International Journal of Grid and Distributed Computing
Vol. 9, No. 9 (2016), pp.193-204
http://dx.doi.org/10.14257/ijgdc.2016.9.9.17

A Resource Reservation based Framework for QoS-aware Resource Provision in Cloud Computing

Hong He

Department of Computer and Communication, Hunan Institute of Engineering
**hhong1970@126.com*

Abstract

Recently, service of quality (QoS) of cloud platform has attracted more and more attentions as plenty of commercial clouds are deployed in real world. However, providing desirable QoS for end-users still remains a challenging problem because of the unpredictable workload in cloud platforms. In this paper, we present a novel QoS framework called QVRP, which enables us to enhance user's QoS satisfactory through a flexible virtual machine reservation mechanism. The implementation of QVRP is implemented as an integrated middleware that makes it easy to be incorporated in many cloud platforms. Extensive experiments are conducted to examine the effectiveness and performance of QVRP, and the results indicate that it can significantly improve many important QoS metrics, which in turn improves the cloud user's QoS satisfactory.

Keywords*: Cloud Computing; QoS; Resource Reservation; Task Scheduling*

1. Introduction

Cloud computing is emerging as an alternative to conventional high-performance and distributed systems [1,2]. It integrates the concepts of grid, cluster, and utility computing as a unified infrastructure, which provides a highly available, scalable, and flexible network-based platform for various kinds of applications. By using virtualization platforms, each cloud application is hosted in an individual virtual machine (VM) so as to provide the required isolation and protection from other applications [3,4,5]. When deploying cloud applications, resource providers are required to optimize selection of virtual resources, which can be independent or belong to a multi-component service, amongst different clouds must take into account requirements such as configuration of individual resources, aggregated service performance, total cost and *etc* [6,7]. As more and more commercial cloud platforms are developed, user's QoS becomes an important issue when cloud providers are operating their IT-infrastructures. However, to the best of our knowledge, there is still no integrated framework or middleware that can provide desirable user QoS in cloud platform [8].

According to many existing studies in [9,10,11], virtual resource provision plays a key role when providing end-to-end QoS for cloud users. For example, if the resource is over-provisioned by cloud providers for ensure better QoS, the resource utilization will be decreased which in turn reduces the benefits of commercial cloud platforms. On the other sider, if the resource is under-provisioned for maximal resource utilization, cloud user's QoS satisfactory might be compromised due to unpredicted resource performance. So, how to obtain better QoS and maximize benefits for cloud providers at the same time becomes a difficult problem due to its conflict nature. To deal with this problem, in this paper we present an integrated resource provision framework, namely QoS-aware Virtual Resource Provision (QVRP), in which user's QoS can be better guaranteed by using a novel elastic VM reservation mechanism.

ISSN: 2005-4262 IJGDC

The rest of this paper is organized as following. In Section 2, we summarize the related work. In Section 3, we present the overview framework of the proposed middleware. In Section 4, the implementations of VM reservation mechanism are presented in details. In Section 5, extensive experiments are conducted to evaluate the performance of the proposed system. Finally, Section 6 concludes the paper with a brief discussion of our future work.

2. Related Work

Recently, many researchers have take efforts to study the approaches for improving the QoS in cloud platform. We briefly summarize them as following.

In [12], Goiri et al. proposed a resource-level metric for specifying fine-grain guarantees on CPU performance. This metric allows resource providers to allocate dynamically their resources among running services depending on their demand. This is accomplished by incorporating the customer's CPU usage in the metric definition, but avoiding fake SLA violations when the customer's task does not use all its allocated resources. In [13], the authors presented a Service Deployment Management System for Optimization (SDMS-O) designed with a novel optimization approach for service deployment to improve deployment efficiency and reduce deployment cost while guaranteeing the users' QoS requirements. In SDMS-O, atom-services as its basic units of service applications are first divided into different service families according to compatibility and installation policy, and a service deployment requirement is expressed as an installation expression sequence. In [14], the authors proposed a workflow system architecture which enforces QoS for the simultaneous execution of multiple scientific workflows over a shared infrastructure. This approach involves multiple pipeline workflow instances, with each instance having its own QoS requirements. In [15], the authors investigated if latency in terms of simple ping measurements can be used as an indicator for other QoS parameters such as jitter and throughput. Their experiments were carried out on a global scale, between servers placed in universities in Denmark, Poland, Brazil, and Malaysia. The results show the correlation between latency and throughput, and between latency and jitter, even though the results are not completely consistent. In [16], the authors proposed an Adaptive-Scheduling-with-QoS-Satisfaction algorithm for the hybrid cloud environment to raise the resource utilization rate of the private cloud and to diminish task response time as much as possible. In [17], the authors suggested that fault tolerance and QoS scheduling using CAN (Content Addressable Network) in Mobile Social Cloud Computing (MSCC). In [18], the authors propose a QoS-Aware Resource Elasticity (QRE) framework that allows service providers to make an assessment of the application behavior and develop mechanisms that enable dynamic scalability of cloud resources hosting the application components.

