6-1-2017

Optimizing Cloud-Service Performance: Efficient Resource Provisioning via Optimal Workload Allocation

Zhuoyao Wang

Majeed M. Hayat

Nasir Ghani

Khaled B. Shaban

IEEE Transactions on Parallel and Distributed Systems, Vol. 28, No. 6 (June 1, 2017): 1689-1702. DOI.

Optimizing Cloud-Service Performance: Efficient Resource Provisioning via Optimal Workload Allocation

Zhuoyao Wang
Department of Electrical and Computer Engineering, University of New Mexico
Majeed M. Hayat
Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM
Nasir Ghani
Department of Electrical Engineering, University of South Florida, Tampa, FL
Khaled B. Shaban
Computer Science and Engineering Department, Qatar University, Doha, Qatar

Abstract

Cloud computing is being widely accepted and utilized in the business world. From the perspective of businesses utilizing the cloud, it is critical to meet their customers' requirements by achieving service-level-objectives.

Hence, the ability to accurately characterize and optimize cloud-service performance is of great importance. In this paper a stochastic multi-tenant framework is proposed to model the service of customer requests in a cloud infrastructure composed of heterogeneous virtual machines. Two cloud-service performance metrics are mathematically characterized, namely the percentile and the mean of the stochastic response time of a customer request, in closed form. Based upon the proposed multi-tenant framework, a workload allocation algorithm, termed maxmin-cloud algorithm, is then devised to optimize the performance of the cloud service. A rigorous optimality proof of the max-min-cloud algorithm is also given. Furthermore, the resource-provisioning problem in the cloud is also studied in light of the max-min-cloud algorithm. In particular, an efficient resource-provisioning strategy is proposed for serving dynamically arriving customer requests. These findings can be used by businesses to build a better understanding of how much virtual resource in the cloud they may need to meet customers' expectations subject to cost constraints.

SECTION 1 Introduction

Cloud computing is having a profound effect in today's business world. Many services, including email service, application hosting, data storage, e-commerce and so on, have been implemented on cloud infrastructures. When a cloud consumer (refers to a business in this paper) is deploying a cloud-based service, it is essential for the business to deliver services that satisfy customers' requirements by having adequate computing resources, namely virtual machines (VMs). Meanwhile, it is also important for the business to avoid the cost of having unnecessary VMs (i.e., excessive computing power beyond its need). Therefore, an analytical model that can accurately predict and optimize cloud-service performance is vital as it would be of great benefit to enhancing the quality of service while keeping the cost of the business within a budget.

In recent years, numerous works have looked at modeling cloud services and analyzing the performance. Several performance metrics, such as the response time of a task or a batch of tasks, as well as the throughput and power consumption of cloud services have been analytically characterized [1], [2], [3], [4], [5]. In most of these existing works, the analysis of the cloud-service performance is based upon queuing theory (a detailed review of the related work will be given in Section 2).

Queuing models have proven their value in studying the response time, throughput and stability for cloud services. However, a necessary assumption adopted by the queuing models is that the service rate for all the tasks are the same. Due to the business (namely cost) concern, a modern cloud-service virtual infrastructure typically contains a cluster of heterogeneous VMs. Although the virtual infrastructure can start with near-homogeneous VMs, the facility will likely grow more heterogeneous over time due to upgrades and replacement [5]. Furthermore, it is of great benefit to have an analytical model that could simultaneously address and investigate the essential characteristics and concerns that are critical in operating a cloud service including (i) stochastic response time of a customer request with a general probability distribution, (ii) service-performance optimization by means of workload allocation, and (iii) efficient resource provisioning for serving dynamically arriving customer requests. Therefore and to the best of our knowledge, no existing analytical work is suitable to solve the problem that we considered in this paper.

To fill this gap, we develop a general analytical framework to complement existing models for analyzing and optimizing the computational performance of cloud services. In the proposed framework, we consider a cluster of heterogeneous VMs that are utilized to construct a cloud-service infrastructure and serve customer requests.

The customer request studied here is assumed to be a large-scale application/program that can be divided into several small sub-applications/sub-programs. To serve a customer request, a set of VMs will be utilized to execute the sub-applications of the request concurrently. To speed up the completion of a customer request, either vertical scaling (i.e., utilizing VMs with higher vCPU speed) or horizontal scaling (i.e., increasing the number of vCPUs/VMs to serve the request) can be applied. To make the problem more complicated and realistic, we assume that a request will be divided up to a certain number of sub-applications without degrading the performance. The execution time of a sub-application in the request is assumed stochastic with a general probability distribution. Based on this multi-tenant framework, we analytically characterize two cloud-service performance metrics, namely the percentile and the mean of the response time of a customer request, in closed form. These two metrics have been widely used in cloud service-level agreements (SLAs) and studied in research literature.

With the two service-performance metrics characterized, we proceed to rigorously prove the optimality of the max-min-cloud workload allocation algorithm that was initially proposed (without proof) in our prior work [6]. We then conduct extensive experiments and Monte-Carlo (MC) simulations to examine the efficacy of the max-min-cloud algorithm for executing customer requests with various workload patterns, and for the cases when different probability distributions are considered for the execution times of the sub-applications in the requests. In light of the max-min-cloud algorithm, we further investigate the resource-provisioning-problem in the cloud. In particular, we devise the minimum-provisioning-cost (MPC) provisioning strategy to determine the appropriate amount of virtual resources to be scheduled for efficiently serving the dynamically arriving customer requests. The performance of the MPC strategy is compared with two other practical resource-provisioning strategies, termed the greedy-provisioning (GP) strategy and the random-provisioning (RP) strategy. This is done for two scenarios when either on-demand VMs or reversed VMs are utilized in the virtual infrastructure. To this end, we have utilized the proposed multi-tenant framework to completely address the limitations listed above, which have not been simultaneously addressed by the existing work.

SECTION 2 Related Work

In this section we review three categories of work related to this paper. The first category is the work on the analytical performance modeling of cloud services. The second category of related work is on workload allocation (or task scheduling) in heterogeneous computing and cloud computing environments. The third and last category is on the resource-provisioning-problem in the cloud.

Analytical performance modeling for distributed systems under parallel and grid computing environments has been the focus of attention for a long time. To the best of our knowledge, however, there are only few works to date that have addressed cloud environments. One of the pioneering works was proposed by Xiong et al. [1], where the cloud service environment is modeled as an M/M/1 queuing network. In their model, the arrival and response times of customer requests were assumed to be exponentially distributed. The probability distribution of the response time was characterized by using the Laplace transform. The relationship among the maximum number of tasks, minimum service resources and highest level of service was then determined. Subsequently, many queuing models were proposed to relax the assumptions in [1]and to consider additional stochastic factors that are inherent in the cloud. In [2], Yang et al. used an M/M/m/m+r queue to model the service environment of the cloud. The service response time of a customer request is assumed to be composed of submission, waiting, service and execution times. The probability density function and the mean of the user's response time were derived. Khazaei et al. [3] assumed a general execution time for customer requests as well as a large number of servers in the cloud environment. The authors then modeled the cloud service based upon a M/G/m queuing system, and proposed an analytical technique for performance evaluation based on an approximate Markov-chain model. Due to the fact that statistics of the response time was typically

characterized by using the inversion of its Laplace transform in the queuing models and such inversion was usually done numerically, hence there is generally no closed-form solution by using the queuing models.

