renting these virtual resources. How many resources a provider rents has become a key thing for cloud applications. This paper addresses the reconciliation objectives by proposing a profit-driven resource scheduling method for virtualized cloud systems. Compared with traditional methods, our method aims at maximizing the revenues by introducing SLA and the cost of renting cloud resource, instead of increasing resource utilization or decreasing early finishing time. We model the performance of applications with queueing theory; calculate the revenues according to the SLA and renting cost; adjust the amount of virtual resources to adapt to dynamic workloads in period. We have implemented a framework for scheduling virtual resources, and applied it in our IaaS cloud platform OnceCloud. The experimental results demonstrate that our method has advantages over existing ones in revenues.**

Keywords-cloud computing; virtual machine; resource scheduling; service level agreement

I. INTRODUCTION

Cloud computing has five main principles including pooled computing resources available to any subscribing users, virtualized computing resources to maximize hardware utilization, elastic scaling up or down according to need, automated creation of new virtual machines or deletion of existing ones, resource usage billed only as used. These principles, which provide convenient system management and lower maintenance cost, bring significant business profit. According to the types of provided cloud services, cloud computing is classified at three levels including IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service). Service providers of IaaS, which is the lowest level, provide imagines for different operating systems. These imagines can be customized by customers on demand. Customers obtain and use the instances of these imagines online, and pay for the usage of computing, storage and network resources. The typical example of IaaS is Amazon Elastic Compute Cloud (EC2).

The cloud computing platform ought to meet the customers' requirements for QoS (Quality of Service). Meanwhile, as cloud computing introduces a new business model, investment has become a focus. Thus, cloud computing ought to consider both requirements in QoS and investment in physical resources. A challenge in cloud computing is to allocate reasonable physical resources to maximize profit. IaaS makes use of virtual machine technology to allocate and relieve physical resources (e.g., CPU, memory, disk, network) on demand. Virtual machine management has become the critical method for scheduling physical resources. Although the IaaS model of renting virtual resources is much more flexible than the traditional model of renting physical resources, the cloud computing platform still needs to separate physical resources into BTU (Billing Time Unit) [1]. Figure 1 shows that cloud application providers utilize the management console to deploy cloud applications in virtual machines. The resource scheduler allocates virtual resources in BTUs for customers, according to SLA profit functions, resource renting cost and dynamic workloads.

SLA (Service Agreement Level) has become the major attribute to evaluate QoS. SLAs, which are the agreements between customers and service providers, are widely used to guarantee that the QoS meet the customers' requirements. The violation of SLAs will bring loss in the economy, so cloud computing platforms ought to take into account the trade off between SLA guarenteements and physical resource cost. How to calculate the amount of rented physical resources to maximize application providers' profit is a critical problem.

This paper proposes a profit-driven resource scheduling method for virtualized cloud systems which takes into account of both the loss of SLA violation and the cost of renting virtual resources. The aim of our method is to

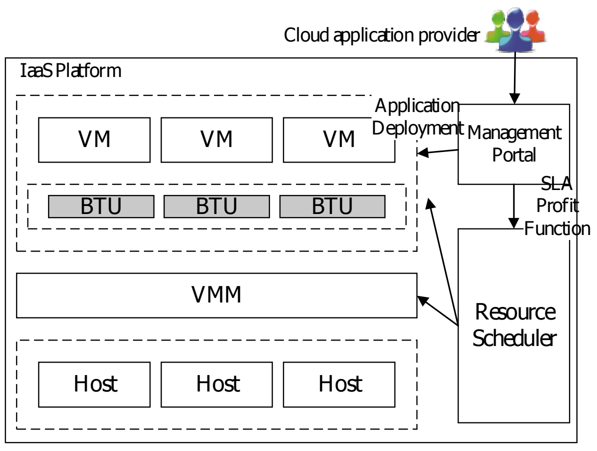


Figure 1 IaaS Resource Scheduling Framework

maximize the customers' profit. We use the queueing theory to model the procedure of processing requests, decide SLA violation, and calculate economic loss. Then, we use a hill climbing algorithm to calculate the influence of the resource amount on overall profit. We present an algorithm to maximize the customers' profit by adjusting the amount of allocated resources dynamically. Finally, we have implemented a framework for scheduling virtual resources, and applied it in our IaaS cloud platform OnceCloud. The experimental results demonstrate that our method has advantages over existing ones in revenues.