3. Framework of QVRP

3.1 QoS Attributes in QVRP

In open cloud environments, user-oriented QoS attributes have attracted more and more attends nowadays, because both cloud providers and cloud users are involved in an opening resource market and aims to maximize their utilities. According to existing studies, the QVRP-CLOUD defines four user-oriented QoS attributes for providing QoS-aware resource provision service for users. We briefly describe these attributes as following:

(1) Trust and Reputation: It is a subjective measure of the perception that members of a social network has of one another. This perception is based on past experiences. The

reputation ranks aggregate experiences of all members of a virtual organization, sometime crossing multiple virtual organizations. There are many existing studies on how to model and evaluate user's reputations, and how to calculate the trust attribute by using reputation mechanisms.

(2) Dependability and Fault-tolerance: In cloud environments, all IT-infrastructures are virtualized as a set of VM instances, which are provisioned in an on-demand manner. Therefore, conventional approaches of improving system's dependability are not suitable for cloud platforms. On the contrary, user-oriented dependability should be measured in terms of a combination of VM's dependability, user's resource demands, and system's resource provision policy.

(3) Energy-efficiency: Cloud platform contributes to reduce energy consumption by consolidating workloads from different users in a smaller number of physical servers, so as to power off those unused resources. Such an energy-consumption optimization might result in significantly degradation of system's performance, such as execution time delay, longer responsiveness and *etc*. Therefore, the key concern to confront is the tradeoff between performance and energy consumption, i.e., to minimize energy consumption but still accomplishing desired QoS. To address the tradeoffs, energy efficiency must be treated equal as the other critical parameters, already including service availability, reliability, and performance.

(4) Economic feasibility and Competitiveness: Current commercial providers offer a variety of capabilities under different pricing schemes, but it is hard to differentiate among the offerings without sufficient knowledge of the repercussions on internal performance, ecologic, and economic goals. To improve this situation, more complex economic models are needed. These models must include features so as to compare economical utilities between alternative configurations. To this end, such models must employ business related terms that can be translated to service and infrastructure parameters during development and deployment of services.

3.2 System Framework

In Figure 1, we depict an overview of the components of the framework, which provides a layer allowing the framework to be deployed over different virtual organizations and conventional platforms.

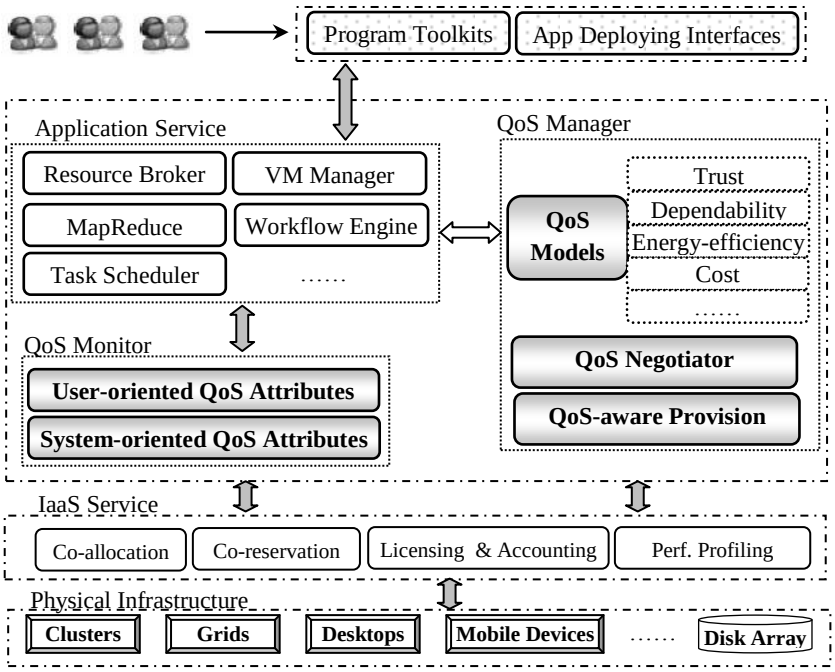

Figure 1. Framework of QoS-aware Resource Provision

In the Application Service container, there are a set of services which provide resource allocation and task management functions for executing user applications. For example, Resource Broker service is responsible for representing user applications when requesting virtual resources; VM manager is responsible for managing a set of VM instances (often called VM pool) with aiming at improving resource utilization of the system; Workflow Engine and Task Scheduler services often provide a set of interfaces for deploying large-scale workflow applications on cloud-based platforms; as to MapReduce, it can translate a certain kind of data-intensive application into a normalized two-phase model and deploying them onto a set of virtual execution nodes. Generally speaking, the Application Service container will provide a suitable execution platform for various user applications.