Besides the above works based on queuing models, Yeo and Lee [5] considered the heterogeneity of the servers in the cloud. Namely, the authors assumed that the CPU speed of the servers in the cloud are uniformly-distributed random variables, and as such, the response times for executing a customer request also followed an uniform distribution. They then derived statistics (including the mean and variance) of the execution time for a given number of requests. The authors also applied regression methods to estimate the relationship between power and performance over time, and further performed energy-consumption analysis. Recently, our group proposed a multi-tenent model [6] to characterize the mean of the stochastic response time when a group of customers submit their requests to the cloud. The model considered certain essential characteristics of cloud computing including virtualization and multi-tenancy. The impact of the load ratios at servers on the service performance of the cloud was also investigated. We extend our prior multi-tenant model in this paper by analytically characterizing the two cloud-service performance metrics, namely the percentile of the response time of a customer request and the mean of the response time of a customer request, in closed form.

As for the second category of the related work, i.e., the workload allocation (or task scheduling) in heterogeneous computing and cloud environments. A great volume of heuristic allocation/scheduling algorithms based upon queuing models have been proposed for various schemes that are used. For example, Braun et al. in [7] provided a comparison of 11 algorithms through experiments. The experimental results indicated that a genetic algorithm leads to the best performance for all the cases. The min-min algorithm reported in [7] was the second best algorithm but with significantly less computational cost than that of the genetic algorithm. Later, Ritchie and Levine embedded a local-search procedure into the min-min algorithm [8]; they proposed the min-min+LS algorithm that significantly improves the performance of the min-min algorithm but maintains a similar computational cost to that for the min-min algorithm. In [9], Maguluri et al. defined a stochastic model for a cloud service where tasks are assumed to arrive according to a stochastic process and are subsequently queued. The authors focused on studying the stability of the cloud service and developed frame-based non-preemptive VM configuration algorithms. These algorithms can be made nearly throughput-optimal by choosing sufficiently long frame durations. In [6], our group proposed the max-load-first VM mapping algorithm that smartly places the VMs to the physical machines. Simulation results showed that the max-load-first VM mapping algorithm enhanced the performance of cloud computing infrastructures compared to the other two algorithms. In this paper, we propose the max-min-cloud workload allocation algorithm based on the developed multi-tenant model to optimize cloud-service performance. We also provide a rigorous proof to show the optimality of the max-min-cloud algorithm.

As for the third category of related work, we shall discuss the resource-provisioning problem in the cloud. Typically, a resource-provisioning strategy is required to schedule a set of computing resources for serving dynamically arriving consumer requests. An efficient provisioning strategy is able to serve more customers while save on costs. Most of the existing work is based on the macroscopic level [10], [11], [12],[13], i.e., long-term provisioning for large-scale workflows. For example, Yang et al.[11] proposed a profile-based approach for developing just-in-time scalability for cloud applications. As a result, on-demand resources in cloud can be efficiently provisioned. In [12] and [13] , Chaisiri et al. proposed the OVMP algorithm and its refinement, the OCRP algorithm, to minimize the total cost of resource provisioning. The authors also applied stochastic programming to solve the resource provisioning problem. The uncertainty of the demand and provisioning cost is considered in their cloud-service model. The OVMP and OCRP algorithms can optimally adjust the tradeoff between reservation of resources and allocation of on-demand resources. In this paper, we devise a macroscopic-level resource-provisioning strategy, termed the MPC strategy, to determine the appropriate

amount of virtual resources to be scheduled for efficiently serving the dynamically arriving customer requests. The MPC strategy aims to help service providers maximize their profits.

For more details on cloud computing and research challenges, we point interested readers to the following survey papers [14], [15], [16], [17], [18].

SECTION 3 Probabilistic Multi-Tenant Framework for Cloud Services

In this section, we first describe the cloud-service environment and the high-level modeling of the problem to be investigated. The response time of an arbitrary customer request is then analytically characterized. In general, the response time is one of the most commonly adopted performance metrics specified in the cloud SLAs[19], [20], [21]; it is also considered extensively in the research literature [1], [2], [3], [22].

3.1 Cloud Service Environment

Suppose that a business deploys a virtual cloud-service infrastructure by utilizing on-demand or reserved VMs, purchased from cloud providers, to serve its customers. Generally, these VMs are heterogeneous and they may belong to different categories. Each category of VMs has several sub-types that are comprised of varying combinations of attributes, such as elastic computing unit (ECU), number of virtual CPUs (vCPUs), memory, storage, and networking capacity. As an example, Table 1lists four categories of VMs provided by Amazon EC2. Depending upon the purpose of a service, the business can choose the corresponding category of VMs or mixed categories of VMs to run the service. Since this work focuses on analyzing and optimizing the computational performance of cloud services, we assume that the virtual cloud-service infrastructure investigated in this study is comprised of only Compute Optimized and General Purpose [23] VMs, as listed in Table 1.

TABLE 1 Details of VMs Provided by Amazon EC2

Category	Type	ECU	vCPUs	Memory(GB)	Storage	Price(per hour)
	m1.small	1	1	1.7	1x160	$0.022
General	m1.medium	2	1	4	1x410	$0.044
Purpose	m3.large	6.5	2	7.5	1x4 SSD	$0.133
	m4.xlarge	13	4	16	EBSOnly	$0.239
	c1.medium	5	2	1.7	1x350	$0.075
Compute	c3.large	7	2	3.75	2x16 SSD	$0.105
Optimized	c4.xlarge	16	4	7.5	EBS Only	$0.209
	c4.2xlarge	31	8	15	EBS Only	$0.419
Memory Optimized	r3.xlarge	13	4	30.5	1x80 SSD	$0.333
Storage Optimized	i2.xlarge	14	4	30.5	1x800 SSD	$0.853

Category	Type	ECU	vCPUs	Memory(GB)	Storage	Price(per hour)[1]
General Purpose	m1.small	1	1	1.7	1×160	$0.022
	m1.medium	2	1	4	1×410	$0.044
	m3.large	6.5	2	7.5	1×4 SSD	$0.133
	m4.xlarge	13	4	16	EBSOnly	$0.239
Compute Optimized	c1.medium	5	2	1.7	1×350	$0.075
	c3.large	7	2	3.75	2×16 SSD	$0.105
	c4.xlarge	16	4	7.5	EBS Only	$0.209
	c4.2xlarge	31	8	15	EBS Only	$0.419
Memory Optimized	r3.xlarge	13	4	30.5	1×80 SSD	$0.333
Storage Optimized	i2.xlarge	14	4	30.5	1×800 SSD	$0.853

The customer requests studied in this paper are assumed to be large-scale applications/programs. To complete a customer request in a timely fashion, each request will be divided into several sub-applications/sub-programs in advance and executed concurrently. Then, a set of VMs will be scheduled to serve the request. We assume that one vCPU in a VM executes one sub-application. This may be due to various reasons, which include the

thread locks for accessing data in the same sub-application, the high concurrency and synchronization costs among all sub-application or the high overhead for transmitting data stored at different locations, etc. Such service model has been widely utilized due to the multi-tenancy characteristic in modern cloud with the implementation of MapReduce [24] (e.g., Apache Hadoop [25]). To fulfill a request, all sub-applications in the request must be executed. To make the problem more general, the sub-applications may have heterogeneous workload sizes. Specifically, the workload size of a sub-application is the quantity that indicates the number of computation tasks or the volume of data to be processed. A concrete example of a customer request could be the application that collect and process huge amount of data with high variety such as audio, image, video, and etc. Other examples include the distributed bioinformatics computing applications such as parallel/cloud versions of CAP3 [26], DNA-sequencing [27], RNA-sequencing [28], etc.