II. MODELING APPLICATION PERFORMANCE BY QUEUEING THEORY

To calculate the economic loss brought for SLA violation, we first estimate the time of processing customers' requests. Queueing theory is a random service system theory. A queue is composed of customers and service stations. Customers are the object of service stations, and service stations provide service for customers. If we cannot provide sufficient service stations for customers, the requirements of customers for guaranteeing the quality of service cannot be met. On the other hand, if there are redundant service stations, the cost of renting virtual resources increases. Therefore, in essence, the queueing theory is to calculate the tradeoff between the requirements of QoS and the cost of renting virtual resources.

Figure 2 describes a queue composed of a request input, queueing rules, and a service procedure. During the input period, different types of requests arrive at the service system with different arriving rates. A request waiting in queues leaves when a service station has processed the request. Many service stations in parallel provide services for the requests of different customers. In the cloud computing environment, application providers always rent many instances of virtual machines to deploy the same application for guarantee QoS. Thus, we use queueing theory model in M/M/S to model cloud computing systems. We assume that requests are independent and according to negative exponential function model. The service times for different requests are also independent.

Different requests arrive at systems at different rates at random. A request scheduler adds requests in different queues according to the request types and server resources to wait for scheduling. Every BTU is abstracted as a service station, and every server is abstracted as many service

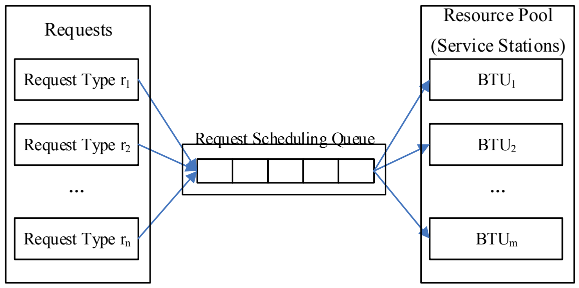


Figure 2 Queueing Model for Processing Requests

stations according to resource allocation.

It is assumed that the number of arriving requests with type i is n_i. The arrival probability of request i is :

$$p_i = \frac{n_i}{\sum_{i=1}^{m} n_i},$$

$$\sum_{i=1}^{m} p_i = 1.$$

The arrival rate of request i is:

$$\lambda_i = \frac{T}{n_i}.$$

The mathematical expectation of the arrival number of requests i during period T is:

$$\mathrm{E(n)} = \sum_{i=1}^{m} p_i n_i.$$

The arrival rate of all the requests is:

$$\lambda = \frac{1}{E(\mathrm{n})}.$$

It is assumed that the service time of request i is:

$$\mathrm{E(t)} = \sum_{i=1}^{m} p_i t_i.$$

The service rate is:

$$\mu = \frac{1}{E(t)}.$$

It is assumed that customers arrive at the system one by one, the time interval between the arrived requests and the service time of every service station is in accordance with negative exponential distribution with parameter λ. There are m types of requests and s service stations in the system. When a customer arrives, the service station accepts the request if the station is idle, or the request waits in a queue waiting for processing.

When the system is in a stable status, the probabily that the length of the queue is N is:

$$p_n = P\{N=n\} (n=0,1,2,\dots),$$

$$\lambda_n = \lambda, n = 0,1,2\dots,$$

$$\mu_n = \begin{cases} n\mu, n = 0,1,2,\dots,s \\ s\mu, n = s, \mathrm{s}+1,\dots \end{cases},$$

where s is the number of service stations.

The service strength is:

$$\rho_{\mathrm{s}} = \frac{\rho}{s} = \frac{\lambda}{s\mu},$$

when $\lambda < 1$.