The QoS Manager container is the key components in our QVRP framework. Firstly, it provides a set of QoS Models for each QoS metric. By these models, we can evaluate or describe QoS performance from different aspects. For example, before running a volunteering application, we can use trust and dependability models to evaluate the QoS performance of availability nodes, and then select some of them for application execution. If the goal is to saving energy consumption for running applications, energy-efficiency model will be used to describe the energy-efficiency of a given VM allocation scheme. If the goal is to improving the energy-efficiency of underlying virtual resources, energy-efficiency and costs model will be used to evaluate current VM provision scheme, and them the QoS-aware Provision service will be called for configuring the underlying VM pool if necessary.

Besides the QoS Models and QoS-aware Provision service, QoS Negotiator component is also a key service in our QVRP framework. When running user application with specific QoS requirements, Resource Broker service will interact between applications and cloud providers through QoS Negotiator. Typically, the QoS negotiation process aims to reach some agreement between resource consumers and providers on either the reservation schedule, or the parameters involved in providing a given service. It is not necessary for such a process to take place – especially if the service provider can meet the request made by a client immediately. However, if the constraints identified in the service request by an application can not be satisfied, it is necessary for the service provider and

user to agree on some mutually agreeable constraints. Therefore, the QoS Negotiator component is essentially for a matchmaking process between the application's desired QoS constraints, and the service provider's resource capacity, to ensure the request does not exceed such resource capacity.

With respect to QoS monitor, it consists of two services which monitor the user-oriented and system-oriented QoS metrics for underlying virtual resources in an online manner. At present, the implementation of this component is deprived from our previous work, namely CPMEM-ClOUD [19], which is aiming to provide users and researchers an easy-to-use toolkit to evaluate the runtime performance of cloud systems.

4. Resource Provision Mechanism in QVRP

To achieve the goal of QoS-aware resource provision, we incorporate two mechanisms in the QVRP framework. Firstly, a VM reservation mechanism is implemented for guaranteeing the availability of virtual resources. Then, we design an adaptive resource provision policy which can dynamically configure the setting of VM pool according the changing of workload as well as the system's QoS performance.

4.1 VM Reservation

Performance reports on practical clouds have indicated that unpredictable workload and dynamic availability of resources are the two significant characteristics in cloud environments. Therefore, it is difficult for applications to precisely estimate the reservation time, if not impossible. As a result, user applications tend to overestimate the reservation time so as to ensure their successful execution. This behaviour inevitably results in high rejection rate and low resource utilization. Motivated by these observations, in the QVRP we implement a *Two-Dimension Relaxed Reservation Policy* (TDRRP), which can accept the reservation requests that overlaps with existing ones under certain conditions.

A reservation request can be characterized as a 3–tuple $<ts,te,res>$, where ts is the reservation start time, te is reservation deadline, res refers to resource demands. A time-slot can be characterized as a 3–tuple $<slot\_ts,slot\_te,slot\_res>$, where $slot\_ts$ is the start time of the time-slot, $slot\_te$ is the deadline of the time-slot, $slot\_res$ refers to the available resources between the duration $[slot\_ts,slot\_te]$. Given a time-slot $slot_k$ and a reservation request req_i, $slot_k$ can rigidly meet the requirements of req_i if $slot\_ts_k \le ts_i \cap slot\_te_k \ge te_i \cap slot\_res_k \ge res_i$