To make our problem more challenging and realistic, we further assume that a request will be divided up to a certain number of, say n, sub-applications/sub-programs without degrading the performance. (It is noted that how to optimally partition a request is another problem that is out of the scope of this study. We assume that such partition has been given in advance by each request.) To this end, the n sub-applications in a customer request are allocated to and served concurrently by a set of VMs that has at least n vCPUs. The execution time of a sub-application is assumed to be stochastic that follows a general probability distribution with a certain mean value. To guarantee the security and privacy of the cloud service, we assume that a VM can only serve the sub-applications that belong to the same customer request. When a request is completed, the set of VMs that serves the request will be released and becomes available for serving subsequent customer requests.

For convenience, we give a simple example to illustrate how a request is served in the proposed cloud-service environment, which is also shown in Fig. 1. Suppose that a customer submits his/her request consisting of five sub-applications with heterogeneous workload sizes for execution. Suppose also that two VMs in the virtual infrastructure, say VM 1 with four vCPUs and VM 2 with two vCPUs, are scheduled to serve the request. One scenario is that three of the sub-applications in the request are hosted on VM 1 and the other two of the sub-applications are hosted on VM 2, as shown in Fig. 1b. In fact, there are $\binom{5}{1} + \binom{5}{2} = 15$ ways in total to allocate the five sub-applications to the two VMs.

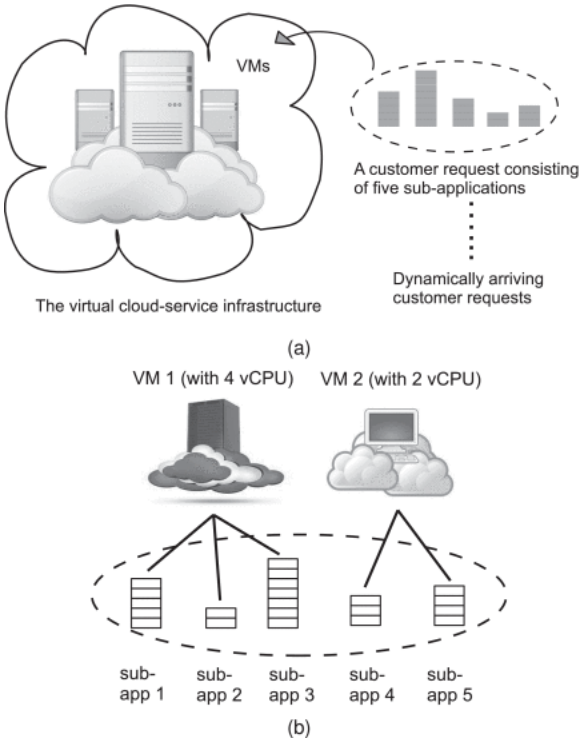

Fig. 1. High-level cloud-service environment.

Given the multi-tenant cloud-service environment described above, two challenging questions are raised.

- Suppose that VM 1 and VM 2 are scheduled to serve the request as illustrated in Fig. 1b, then which allocation pattern of the 15 possibilities will result in the best performance of the request?
- Which VMs (including number and type) in the virtual infrastructure are the most appropriate amount of computing resource to serve the request?

We will specifically answer the above two questions in Sections 4 and 6.

3.2 Cloud-Service Performance Characterization

From a customer's point of view, the response time of his/her request is one of the most important concerns when using the cloud service [29]. We therefore focus on analyzing and minimizing the response times of customer requests as detailed next.

Let $\mathbf{w} = \{w_1, \dots, w_n\}$ be the set representing a customer request. Such a request consists of n arbitrary sub-applications, where w_i is the workload size of the ith sub-application. Based on the cloud-service environment described above, the n sub-applications are executed by n vCPUs concurrently. Let T_{w_i}, for $i = 1, \dots, n$, represent the execution times of the n sub-applications in the request $\mathbf{w}$. In this paper, we assume that the execution times of the n sub-applications dominate the response time of such a request. Hence, the response time of the request $\mathbf{w}$ is determined by the sub-application that is completed last, and we can write

$$T_{\mathbf{w}} = max(T_{w_1}, \dots, T_{w_n}). \quad (1)$$

Note that the execution time of the ith sub-application T_{w_i}, for i= $i = 1, \dots, n$, is assumed to be stochastic. The probability distribution of T_{w_i} is considered to be a general distribution with the execution rate λ_{w_i} that equals the reciprocal of the mean of its execution time.

Next, we analytically characterize two statistics of the stochastic response time of the customer request as key performance metrics to evaluate the cloud service. The first metric is the percentile of the response time of the customer request before time t, which is defined as the cumulative distribution function (CDF) of the response time of the request [1] . In particular, the percentile of the response time gives the probability that request w can be completed before time t. This performance metric has also been considered by other researchers in prior works [30], [31], [32]. Namely, let $PT(\mathbf{w}; t)$ represent the percentile of the response time of the request w. By definition, we have

$$PT(\mathbf{w}; t) \triangleq F_{T_{\mathbf{w}}}(t), \quad (2)$$

where $F_{T_{\mathbf{w}}}(t)$ is the CDF of $T(\mathbf{w})$. To this end, we can further write the percentile of the response time as

$$PT(\mathbf{w}; t) = \prod_{i=1}^{n} F_{T_{w_i}}(t), \quad (3)$$

where $F_{T_{w_i}}(t)$ is the CDF of the execution time of the ith sub-application. Note that when a request $\mathbf{w} = \{w_1, \ldots, w_n\}$ is served by two different sets of VMs, $\mathbf{v}$ and $\mathbf{v}'$, for which the vCPUs in $\mathbf{v}$ are pair-wise faster than for those in $\mathbf{v}'$ (i.e., $\lambda_{w_i} \geq \lambda'_{w_i}$ for $i = 1, \ldots, n$, then

$$PT(\mathbf{w}; t) \geq PT'(\mathbf{w}; t) \text{ for all } t > 0. \quad (4)$$

The second performance metric to be characterized is the mean of the response time of a customer request, which is another widely-adopted performance metric used in cloud SLAs and prior works [21] , [33], [34], [35]. Let $ET(\mathbf{w})$ represent the mean of the response time of the request $\mathbf{w}$. From Equation (1), we can write

$$ET(\mathbf{w}) \triangleq \mathrm{E}[T_{\mathbf{w}}] = \mathrm{E}[max\{T_{w_1}, \ldots, T_{w_n}\}]. \quad (5)$$

Note that for any non-negative random variable X,

$$\mathrm{E}[X] = \int_0^\infty (1 - F_X(x))dx,$$

where $F_X(x)$ is the CDF of random variable X. Hence, $ET(\mathbf{w})$ can be further written as

$$\begin{aligned} ET(\mathbf{w}) &= \int_0^\infty (1 - F_{T_{\mathbf{w}}}(t))dt \\ &= \int_0^\infty (1 - \prod_{i=1}^{n} F_{T_{w_i}}(t))dt\,. \end{aligned} \quad (6)$$

Equation (6) can be numerically evaluated given the knowledge of $F_{T_{w_i}}(t)$ for $i = 1, \ldots, n$.