The average length of the queue for multiple waiting service stations L_q is:

$$L_q = \sum_{n=s+1}^{\infty} (\mathrm{n-s})\,\mathrm{p}_n = \frac{p_0\rho^s}{s!}\sum_{n=s}^{\infty}(\mathrm{n-s})\rho_s^{n-s} = \frac{c(\mathrm{s},\rho)\rho_s}{1-\rho_s}.$$

The average number of customers waiting for services $\bar{s}$ is:

$$\bar{\mathrm{s}}=\sum_{n=0}^{s-1} np_n + s\sum_{n=s}^{\infty} p_n = \sum_{n=0}^{s-1}\frac{n\rho^n}{n!}p_0 + s\frac{\rho^s}{s!(1-\rho_s)}p_0 = \rho.$$

The average length of the queue L_s is:

L_s =the average length of queues +the average number of customers waiting for services =L_q+ ρ .

In a queue system with multiple service stations, the average waiting time is W_s:

$$W_s = \frac{L_s}{\lambda}.$$

The response time of request *i* is:

$$rt_i = W_s + \text{service time } t_i.$$

III. SLA-AWARE PROFIT CALCULATION

In this section, we calculate the profit of applications on the basis of the response time estimation in the above section. Figure 3 describes the QoS demand of applications as follows.

1) *$rt > T_{defined}$*: SLA defines the expected response time *rt* of requests. When the response time is lower than $T_{defined}$, we think that users get the satisfied QoS, and there is no loss in SLA.

2) *$rt < T_{expired}$*: SLA defines the timed out of requests. When the response time is longer than $T_{expired}$, the server thinks that QoS cannot be satisfied, and the profit brought from processing the request is lost.

3) *$T_{defined} < rt < T_{expired}$*: When the response time is between $T_{defined}$ and $T_{expired}$, the quality of service decreases from MaxSLA to 0 as the response time increase.

$T_{defined}$ and $T_{expired}$ together describe the QoS requirements for applications. When a server completes a request for an application, the application gets a certain profit.

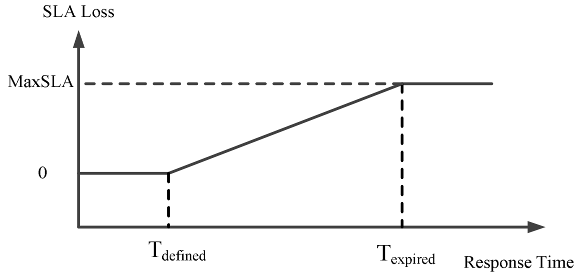


Figure 3 SLA Profit Loss Function

Let us give the SLA profit loss function as follows:

$$\mathrm{cost}\,SLA(rt) = \begin{cases} 0, 0 < rt < T_{defined} \\ \mathrm{g}(rt), T_{defined} \le rt \le T_{expired} \\ MaxSLA, rt > T_{expired} \end{cases},$$

$$g(rt) = a\times rt + b = \frac{(\mathrm{rt}-T_{defined})}{T_{expired}-T_{defined}} \times Max\mathrm{SLA}, T_{defined} \le rt \le T_{expired},$$

where *rt* is the response time.

Meanwhile, the SLAs for different requests in an application are different. For example, for an e-commerce website, the requests with purchasing is more prone to bring profit. Thus, the purchasing request has much stricter requirements than browsing request. Therefore, the profit functions for different requests are also different.

The cost of renting virtual resources in cloud platform is:

$$\cos\mathrm{tVM}(size) = size \times Cvm,$$

where *size* is the number of physical resource units (BTU), C_{vm} is the cost of the physical resource unit.

We can calculate the response time of request *i* according to the queueing theory.

$$\mathrm{rt}_i = \mathrm{rspTime}\ (\mathrm{t, v, s}),$$

where *t* is the request type, *v* is the number of requests, *s* is the number of physical resource units (BTU).

We focus on the sum of the cost of renting physical resources.