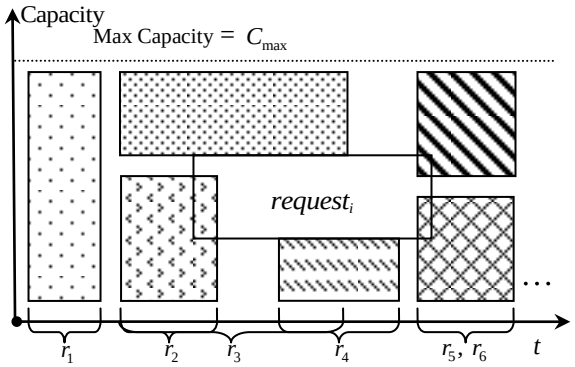

Figure 2. An Example of Time Slot Table

Given current time is t_0, a time-slot table is shown in Figure 2. The existing reservations are illustrated by rectangles with texture. Considering a reservation request $request_i$ arrives, Reservation Manager (RM) finds that no available free slot can rigidly meet the requirements of $request_i$. However, RM notices that a free slot between r_2 and r_5

seems to be a good candidate, except that the start time and the deadline of $request_i$ are slightly overlapping with other existing reservations. If RM reserves this time slot to $request_i$, then the start time of $request_i$ can not be guaranteed because of r_2. Meanwhile, $request_i$ will overlap with the start time of r_5 and r_6.

As mentioned before, reservation requests usually tend to overestimate deadline to ensure their successful completion. If the actual deadlines of r_2 are earlier than the start time of $request_i$, and the actual deadline of $request_i$ is earlier than the start time of r_5 and r_6, then, this time slot is feasible for $request_i$ in practice. So, our strategy takes such overestimation into account, and tries to accept some requests, whose reservation requirements can not be met in conventional way. Given a time-slot $slot_k$ and a reservation request req_i, let random event $\mathbf{E_i}$ represent that no reservation violation occurs on req_i. Then, $slot_k$ can relaxed meet the requirements of req_i with the probability $\Pr\{\mathbf{E}_i\}$ as following

$$\Pr\{\mathbf{E_i}\} = \Pr\{slot\_ts_k \le ts_i \cap slot\_te_k \ge te_i \cap slot\_res_k \ge res_i\} \tag{1}$$

Considering a reservation request req_i, denoted as $<ts_i, te_i, res_i>$, arrivals to a RM. To meet the requirements of req_i, RM should find a free slot which satisfying $C_{free} \ge res_i$ during the period $[ts_i, te_i]$, where C_{free} is the amount of free capacity. If there is no such time-slot that can rigidly meet the requirements of req_i, there must be one ore more existing reservations that overlapping with the reservation time of req_i. As shown in Figure 2, it is clear that these existing reservations can be classified as two kinds:

(1) The start time overlapping set of req_i is those existing reservations that their deadlines overlap with req_i, denoted as

$$set_i^s = \{req_j \mid \forall j \in [1 \ldots i-1],\ st_j < st_i < et_j\} \tag{2}$$

(2) The deadline overlapping set of req_i is those existing reservations that their start time overlap with req_i, denoted as

$$set_i^e = \{req_j \mid \forall j \in [1 \ldots i-1],\ st_j < et_i < et_j\} \tag{3}$$

Clearly, if $set_i^s = \varnothing \cap set_i^e = \varnothing \cap C_{free} \ge c_i$, then request req_i can be guaranteed. This is also the criterion of conventional reservation mechanism while accepting reservation requests. As shown in Figure 1, if the RM decides to accept req_i, then the start time overlapping set of req_i is $set_i^s = \{r_2, r_3, r_4\}$, and the deadline overlapping set is $set_i^e = \{r_4, r_5, r_6\}$. Given there has been $i-1$ existing reservations when req_i arrivals. To know the risk of accepting req_i, we should figure out the probability of reservation violation if it is accepted. Let random event $\mathbf{E_i}$ represent that no reservation violation occurs on req_i, random event $\mathbf{E_i^s}$ represent that all the reservations in set_i^s do not incur the violation of req_i. Let random event $\mathbf{E_i^e}$ represent that req_i do not incur any violation on all the reservations in set_i^e. We assume that all reservation requests are independent, so the probability of $\mathbf{E_i}$ can be expressed as

$$\Pr\{\mathbf{E_i}\} = \Pr\{\mathbf{E_i^s}\} \cdot \Pr\{\mathbf{E_i^e}\} \tag{4}$$

For a job with m reservation requests, the probability of successful reservation for this job can be expressed as

$$\Pr\{\mathbf{E}\} = \prod_{i=1}^{m} \Pr\{\mathbf{E_i^s}\} \cdot \Pr\{\mathbf{E_i^e}\} \tag{5}$$

where $\mathbf{E}$ is a random variant representing that relaxed reservation policy does not result in reservation violation. To calculating Pr{**E**}, we need to know the probability model of VM service time. Such a probability model can be obtained by analyzing the logs of QoS Monitor services. For example, if the VM service time of requests following Exponential distribution with rate μ, then its cumulative distribution function can be expressed as $F(t)=1-e^{-u\cdot t}$. So, we can easily have as following.