It is important to note that there is a connection between $ET(\mathbf{w})$ and $PT(\mathbf{w}; t)$, which is

$$ET(\mathbf{w}) = \int_0^\infty (1 - PT(\mathbf{w}; t))dt. \quad (7)$$

To completely characterize the two performance metrics, it remains to model the CDFs, $F_{T_{w_i}}(t)$, of the execution times for the n sub-applications. Here, we assume that a set of m available VMs in the virtual infrastructure, denoted by the set $\mathbf{v} = [v_1, \ldots, v_m]$, are scheduled to execute the n sub-applications. Now suppose also that the ith sub-application is hosted on the jth VM, whose ECU per vCPU is denoted by λ_{v_j}. Then the execution rate, namely the reciprocal of average execution time, of the ith sub-application, λ_{w_i}, is modeled the following two realistic rules: 1) λ_{w_i} is proportional to λ_{v_j}; and 2) λ_{w_i} is inversely-proportional to the workload size of the sub-application. Hence, we can write

$$\lambda_{w_i} = c_i \frac{\lambda_{v_j}}{w_i}, \quad (8)$$

where ci, termed the VM processing coefficient, indicates the rate for serving one unit of workload (e.g., 1 MB data) in the ith sub-application by one unit of ECU of a VM. The value of c_i is determined by the details of the i th sub-application. Depending upon the specific probability distribution of the stochastic execution times of the sub-applications, $F_{T_{w_i}}(t)$ can be further characterized given the knowledge of λ_{w_i} for $i = 1, \ldots, n$.

It is worth noting that c_i may be different for $i = 1, \ldots, n$, due to the possible variety of sub-applications in the same request. Specifically, a sophisticated sub-application has the less VM processing coefficient compared to a

simple sub-application. Without loss of generality, it is convenient to normalize the workload sizes of the sub-applications in the same request so that the VM processing coefficients are same for all the sub-applications. For example, suppose that a request is consisted of two sub-applications with workload sizes w_1 and w_2, and the corresponding VM processing coefficients are $c_1 = 1$ and $c_2 = 2$, respectively. Hence, it takes half the time for a VM to execute the second sub-application compared to the first sub-application. In this case, the workload-size normalization can be done by doubling the workload size of the second sub-application. To this end, both of the two sub-applications have the same rate for serving one unit of workload by one unit of ECU. In the remainder of this paper, the workload sizes of the sub-applications in the same request are assumed to be normalized.

We would also like to clarify that $PT(\mathbf{w}; t)$ and $ET(\mathbf{w})$ also depend upon the set of m VMs $\mathbf{v}$ as well as the workload allocation algorithm that specifies how the n sub-applications are hosted on the m VMs. However, the explicit reference to this dependence can be omitted from the notation for convenience.

SECTION 4 Optimal Workload Allocation

Workload allocation (or task scheduling) is a critical issue in heterogeneous computing environments such as cloud computing. Here, allocating the workloads efficiently can clearly improve service performance. However, finding the optimal workload-allocation algorithm in a heterogeneous computing environment is in general an NP-hard problem [8]. Hence many heuristic workload-allocation algorithms, such as the minimum completion time (MCT), the min-min and the max-min, were proposed in [7] and their efficacy were investigated under various schemes [7], [8]. However, to the best of our knowledge, there is little work in the existing literature addressing the workload-allocation problem based upon the multi-tenancy principle in the cloud. In fact, the proposed algorithms in [7] behave quite differently in the multi-tenant model as illustrated later in this section.

In the following, we begin by reviewing the max-min-cloud algorithm, whose elementary version was first proposed in our prior work [6]. Here we rigorously prove the optimality of the max-min-cloud algorithm in Section 4.2. This proof illustrates that the max-min-cloud algorithm gives the best performance in the multi-tenant cloud-service environment for any arbitrary customer request, i.e., where the optimality is in the sense of maximizing the percentile of the response time of the request and minimizing the mean of the response time of the request.

4.1 The Max-Min-Cloud Workload Allocation Algorithm

In brief, the motivation for devising the max-min-cloud algorithm is based upon the concept of load balancing for the case when the execution times of the sub-applications are deterministic. Intuitively, the necessary condition for obtaining the minimum response time of a request is that the execution times of the n sub-applications should be as close to each other as possible. Specifically, the max-min-cloud algorithm follows a greedy pattern for allocating the n heterogeneous sub-applications to a fixed set of mVMs (with at least n vCPUs by default). It requires that the n sub-applications in request w are sorted from the largest to the smallest, based upon their normalized workload sizes, while the m VMs are also sorted in terms of ECU from the fastest to the slowest. Next, the sub-application with the largest workload size is allocated to the fastest available VM. If there is at least one idle vCPU in a VM, this VM is defined as an available VM. For convenience, details of the max-min-cloud algorithm are summarized in Algorithm 1.

Algorithm 1. The Max-Min-Cloud Algorithm

Initiation: A customer submits his/her request w consisting of arbitrary n sub-applications. Suppose that a set of m VMs in the cloud-service infrastructure are scheduled for serving the request, and the jth server has k_j vCPUs for $j = 1, \ldots, m$, respectively.

Sort the n sub-applications in the request $\mathbf{w}$ from the largest to the smallest based upon the normalized workload sizes of the n sub-applications. Let a vector $\mathbf{w} = [w_1, \dots, w_n]$ represent the sorted n sub-applications.