Thus, the problem we ought to solve is to adjusts to minimize *costTotal(s)*:

$$\cos tTotal(s) = \sum_{i=1}^{n} \cos tSLA(\mathrm{rt}_i) + \mathrm{costVM}(s),$$

where *s* is the number of BTUs.

Thus, we introduce that the issue to be addressed is how to select the number of BTUs to minimize the *costTotal(s)*.

IV. METHOD OF SCHEDULING VIRTUAL RESOURCES

Workloads in a cloud computing environment are in high complexity and dynamism. For example, the number of orders will increase significantly in some special promotion days, and the access pattern will change from browsing to ordering in a typical e-commerce website. Therefore, cloud computing platforms ought to have the capability of adjusting the amount of virtual resources dynamically in different kinds of workloads.

According to the above method, we propose a hill climbing algorithm to adjust resource allocation dynamically. The concrete steps are described as follows: We use tools to collect monitoring data, and use statistics to calculate the arrival rate of requests, service time and current number of service stations in period (1-2). We calculate the response time of requests according to the queueing theory model (16). We calculate the current cost according to the profit loss function and the cost of renting virtual resources (17). We use a hill climbing algorithm to estimate the minimum cost in neighbor values (5-11). We adjust the amount of virtual resources (BTU) by reallocating resources in VMs to minimize the cost.

Algorithm：SLA-ware virtual resource scheduling

Input： SLA profit estimation function of request i SLA(r_i)，cost of a BTU，Scheduling period T

Output：Allocate or release k BTUs

Description：

```
1.  Calculate $n_i$,$t_i$ in period $T$;
2.  Current=$s$; // the number of current BTUs
3.  While(1)
4.  {
5.  last=current;
6.  If(costTotal($s$)>costTotal($s$-1))
7.  $current$=$s$-1;
8.  If(costTotal($current$)>costTotal($s$+1))
9.  $current$=$s$+1;
10. if($last$=$current$)
11. break;
12. }
13. Adjust the number of $s$ as current.
14. costTotal($s$)
15. {
16. rti=queue($n_i$,$t_i$,$s$) with M/M/S model;
17. costR$_i$=costSLA$_i$($rt_i$);
18. return $\sum_{i=1}^{n} \cos tSLA(\mathrm{rt}_i) + \mathrm{costVM}(s)$;
19. }
```

V. A Framework for Scheduling Resource

According to our above proposed resource scheduling method, we have designed and implemented a framework for scheduling virtual resources in a cloud computing platform. The architecture is described in figure 4. It is composed of four components including monitoring proxies, a monitoring data process engine, a resource scheduler to schedule virtual resources, and a management console for system administrators.

A. Monitoring Proxy

Proxies collect monitoring data of different VMs including host metrics, VM metrics, and performance metrics and application workloads. These proxies are deployed on every Web application server and every host in the cloud. The proxies deployed on Web application servers collect performance metrics and workloads for every Web application, and the proxies deployed on hosts collect system level metrics of hosts and VMs. They report the collected monitoring data to the online process engine in period. These proxies require configurations (e.g., monitoring period, monitored metrics) from the management console.

The workload is composed of kinds of requests, and the system performance is influenced by workloads. Thus, we collect workloads and performance to analyze the system status for Web applications with queueing theory. We collect the workload and performance vectors with AspectJ through weaving in the session manager of Web application servers.

As for workloads, we consider both the volume and type of requests. An application server invokes web components to process requests, so the invocation sequence of web components reflects a workload. Thus, we introduce the workload vector, $wv= \{c_1,c_2,...,c_i,...,c_n\}$, to present the real workload in Web applications, where c_i is the invocation frequency of the i^{th} component, and n is the number of performance metrics.

For characterizing the performance, we pay attention to multiple performance metrics (e.g., response time and throughput). Thus, we introduce a performance vector, $pv= \{p_1, p_2,..., p_m\}$, to describe the performance, where p_i is the i^{th} performance metric, and m is the number of components.

Furthermore, we ought to get resource utilization of hosts and VMs. Virtualization platforms almost provide interfaces to allow the hypervisor to extract system level metrics of a VM. Thus, proxies deployed on hosts uses third-party library libxenstat to monitor the resource utilization of VMs from the domain 0.