$$\Pr\{\mathbf{E_i^s}\}=\prod_{j\in set_i^s}(1-e^{(t_i-t_j)\cdot\mu}) \tag{6}$$

$$\Pr\{\mathbf{E_i^e}\}=\min_{i\in set_i^e}\{1-e^{(t_i-t_j)\cdot\mu}\} \tag{7}$$

When cumulative distribution function of task execution is not unknown, we can use the following approach to obtain the probability of Pr{**E**}. Let random variable $TE_j(j\in[1\ldots i])$ represent the actual deadline of reservation request r_j $(r_j\in set_i^s)$, then the probability that set_i^s do not incur the violation of req_i can be calculated as

$$\Pr\{\mathbf{E_i^s}\}=\Pr\{TE_{j_1}\le t_i\bigcap TE_{j_2}\le t_i\bigcap\ldots\bigcap TE_{j_n}\le t_i\mid slot\_res+\sum_{j\in J}res_j>res_i\} \tag{8}$$

where set $J=\{r_{j_1},r_{j_2},\ldots r_{j_n}\}$ and $J\subseteq set_i^s$.

It is clear that there might be many set J that can satisfy the condition of (4). For example, $set_i^s=\{r_2,r_3,r_4\}$ as shown in Figure 1, then $J=\{r_2\}$ or $J=\{r_3\}$ or $J=\{r_2,r_3\}$ can all be applied in (4) to calculate $\Pr\{\mathbf{E_i^s}\}$. So, the question is which one is the optimal J^*, which can be expressed as a programming problem as following

$$\begin{aligned}&\max\quad \Pr\{\mathbf{E_i^s}\}\\ &s.t.\quad slot\_res+\sum_{j\in J^*}res_j\ge res_i\\ &\qquad J^*\subseteq set_i^s\end{aligned}$$

The approach to find J^* is the exact the same as 0-1 knapsack algorithm [XX]. As to $\Pr\{\mathbf{E_i^e}\}$, let random TE_i represent the deadline of req_i, and $\{r_{k_1},r_{k_2},\cdots,r_{k_n}\}$ represent the reservations in set_i^e. Assuming that $\{r_{k_1},r_{k_2},\cdots,r_{k_n}\}$ is sorted in ascending order of reservation start time, it is clear that we should find the k_l that satisfying

$$\sum_{j=k_1}^{k_{l-1}}res_j\le res_{\max}-res_i<\sum_{j=k_1}^{k_l}res_j \tag{9}$$

where $res_{\max}$ is the max capacity of the resource. As long as the actual deadline of req_i is earlier than ts_{k_l}, we can ensure that req_i will not incur any violations on all the reservations in set_i^e, and the probability that req_i won't lead to reservation violation is

$$\Pr\{\mathbf{E_i^e}\}=\Pr\{TE_i\le ts_{k_l}\} \tag{10}$$

4.2 Algorithm of Adaptive Resource Provision

To maintain desirable QoS performance, a cloud system often keeps a set of active VM instances in its VM pool. The VM instances might come from different virtual organizations or IT-infrastructures. Administrators can also define the allocation policy to be adopted for resource provisioning. For example, if current VM pool is not enough, or in case of unavailability of the provider, other cloud providers will be selected for configuring more VM instances.

For the intent of supporting execution of applications with enhanced QoS, a novel provisioning policy has been developed in our QVRP, which is deprived from the famous Hot-Spot VM Provision (HSVP) in Amazon's EC2. In HSVP, users send price-bids that represent the maximum value they are willing to pay for using each VM instance; the system periodically updates price for Hot-Spot VMs. Whenever the Hot-Spot price is equal or smaller than the bid price, corresponding VMs are allocated to user applications. Our adaptive VM provision algorithm is shown as following.

Adaptive VM provision algorithm in QVRP

Begin
1. $r := 0$;
2. **while** $r < r_{max}$ and not find feasible solution of problem (5) **do**
3. find a possible invoke path ip_x which satisfies the maximum number of QoS constraints;
4. **for** each service j **do**
5. start a QoS negotiation process with each service provide p_j, and calculate each attributes according (6)
6. **if** negotiation is successful with one provider **then**
7. break out while loop;
8. **else** r:= r+1;
9. **end if**
10. **end for**
11. **end while**
12. **if** a feasible solution is obtained **then**
13. **return** TRUE
14. **else**
15. **return** FAIL

end.