Sort the m VMs in terms of ECU from the largest to the smallest. Let a vector $\mathbf{v} = [v_1, \dots, v_m]$ represent the sorted m VMs.

for $i = 1$ **to** n **do**

for $j = 1$ **to** m **do**

if the number of sub-applications that are hosted on the j th VM is smaller than k_j, **then**

allocate the ith sub-application to the jth VM;

break;

end if

end for

end for

4.2 Optimality Proof of the Max-Min-Cloud Algorithm

Consider the case when a set of heterogeneous VMs, $\mathbf{v} = \{v_1, \dots, v_m\}$, that have at least n vCPUs in total, is scheduled to serve a customer request $\mathbf{w} = \{w_1, \dots, w_n\}$. We prove that the maximum $PT(\mathbf{w}; t)$, for any $t > 0$, and the minimum $ET(\mathbf{w})$ will be obtained by utilizing the max-min-cloud algorithm. To make the proof tractable, we assume that the stochastic execution times of sub-applications are exponentially distributed. Without loss of generality, we also assume that $c_1 = \cdots = c_n = 1$ in our proof. In this case, the formulas for $PT(\mathbf{w}; t)$ and $ET(\mathbf{w})$, from (2) and (6), become

$$PT(\mathbf{w}; t) = \prod_{i=1}^{n} \left(\left(1 - e^{-\lambda_{w_i} t}\right) \right. \quad (9)$$

and

$$ET(\mathbf{w}) = \int_0^{\infty} \left(1 - \prod_{i=1}^{n} \left(1 - e^{-\lambda_{w_i} t}\right)\right) dt, \quad (10)$$

respectively. However, we will also conduct MC simulations to show the optimality of the max-min-cloud algorithm when the execution times are assigned with other probability distributions.

Remark 1.

Equations (4) and (7) together imply the fact that the mean of the response time of a customer request, $ET(\mathbf{w})$, is minimized when the percentile of the response time of the request, $PT(\mathbf{w}; t)$, is maximized for all $t > 0$.

Theorem 1.

For any $t > 0$, the maximum percentile of the response time of a request $\mathbf{w} = \{w_1, \dots, w_n\}$ is obtained when the max-min-cloud algorithm is utilized to allocate the n sub-applications in the request to a set of VMs $\mathbf{v} = \{v_1, \dots, v_m\}$ that have exactly n vCPUs in total.

Clearly, Theorem 1 also implies that the minimum mean of the response time is obtained when the max-min-cloud algorithm is utilized as a consequence of Remark 1.

In order to prove Theorem 1, we begin by giving an example to define an operation termed app-swap, which helps us introduce Lemma 1 . Suppose that two sub-applications w_1 and w_2, with $w_1 > w_2$, are initially hosted on two VMs v_2 and v_1, with $\lambda_{v_1} > \lambda_{v_2}$, respectively. If we reallocate the sub-application w_1 to v_1, and w_2 to v_2, then this operation is defined as an app-swap between w_1 and w_2. In particular, by performing an app-swap between w_1 and w_2, the sub-application with the larger workload size (i.e., w_1) will be reallocated to the faster VM (i.e., v_1), while the sub-application with smaller workload size (i.e., w_2) will be reallocated to the slower VM (i.e., v_2).

Lemma 1.

Suppose that a customer request $\mathbf{w} = \{w_1, \ldots, w_n\}$ is executed by a set of heterogeneous VMs $\mathbf{v} = \{v_1, \ldots, v_m\}$ that have exactly n vCPUs in total. Suppose also that two of the sub-applications w_1 and w_2, with $w_1 > w_2$, are allocated to the two VMs v_2 and v_1, with $\lambda_{v_1} > \lambda_{v_2}$, respectively. Let $PT^{\text{ini}}(\mathbf{w}; t)$ denote the percentile of response time of the request $\mathbf{w}$ associated with the initial allocation pattern. Suppose that we reallocate w_1 and w_2 by performing an app-swap between them, and let $PT^{\text{swap}}(\mathbf{w}; t)$ denote the percentile of the response time of $\mathbf{w}$ after the reallocation. Then,

$$PT^{\text{swap}}(\mathbf{w}; t) > PT^{\text{ini}}(\mathbf{w}; t), \text{for all} t > 0.$$

Proof.

Note that by performing the app-swap between w_1 and w_2, the execution rates of w_1 and w_2 change while the execution rates for the rest of the $n - 2$ sub-applications in wremain the same. According to (9),

$$PT^{\text{ini}}(\mathbf{w}; t) = \left(1 - e^{\lambda_{w_1} t}\right)\left(1 - e^{\lambda_{w_2} t}\right) \prod_{i=3}^{n} \left(1 - e^{\lambda_{w_i} t}\right)$$

and

$$PT^{\text{swap}}(\mathbf{w}; t) = (1 - e^{\lambda'_{w_1} t})(1 - e^{\lambda'_{w_2} t}) \prod_{i=3}^{n} (1 - e^{\lambda_{w_i} t}).$$

Hence,

$$PT^{\text{ini}}(\mathbf{w}; t) \quad - PT^{\text{swap}}(\mathbf{w}; t) = \left(\left(1 - e^{\lambda_{v_1} t}\right)\left(1 - e^{\lambda_{v_2} t}\right)\right.$$
$$\left. - \left(1 - e^{\lambda_{v'_1} t}\right)\left(1 - e^{\lambda_{v'_2} t}\right)\right) \prod_{i=3}^{n} \left(1 - e^{\lambda_{v_i} t}\right).$$

Using Proposition 2 (which is shown in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2016.2628370), we conclude that

$$\left(1 - e^{\lambda_{w_1} t}\right)\left(1 - e^{\lambda_{w_2} t}\right) - \left(1 - e^{\lambda'_{w_1} t}\right)\left(1 - e^{\lambda'_{w_2} t}\right) < 0 \text{for all} t > 0.$$

Therefore,

$$PT^{\text{ini}}(\mathbf{w}; t) < PT^{\text{swap}}(\mathbf{w}; t), \text{for all} t > 0. \quad (11)$$

Proof of Theorem 1.

Let the n sub-applications in the request be allocated to the m VMs in an arbitrary initial allocation pattern. We will then implement a sequence of app-swaps between two of the n sub-applications based on a sorting algorithm, say bubble sort. After some finite number of app-swaps, there can be no more app-swaps within the n sub-applications that can be implemented. At this point, the n sub-applications are reallocated to the m VMs following the allocation pattern where the largest sub-applications is hosted on the fastest VMs. Note that this final allocation pattern obtained at the end of the sequence of app-swaps is precisely prescribed by the max-min-cloud algorithm.

With the knowledge of Lemma 1, it is clear that each app-swap enhances the performance for serving the request $\mathbf{w}$. Therefore, we can claim that the maximal percentile of the response time of the request $\mathbf{w}$ is achieved for all $t > 0$ by utilizing the max-min-cloud algorithm.

Next, we extend Theorem 1 by considering the general case when the set of m VMs has more than n vCPUs in total.

Theorem 2.

Suppose that a request with n sub-applications is to be executed by a set of m VMs that have more than n vCPUs in total. Then for any $t > 0$, the maximum percentile of the response time of the request $\mathbf{w}$ is obtained when the max-min-cloud algorithm is utilized.

Proof.