B. Process engine

The process engine processes the monitoring data collected from proxies online. The engine filters the noisy data, and extracts required data from applications, platforms and physical resources to build the resource scheduling model. The engine includes two major components including cluster data collector and data processor.

The cluster data collector is responsible for collecting monitoring data from monitoring proxies deployed on different servers. The module listens connection requests from different proxies and receives monitoring data from them. The module maintains a peer list which records IP: Port of every peer in the cluster, and can add/delete peers in the list dynamically.

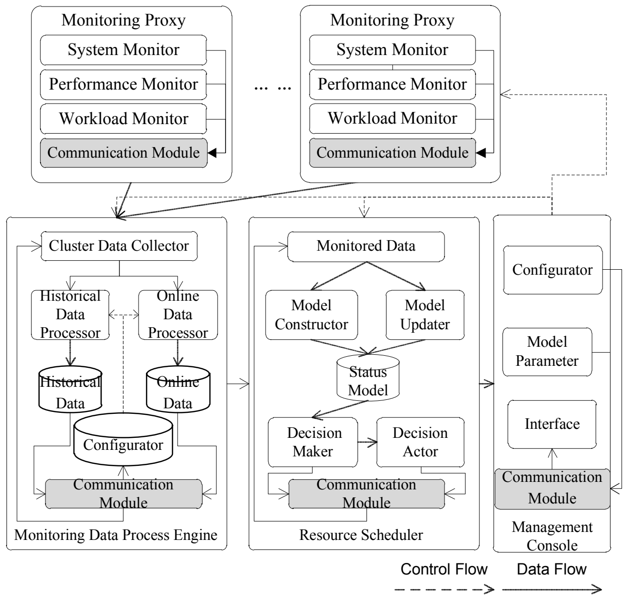


Figure 4 System Architecture

The historical/online data processor preprocesses the collected monitoring data. To standardize items, clear outliers, rectify faults and eradicate duplicated data, the processor adds the lesion data items, smooths data, identifies outliers and cleans the inconsistent data in the data cleaning period. The processor extracts useful data from the monitoring data from the application, platform and physical layers according to the selected resource scheduling model. Finally, the processor sends the processed necessary data to the scheduler for decision making.

C. *Resource Scheduler*

The resource scheduler schedules physical resources according to the monitoring data from the proceeding engine and scheduling models. The scheduler includes three modules that are model constructor and updater, decision maker and decision actor. The constructor and updater module constructs the system status model in the starting period, and updates the model in period. The decision maker module makes scheduling decision, and the decision actor module takes action to scheduling resources based on the updated model. Scheduler provides scheduling interfaces for adding and configuring models through management module for system administrators.

D. *Management Console:*

The management console provides machine human interaction interfaces. Users can use these interfaces to set the system configuration (e.g., monitored metrics, monitoring period), and the console shows the scheduling result. The management console is a B/S based Web application providing management console for system administrators. The console is composed of three modules including monitoring configurator, model parameter and data analyzer. The monitoring configurator connects and communicates monitoring proxies to trace the status of proxies. The model parameter sets configuration parameters including monitoring period, importing/exporting models, and integrating the third-party tools. The data analyzer analyzes monitoring data by extracting and sorting. System administrators use the graphical interfaces to interact with the system.

VI. EXPERIMENT

We have implemented the framework for scheduling resources and applied it in our cloud computing platform OnceCloud which is similar with the IaaS cloud computing platform Amazon EC2. We use experiments to evaluate this method for maximizing profit in the following sections.