5. Experiments and Performance Comparison

5.1. Experimental Settings

In this section, we conduct a set of experiments on the simulator CloudSim V3.0.3 [20]. In order to use a realistic distribution of job metrics, trace inputs to the simulation system were acquired from the EGEE Observatory and converted to the format compatible with CloudSim. The traces consisted of sets of job logs, including submission times, lengths, resource demands, and the corresponding QoS requirements. To comparing the performance of QVRP-CLOUD with other virtual resource provision policies, we implement QoS Matching Policy (QMP), Profit-Driven Policy (PDP), Feedback-Control Policy (FCP). In the experiments, the default resource provision implemented in CloudSim simulator is used as baseline for performance comparison. In the experiments, the parameter *Resource Requirement Intensity* (RRI) is used to represent the intensiveness of user's resource demands in a given period. All the experiments are conducted 4 times, each with a different RRI parameter.

5.2 Comparison of QoS metrics

In this section, we mainly investigate the resource utilization rate (RUR) metric and job rejection rate (JRR) metric. By comparing these two QoS metrics, we can clearly conclude whether a cloud platform can provide desirable QoS or not. The results are shown in Figure 3 and Figure 4.

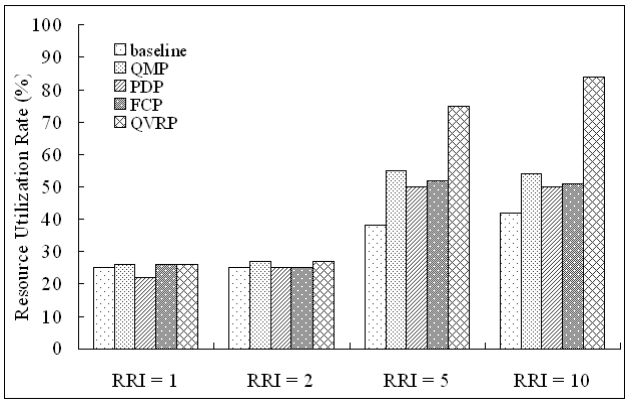

Figure 3. Resource Utilization Rate Comparison with Different RRI Parameters

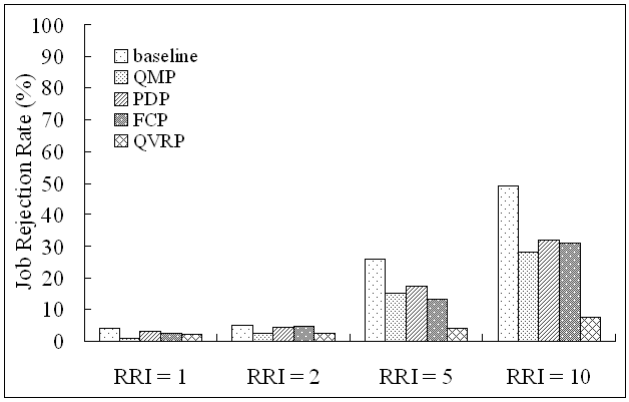

Figure 4. Job Rejection Rate with Different RRI Parameters

For baseline policy, when RRI parameter is gradually increased from 1.0 to 10, resource utilization also increases from about 24% to about 41%. It seems that RRI=5 is a turning point for RUR metric, which means that higher RRI can improve RUR while such a improvement can not be continued if RRI is higher than 5. This is because that many users' jobs are rejected either by excessive resource requirements or deadline expiration. Such a case often can be seen in many practice cloud platform, which applies static resource provision policy. It is well known that static resource provision policy often fail to immediately enlarger resource supply when the resource demand increased quickly. So, if some QoS controlling schemes are used (such as job admission) the system has to face both low resource utilization and higher job rejection rate.

When applying QMP, RUR metric is increased about 24% comparing with the baseline policy when RRI = 5. Besides our QVRP policy, the performance of QMP is the best in the four polices when RRI is in high-level. This is because that QMP applies SLA negotiation mechanism, which enables users to adjust their resource requirements if it is necessary. As a result, fixed resource supply can better meet the changing of resource demands. However, we also notice that SLA negotiation mechanism seems can not further improve the RUR metric when RRI is increased to 10. This is because the resource requirements becomes very overwhelming comparing with resource supply when RRI = 10. So, plenty of jobs have to spend longer time for QoS negotiation. In this experiment, we use simulator which configuring the network costs as constant. When in practical cloud platform, network costs spent on QoS negotiation will be more serious and unpredictable, which in turn will lead to more performance degradation.