First, consider the case when the n sub-applications in the request are allocated to the set of m' fastest VMs, denoted by $\mathbf{v}_{\mathrm{mmc}}$, of the m VMs. Without loss of generality, we can assume that the set of VMs in $\mathbf{v}_{\mathrm{mmc}}$ has exactly n vCPUs. (For the case when there are more than n vCPUs in $\mathbf{v}_{\mathrm{mmc}}$, the vCPUs that has the smallest ECU per vCPU can be ignored, since they will not be executing any sub-applications.) By Theorem 1, the best performance for serving the n sub-applications by the set of VMs $\mathbf{v}_{\mathrm{mmc}}$ is obtained by applying the max-min-cloud algorithm. In this case, we use the term $PT_{\mathrm{mmc}}(\mathbf{w}; t)$ to denote the percentile of the response time of the request $\mathbf{w}$, and the execution rates of the n sub-applications are denoted by $\lambda_{w_1}^{\mathrm{mmc}}, \ldots, \lambda_{w_n}^{\mathrm{mmc}}$.

Next, consider the other case when a different collection of m'' VMs, denoted by $\mathbf{v}_{\mathrm{arb}}$, are selected to serve the n sub-applications, i.e., $\mathbf{v}_{\mathrm{arb}} \neq \mathbf{v}_{\mathrm{mmc}}$. Without loss of generality, we can also assume that the set of VMs in $\mathbf{v}_{\mathrm{arb}}$ has exactly n vCPUs. Again by Theorem 1, the best performance is obtained by applying the max-min-cloud algorithm. Let $PT_{\mathrm{arb}}(\mathbf{w}; t)$ denote the percentile of the response time of the request $\mathbf{w}$ this scenario, and the execution rates of the n sub-applications are denoted by $\lambda_{w_1}^{\mathrm{arb}}, \ldots, \lambda_{w_n}^{\mathrm{arb}}$.

Note that the VMs in $\mathbf{v}_{\mathrm{mmc}}$ are the selection of the fastest VMs. Now when the max-min-cloud algorithm is utilized, the vCPU of the VM in $\mathbf{v}_{\mathrm{mmc}}$ is faster than the vCPU of the VM in $\mathbf{v}_{\mathrm{arb}}$ pair-wisely for executing the same workload, i.e.,

$$\lambda_{w_i}^{\mathrm{mmc}} \geq \lambda_{w_i}^{\mathrm{arb}} \text{for} i = 1, \ldots, n.$$

Hence, we have

$$PT_{\mathrm{mmc}}(\mathbf{w}; t) > PT_{\mathrm{arb}}(\mathbf{w}; t) \text{for all} t > 0$$

as a result of (4). The proof of Theorem 2 is completed by noting that the selection of the collection of m'' VMs is arbitrary.

Theorem 2 also implies that the minimum mean of the response time is obtained when the max-min-cloud algorithm is utilized due to Remark 1. To this end, we have rigorously proven the optimality of the max-min-cloud algorithm in the sense of maximizing the percentile of the response time for any $t > 0$ as well as minimizing the mean of the response time for serving an arbitrary customer request when a fixed set of VMs has been scheduled to serve this request.

SECTION 5 Experimental and Simulation Results

In this section, we conduct experiments along with MC simulations to validate the efficacy of the max-min-cloud algorithm under various scenarios. To do so, we compare its performance to that of a widely-adopted workload-allocation algorithm termed the MCT-cloud algorithm, which extends the minimum completion time algorithm introduced in [8]. The MCT-cloud algorithm allocates the sub-applications in a request $\mathbf{w}$, in arbitrary order, to the fastest available VM in a scheduled set of VMs $\mathbf{v}$. The MCT-cloud algorithm performs well for cloud services whose virtual infrastructure consists of near-homogeneous VMs [5].

Consider the case for which a customer submits a request whose purpose is to collect and process a total of 450 MB of data from different resources. By default, such request will be divided into nine sub-applications. We assume that four VMs, including one c4.xlarge instance, one c3.large instance, one c1.medium instance and one m1.medium instance, are scheduled to serve the request. Furthermore, six workload patterns of the sub-applications, which are listed in Table 2, of such request are examined. The execution times of the sub-applications in the request are first assumed to be exponentially distributed.

TABLE 2 The Mean of the Response Time of Request w with Different Workload Patterns

Request pattern	workload sizes of sub-apps (MB)		mean of response time (in seconds)	
	$[w_1; \ldots; w_9]$	theo. prediction	sim. max-min-cloud	sim. MCT-cloud
1	[370; 10; 10; 10; 10; 10; 10; 10; 10]	26,522	26,496	37,939
2	[90; 90; 90; 90; 50; 10; 10; 10; 10]	13,794	13,785	19,332
3	[50; 50; 50; 50; 50; 50; 50; 50; 50]	13,546	13,547	13,545
4	[½90; 50; 50; 50; 50; 50; 50; 50; 10]	12,953	12,948	15,367
5	[90; 80; 70; 60; 50; 40; 30; 20; 10]	12,412	12,403	16,946
6	[60; 60; 60; 60; 52; 52; 38; 38; 30]	12,104	12,095	14,487

request pattern	workload sizes of sub-apps (MB) $[w_1, \ldots, w_9]$	mean of response time (in seconds) theo. prediction	sim. max-min-cloud	sim. MCT-cloud
1	[370, 10, 10, 10, 10, 10, 10, 10, 10]	26,522	26,496	37,939
2	[90, 90, 90, 90, 50, 10, 10, 10, 10]	13,794	13,785	19,332
3	[50, 50, 50, 50, 50, 50, 50, 50, 50]	13,546	13,547	13,545
4	[90, 50, 50, 50, 50, 50, 50, 50, 10]	12,953	12,948	15,367
5	[90, 80, 70, 60, 50, 40, 30, 20, 10]	12,412	12,403	16,946
6	[60, 60, 60, 60, 52, 52, 38, 38, 30]	12,104	12,095	14,487

To specify the calculation of the VM processing coefficient c_i of the ith sub-application in the proposed multi-tenant model, we take the word-count and the word-median programs given by Hadoop 2.7.2 as two examples of the sub-applications. In particular, our experiment is running on a Intel Core 2 Q9550 CPU desktop, whose computing power is approximately 80 percent as compared to the m4.xlarge VM provided by Amazon EC2 (i.e., the ECU of the desktop equals to 5.2). We conduct a series of experiments processing on a group of text files with different sizes. In Fig. 2, we show the experimental results of the execution times of the word-count and the word-median programs as a function of the workload size of the text files. We fit the two experimental results with two linear functions in order to calculate the corresponding VM processing coefficients of the word-count program, say c_1, and the word-median program, say c_2, respectively. After offsetting the setup overhead of the programs, we determine that $c_1 \approx 0.64$ and $c_2 \approx 1.786$.

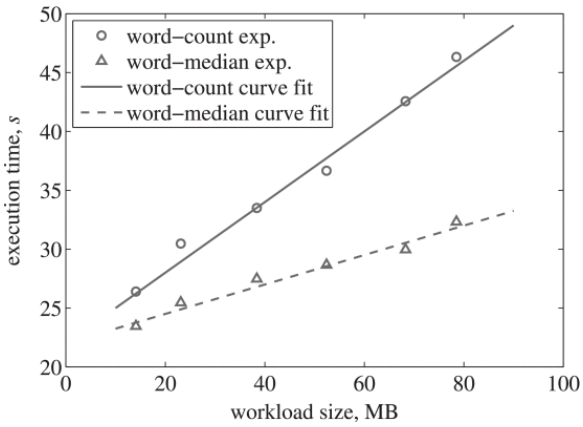

Fig. 2. The execution times of the word-count and the word-median programs as a function of workload size.