Table 1 Profit Attributes of Different Requests

	MaxSLA	$T_{defined}$(ms)	$T_{expired}$(ms)
browse	100	1000	2000
addCart	300	1000	3000
purchase	700	2000	6000

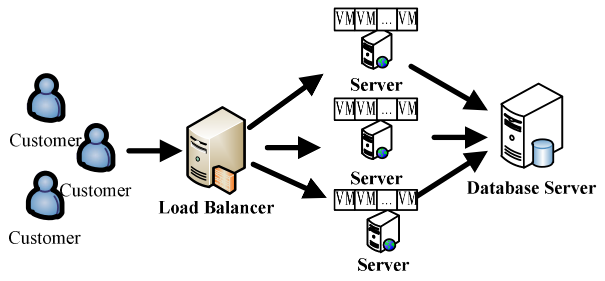


Figure 5 Experiment Architecture

A. *Experimental environment*

We use an on-line bookstore application in our benchmark Bench4Q confirming to the TPC-W specification as our target application. For simplicity, we choose three request types including browsing, shopping and ordering. Fifty percents of all the simulated users send browsing, shopping and ordering requests in sequence. Fifty percents of all the simulated users browse, and then depart. Meanwhile, the users whose requests are timed out do not send the next request, and the access procedure is over.

Requests are organized as sessions (i.e., the same users send a series of continuous requests). The session completing a deal brings profit for an application. The sessions without purchasing and the sessions timed out cannot bring profit for an application. Thus, we define different SLA loss functions for different types of requests.

Figure 5 describes the experimental environment including three servers, a database server, a load balance and a simulator. The physical resources are allocated as virtual machines, and organized as a web application server cluster. The experimental period is divided as initializing period 60 seconds, testing period 300 seconds, and finishing period 30 seconds. The simulator only sends requests without recording test results in the initializing period and finishing period. We calculate processing result during the testing period.

From the analysis of the system requirement, the influences of three requests are different from the application as follows. The browsing requests cannot introduce profit, so they are the least important; the shopping requests can introduce underlying profit, so they are in the middle; the purchasing requests can introduce immediately profit, so they are the most important. Table 1 gives the importance for every type request. In addition, we adjust the resource allocation every ten seconds, and the cost of renting virtual resources is twenty units.

B. *Experimental results*

To validate our method, we compare our method with the algorithm in [1], and the experimental results are described in figure 6.

The 1VM4ALL scheduling algorithm uses a single virtual resource to process all of the incoming requests. The new arrived request is added in the tail of the waiting queue. The application always rents only one BTU to process all of the incoming requests, so the cost of renting virtual resources is lowest. However, lots of requests cannot be processed in time, so the loss of SLA profit is the maximum.

The 1VMPerReq scheduling algorithm is at the other extreme. This algorithm does not put all of the requests in a queue, but allocates an idle BTU to process a new request. In the period of processing requests, most of the requests can be processed in time, so the loss of SLA profit is minimized. However, as lots of virtual resources are rented, the cost of renting resource increases significantly.

The optimized online bin packing algorithm allocate a request to a running machine, and the idle time of a BTU ought to match the required time of a request. This algorithm aims at maximizing the resource utilization, but cannot consider the definition of SLA. Thus, some requests with high priority cannot be processed in time, and the overall cost increases.

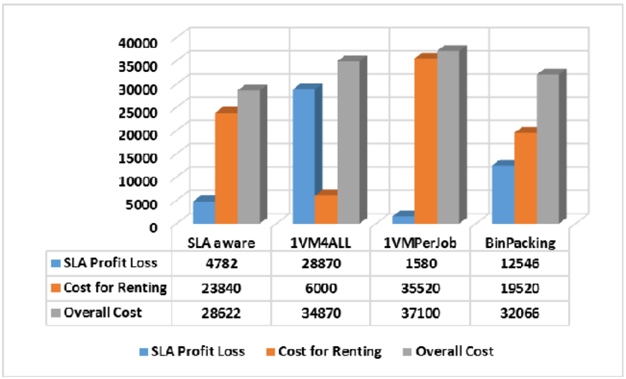
Compared with the above methods, the loss of SLA profit of our method is higher than that of 1VMPerReq, and cost of renting virtual resources of our method is higher than that of 1VM4ALL and Bin Packing. However, because we consider both the loss of SLA profit and the cost of renting virtual resources, our method has the overall lowest cost. Compared with 1VMPerReq, 1VM4ALL and Bin Packing, our method decreases the cost of application providers in 17.92%, 22.85% and 10.74%.