To clearly explain the advantage of QVRP policy, we shown the job rejection rate results of all experiments in Figure 4. It is clear that the JSS metrics of QVRP in all cases are significantly lower than other policies especially when RRI is in high-level. More importantly, when RRI parameter is increased from 1 to 10, the JRR metric only increases from 2.1% to 7.5%. This is the key reason that QVRP can obtain better performance in term of RUR. In practical cloud platform, users are generally difficult to predict their real resource requirements especially when the performance of virtual resources is highly dynamically because of unpredictable overheads. So, many cloud users tend to overestimate their resource requirements so as to keep better QoS performance. However, if all users do the same thing, it will result in significantly performance degradation in the whole system. The QVRP takes this observation into consideration, so its relax reservation policy makes the cloud system achieve best JRR which in turn significantly improves the RUR metric.

In the second experiment, we investigate the task response time metric since it is a key QoS metric for cloud users. In our experiments, the tested workload consists of a set of jobs which arrive the target cloud platform continuously. It is well-known unsuitable resource provision policy will result in the increasing of *Mean Response Time* (MRT). The overall MRT is shown in Figure 5.

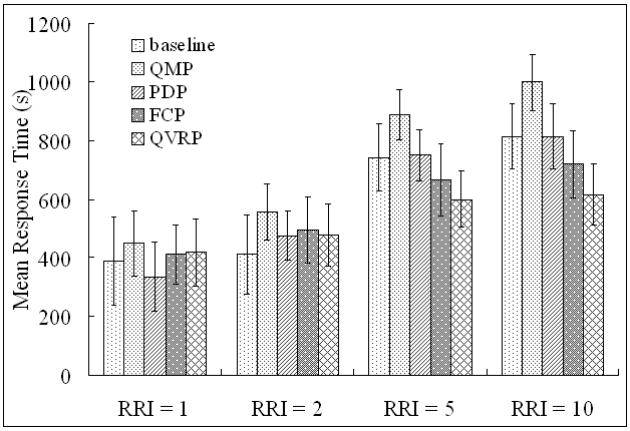

Figure 5. Mean Response Time with Different RRI Parameters

As shown in Figure 5, the MRT metrics of PDP, FCP and QVRP are almost the same when the system's RRI parameter is relatively low (RRI = 1 and 2). In these cases, baseline policy performs best in all experiments. When the RRI parameter is increased to 5 and 10, the MRT metric of baseline increases quickly. Such a result indicates that static resource provision is not suitable for the workload with intensive resource requirements. Among the five tested policies, we noticed that QMP performs worst in the term of MRT in all cases. By carefully investigate the experimental logs, we find that the time of SLA negotiation is about 5% to 17% of the total response time when using QMP. More importantly, time of SLA negotiation tends to increase with the increasing of RRI parameter.

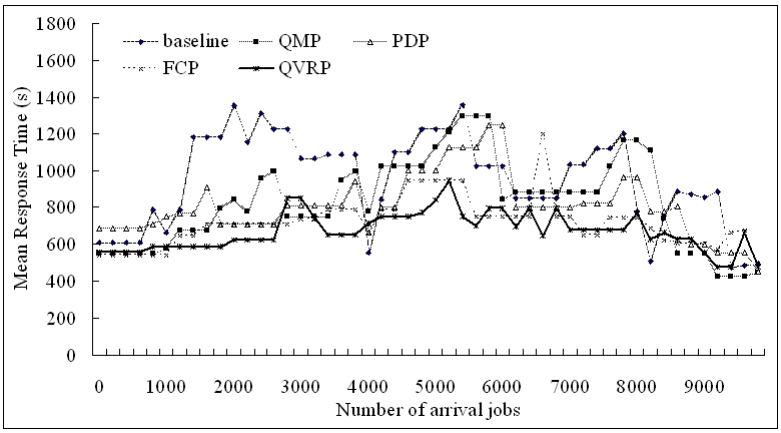

Figure 6. Runtime Mean Response Time (RRI = 10)

To further investigate the experimental results, we log the MRT at runtime as shown in Figure 6 (RRI = 10). As to PDP, we find that the key factor that influences the MRT metric is the resource price. For example, when the overall resource price is in low-level, the MRT metric is often longer than that of higher price resources. With respect to FCP, we find that RRI is the key factor that influences the MRT metric. As to our QVRP, its MRT metric also gradually increased with RRI parameter. However, such an increasing is slower than other policies. By the results in Figure 6, it is clear that the logged MRT of QVRP is always kept in a low-level, however, the MRTs of other policies generally fluctuate dramatically. This indicates that workload characters also have significantly effects on MRT metric. So, by this experimental result, we can conclude that QVRP can provide more stable QoS performance for users than other existing provision policies.