In reality, the value of c_i is determined by the specific cloud service that is provided by a business. Typically, the business has the prior knowledge of all the cloud services that are provided to the customers (so as to formulate the pricing strategy). Hence, a lookup table can be created by the business to store the values of the VM processing coefficients for all kinds of the sub-applications. When a customer submits his/her request, the VM processing coefficient c_i of each sub-application for $i = 1, \dots, n$ in the request can be extracted from the lookup table.

With the knowledge of the VM processing coefficient, we proceed to conduct MC simulations to show the mean of response times of the six requests for the cases when the max-min-cloud algorithm and the MCT-cloud algorithm are utilized. For convenience, the workload sizes of the nine sub-applications in the simulations have been normalized. Furthermore, we assume that $c_1 = \cdots = c_9 = 0.0035$, which implies that the sub-applications considered in the simulations are more sophisticated programs than the word-count or the word-median programs. In Table 2, the columns labeled as "theo." and "sim." present the results obtained from numerically evaluating the analytical characterization of $ET(\mathbf{w})$ in (6) and after averaging 10,000 realization of independent experiments, respectively. It is noted that the execution rates of the sub-applications cannot be determined when the MCT-cloud algorithm is utilized. Hence, there is no theoretical prediction of $ET(\mathbf{w})$ for the MCT-cloud algorithm.

It can be observed from Table 2 that for any of the six patterns, the mean of the response time corresponding the max-min-cloud algorithm are less than or equal to that for the MCT-cloud algorithm. Note that the theoretical prediction of $ET(\mathbf{w})$ matches the MC simulation results. It is also important to note that different patterns of the request $\mathbf{w}$ may lead to variation in the values of $ET(\mathbf{w})$, even though these patterns have the same total workload size. For example, $ET(\mathbf{w})$ of "pattern 1" is 26,522 seconds, which is about two times longer than that of "pattern 6" (i.e., 12,104 seconds). This is because the workloads in "pattern 1" are highly unbalanced as compared to the distribution of the ECU per vCPU of the four scheduled VMs that serve the request. Meanwhile, the workloads in "pattern 6" have the most similar distribution of ECU per vCPU of the four VMs among the six patterns of the request under study.

Next, in Fig. 3, we show the percentile of the response time of the request $\mathbf{w}$ by applying the max-min-cloud algorithm and the MCT-cloud algorithm (for the six patterns described earlier). The solid curves representing the theoretical predictions are obtained by evaluating (2) numerically when the max-min-cloud algorithm is utilized. The MC simulation results marked by squares and crosses are presented respectively for the max-min-cloud algorithm, and the MCT-cloud algorithm. Each point in the MC simulations are averaged using 10,000 independent experiments. As expected, the max-min-cloud algorithm leads to larger values of $PT(\mathbf{w}; t)$ for all $t > 0$. Here too, our theoretical predictions agree well with the MC results when the max-min-cloud algorithm is utilized. It is interesting to observe in Fig. 3 c that all the curves for "pattern 3" coincide. This is

because all the nine sub-applications in the request have the same workload size. As such, there is no difference for allocating sub-applications in the request when the two algorithms are utilized.

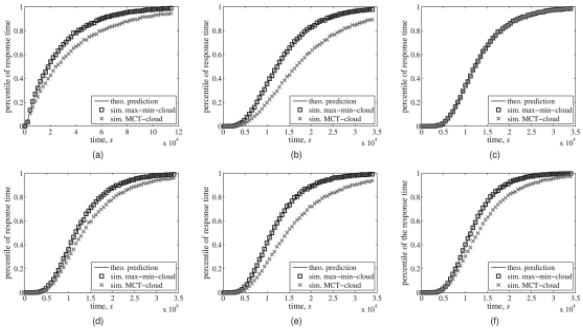

Fig. 3. Percentile of the response time of request (higher value implies a better performance).

To better illustrate the impact of the workload pattern of the request on $PT(\mathbf{w}; t)$ we also show results obtained by theoretical predictions for the six workload patterns of the request together in Fig. 4.

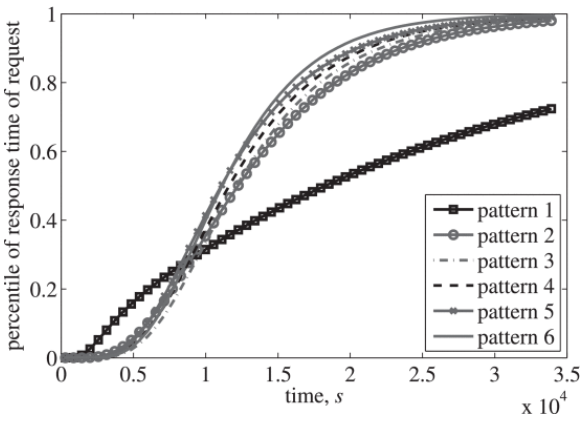

Fig. 4. Percentile of the response times for the six patterns.

Next, we illustrate the optimality of the max-min-cloud algorithm when the execution times of the sub-applications are not exponentially-distributed (but with the same execution rates as the times are exponentially-distributed). Specifically, we consider two cases when the execution times are (a) truncated normally-distributed whose variance is one third of its mean (as similarly considered in [5]) and (b) Erlang-distributed. The corresponding percentile of the response time of the "pattern 4" under these two distributions are shown in Figs. 5a and 5 b, respectively.

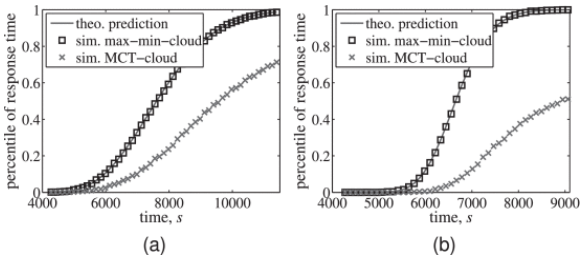

Fig. 5. Percentile of the response times, where the execution times of the sub-applications in the request are (a) truncated normally-distributed and (b) Erlang-distributed.

It can be observed by comparing the results from Figs. 5 and 3d to find that the superiority of the max-min-cloud algorithm over the MCT-cloud algorithm is more profound versus the case when the execution times of the sub-applications are not exponentially-distributed. For example, in Figs. 5a and 5b when $t = 8{,}000$ seconds, the

max-min-cloud algorithm can lead to approximately 0.5 and 0.6 higher probability for completing the request than that for the MCT-cloud algorithm. Meanwhile in Fig. 3d such probability is less than 0.1.