VII. Related work

Cloud computing is a hot research topic in recent years. The current researchers aim at improving the resource utilization and decreasing the number of rented physical resources. Li et al. assumed that a certain number of requests were processed in a period [2]. This work aimed at maximizing resource utilization to guarantee QoS with deploying and scheduling strategy in the perspective of an application provider. In contrast, this paper aims at finding the balance point between the loss of performance and the cost of renting physical resources.

SLA is an active topic in the service-oriented computing, and there are many specification in the field (e.g., WSLA, WS-Agreement). Alhamad et al. considered the SLA of applications in the cloud computing environment, including conceptual frameworks, and some service level object [3]. However, these works did not consider the SLA defined by customers. Leitner et al. faces the profit defined by customers to maximize the profit by scheduling requests [4]. However, this work did not consider the cost of renting physical resources. Our work considers both the cost of renting resources and the loss of SLA profit.

Satzger et al. introduced the idea of scheduling grid work [7]. This work uses the queueing theory to model the performance of applications to improve application performance and resource utilization. Lee et al. introduced a dynamic priority scheduling algorithm [8]. This work focused on the scheduling method for cloud operators (e.g., scheduling virtual resources to physical machines). These methods pay attention to scheduling requests from clients and resources on servers, but do not take into account the customers' profit defined in the SLA.



	SLA aware	1VM4ALL	1VMPerJob	BinPacking
🟦 SLA Profit Loss	4782	28870	1580	12546
🟧 Cost for Renting	23840	6000	35520	19520
⬜ Overall Cost	28622	34870	37100	32066

Figure 6 Comparison of Profit Loss

VIII. Conclusion

This paper proposed a profit-driven resource scheduling method for virtualized cloud systems in a cloud computing environment. Compared with traditional methods, our method considers both the profit from SLA and the cost of renting virtual resources in the cloud platform from the customers' perspective. This paper uses the queueing theory to estimate the system performance; calculate the profit brought from SLA according to the predicted performance; use a hill climbing algorithm to maximize profit by adjusting the number of rented BTUs. We have implemented a framework for scheduling virtual resources, and applied it in our IaaS cloud platform OnceCloud. The experimental results demonstrate that our method decreases more than ten percent of the cost compared with traditional methods.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No.61173004.

References

[1] Genaud S, Gossa J. Cost-wait Trade-offs in Client-side Resource Provisioning with Elastic Clouds[C]//Proceedings of the 4th International Conference on Cloud Computing, Washington DC, USA, 2011. Washington DC, USA: IEEE, 2011: 1-8.

[2] Li J, Su S, Cheng X, et al. Cost-Conscious Scheduling for Large Graph in the Cloud[C]//Proceedings of International Conference on High Performance Computing and Communications, Banff, Canada, 2011. Washington DC, USA:IEEE, 2011:808-813.

[3] Alhamad M, Dillon T, Chang E. Conceptual SLA Framework for Cloud Computing[C]//Proceedings of International Conference on Digital Ecosystems and Technologies, Dubai, United Arab Emirates, 2010. Washington DC, USA:IEEE, 2010: 606 - 610.

[4] Leitner P, Hummer W, Dustdar S. Cost-Based Optimization of Service Compositions[J]. IEEE Transactions on Services Computing, 2013, 6(2): 239-251.

[5] Satzger B, Hummer W, Leitner P, et al. ESC: Towards an Elastic Stream Computing Platform for the Cloud[C]//Proceedings of International Conference on Cloud Computing, Washington DC, USA, 2011. Washington DC, USA:IEEE, 2011:348-355.

[6] Lee Z, Wang Y, Zhou W. A Dynamic Priority Scheduling Algorithm on Service Request Scheduling in Cloud Computing[C]//Proceedings of International Conference on Electronic and Mechanical Engineering and Information Technology, Harbin, China, 2011. Washington DC, USA:IEEE, 2011:4665–4669.