6. Conclusion

In this paper, we propose an integrated resource provision framework, namely QVRP, in which user's QoS can be better guaranteed by using a novel elastic VM reservation mechanism. The experimental results indicate that QVRP can better deal with the workload with intensive resource requirements. This feature is very important for many practical cloud platforms since peak resource demands often drive cloud providers equipped their IT-infrastructure with more and more servers, which in turn increases the building and operational costs of cloud infrastructures. In the future, we are planning to incorporate it with workload consolidation mechanism for optimizing some specific QoS metrics, such as energy consumption and throughput.

References

[1] Dikaiakos, M.D., et al., Cloud Computing Distributed Internet Computing for IT and Scientific Research. Ieee Internet Computing, 2009. 13(5): pp. 10-13.
[2] Leiba, B., Having One's Head in the Cloud. Ieee Internet Computing, 2009. 13(5): pp. 4-6.
[3] Canali, C. and R. Lancellotti, Improving Scalability of Cloud Monitoring Through PCA-Based Clustering of Virtual Machines. Journal of Computer Science and Technology, 2014. 29(1): pp. 38-52.
[4] Xiao, Z., W. Song, and Q. Chen, Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment. Ieee Transactions on Parallel and Distributed Systems, 2013. 24(6): pp. 1107-1117.
[5] Shiraz, M., et al., A study on virtual machine deployment for application outsourcing in mobile cloud computing. Journal of Supercomputing, 2013. 63(3): pp. 946-964.
[6] Goiri, I., J. Guitart, and J. Torres, Economic model of a Cloud provider operating in a federated Cloud. Information Systems Frontiers, 2012. 14(4): pp. 827-843.
[7] Anselmi, J., et al., The economics of the cloud: Price competition and congestion. Performance Evaluation Review, 2014. 41(4): pp. 47-49.

[8] Pedersen, J.M., et al., Using latency as a QoS indicator for global cloud computing services. Concurrency and Computation-Practice & Experience, 2013. 25(18): pp. 2488-2500.
[9] Chaisiri, S., B.-S. Lee, and D. Niyato, Optimization of Resource Provisioning Cost in Cloud Computing. Ieee Transactions on Services Computing, 2012. 5(2): pp. 164-177.
[10] Islam, S., et al., Empirical prediction models for adaptive resource provisioning in the cloud. Future Generation Computer Systems, 2012. 28(1): pp. 155-162.
[11] Li, C. and L.Y. Li, Optimal resource provisioning for cloud computing environment. Journal of Supercomputing, 2012. 62(2): pp. 989-1022.
[12] Goiri, Í., et al., Supporting CPU-based guarantees in cloud SLAs via resource-level QoS metrics. Future Generation Computer Systems, 2012. 28(8): pp. 1295-1302.
[13] Liu, T., et al., SDMS-O: A service deployment management system for optimization in clouds while guaranteeing users' QoS requirements. Future Generation Computer Systems, 2012. 28(7): pp. 1100-1109
[14] Qi, L., et al., A QoS-aware composition method supporting cross-platform service invocation in cloud environment. Journal of Computer and System Sciences, 2012. 78(5): p. 1316-1329.
[15] Pedersen, J.M., et al., Using latency as a QoS indicator for global cloud computing services. Concurrency and Computation-Practice & Experience, 2013. 25(18): p. 2488-2500.
[16] Wang, W.-J., et al., Adaptive scheduling for parallel tasks with QoS satisfaction for hybrid cloud environments. Journal of Supercomputing, 2013. 66(2): p. 783-811.
[17] Choi, S., K. Chung, and H. Yu, Fault tolerance and QoS scheduling using CAN in mobile social cloud computing. Cluster Computing, 2014. 17(3): p. 911-926.
[18] Kaur, P.D. and I. Chana, A resource elasticity framework for QoS-aware execution of cloud applications. Future Generation Computer Systems, 2014. 37: p. 14-25.
[19] Xiao, P. and D. Liu, Configurable performance analysis and evaluation framework for cloud systems. International Journal of Information and Communication Technology, 2013. 5(2): pp. 137-149.
[20] Calheiros, R.N., et al., CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software-Practice & Experience, 2011. 41(1): p. 23-50.

Author

Hong He received his B.S. degree at Wuhan University of Technology in 1996, and M.S. degree at Xiangtan University in 2006. Currently, he works in Hunan Institute of Engineering as an associate professor. His research interesting is grid computing, cloud computing, distributed resource management.