SECTION 6 Efficient Resource Provisioning

When a customer submits his/her request, the amount of computing resources (namely a set of VMs) for serving this request needs to be determined and scheduled before the execution of the request. To schedule an appropriate set of VMs for serving customer requests is a challenging problem, which is typically termed as resource provisioning in the cloud [20]. Here the challenge in devising an efficient provisioning strategy is twofold. On one hand, the business must provide a service that would not miss the deadlines for customers (i.e., avoid under-provisioning). On the other hand, the business must try to maximize its profit or keep the cost at a minimum for operating the cloud service (i.e., avoid over-provisioning).

Therefore, our goal is to find an efficient provisioning strategy that brings as much profit or revenue to the business as possible. To this end, in this section we propose the minimum-provisioning-cost resource-provisioning strategy for serving the incoming customer requests and compare its performance with other resource-provisioning strategies via MC simulations.

6.1 The MPC Resource-Provisioning Strategy

In brief, the idea is to minimize provisioning costs while guaranteeing the performance for serving customer requests. Specifically, suppose that there are M types of VMs that can be chosen by the business to serve a customer request $\mathbf{w} = \{w_1, \ldots, w_n\}$. The jth type VM has k_j vCPUs and its usage price per unit of time is denoted as price$_j$. The MPC strategy needs to determine the appropriate set of VMs, denoted by $\mathbf{s} = \{s_1, \ldots, s_j, \ldots, s_M\}$, to be scheduled to serve the request $\mathbf{w}$. Here, sjrepresents the number of jth type VM in s for $j = 1, \ldots, M$. We model the provisioning cost for serving the request $\mathbf{w}$ as the product of the total price paid for using the set of VMs s and the mean of the response time of the request $\mathbf{w}$. Mathematically, we define the provisioning cost as

$$C(\mathbf{s}) = ET(\mathbf{w}) \sum_{j=1}^{M} \text{price}_j \text{ . (12)}$$

The set of VMs determined by the MPC strategy is the set $\mathbf{s}$ that minimizes $C(\mathbf{s})$ subject to the following three constraints:

(a) $\sum_{i=1}^{M} k_j s_j \geq n$, namely the total number of vCPUs of the VMs in $\mathbf{s}$ should be greater than or equal to the number of sub-applications in the request $\mathbf{w}$.

(b) $\sum_{i=1}^{M} k_j s_j \leq n + \max_{j=1,\ldots,M}(k_j)$.In light of the optimality proof of the max-min-cloud algorithm given in Section 4.2, it is clear that the redundant and idle VMs/vCPUs that do not run any workload do not enhance the service performance of the customer request.

(c) $PT(\mathbf{w}; t_D) \geq \alpha$. Given the desired response time t_D, the probability of completing the request $\mathbf{w}$ before t_D should be greater than or equal to a certain confidence factor α. Here $PT(\mathbf{w}; t_D)$ is computed by utilizing the max-min-cloud algorithm.

To solve this optimization problem required by the MPC strategy, we have to exhaustively check all the combinations of possible values of n_1, n_2, ..., and n_M that satisfy the three constraints. The solution is one of the elements in the search space that leads to the minimum $C(\mathbf{s})$. However, it is important to note that constraints

(a) and (b) together reduce the size of the search space. Hence, the computational complexity for finding the solution is much lower than a typical combinatorial optimization problem.

Next, we investigate the efficacy of the MPC strategy under two practical scenarios when the arrival of customer requests is unpredictable and predictable. For the unpredictable case, we assume that on-demand VMs are utilized to serve the customer requests and the virtual cloud-service infrastructure is elastic. For the predictable case, on the other hand, reserved VMs are utilized and the virtual cloud-service infrastructure is fixed.

6.2 Elastic Virtual Cloud-Service Infrastructure with On-Demand VMs

Consider a scenario where the arrival of customer requests is highly dynamic and unpredictable. In this case, on-demand VMs are typically used by the business for serving these requests. The business can take full advantage of the elasticity of the cloud, i.e., the VMs in the virtual infrastructure can be unilaterally expanded and reduced depending upon the demand [15], [36] . The cost for operating the cloud service on an elastic virtual infrastructure is pay-per-use, i.e., the longer a VM is used the more the business pays.

Next, we present an experimental study to illustrate the efficacy of the MPC strategy. The setting of the experiment is similar to what we have in Section 5. In particular, we assume that four types of VMs, including c4.xlarge, c3.large, c1.medium and m1.medium, can be chosen to serve the customer requests. The costs for utilizing these four VMs (per hour) are listed in Table 1. For one realization of the experiment, we assume that 100 customers submit their requests to the virtual infrastructure. Here, each request is randomly chosen from eight different patterns of a request with 450 MB workload size as listed in Table 3.

TABLE 3 Eight Patterns of a Request with 450 MB Workload Size

pattern #	workload sizes of sub-apps (MB)
1	[80; 70; 60; 50; 40; 40; 35; 30; 25; 20]
3	[80; 80; 45; 45; 45; 45; 45; 45; 10; 10]
3	[90; 90; 50; 50; 50; 50; 50; 10; 10]
4	[90; 80; 70; 60; 50; 40; 30; 20; 10]
5	[90; 80; 70; 65; 55; 40; 30; 20]
6	[90; 90; 60; 60; 55; 55; 20; 20]
7	[90; 80; 70; 60; 60; 50; 40]
8	[90; 85; 65; 65; 65; 50; 30]

pattern #	workload sizes of sub-apps (MB)
1	[80, 70, 60, 50, 40, 40, 35, 30, 25, 20]
2	[80, 80, 45, 45, 45, 45, 45, 45, 10, 10]
3	[90, 90, 50, 50, 50, 50, 50, 10, 10]
4	[90, 80, 70, 60, 50, 40, 30, 20, 10]
5	[90, 80, 70, 65, 55, 40, 30, 20]
6	[90, 90, 60, 60, 55, 55, 20, 20]
7	[90, 80, 70, 60, 60, 50, 40]
8	[90, 85, 65, 65, 65, 50, 30]

In this scenario, the metric to be used for evaluating the efficacy of a resource-provisioning strategy is the average profit for serving a certain number of customers. We further assume that the business can earn (1) full revenue from a customer if his/her request is completed within the desired response time; (2) half revenue if the completion time is larger than the desired response time but smaller than or equal to two times of the desired response time; (3) nothing if the completion time is larger than two times of the desired response time. Hence, the revenue for serving a customer (same for the eight patterns) is written as

$$\text{rev}(\mathbf{w}) = \begin{cases} r|\mathbf{w}| & \text{if} T_{\mathbf{w}} \le t_D \\ r|\mathbf{w}|/2 & \text{if} t_D < T_{\mathbf{w}} \le 2t_D \\ 0 & \text{if} T_{\mathbf{w}} > 2t_D. \end{cases} \quad (13)$$

Here, r is the rated revenue for finishing a unit of workload (e.g., 1 MB data in our experiment) within the desired response time. The term $|\mathbf{w}|$ denotes the total workload size of the request $\mathbf{w}$, i.e., $|\mathbf{w}| = \sum_{i=1}^{n} w_i$. Hence the profit for serving a customer is

$$\text{profit}(\mathbf{w}) = \text{rev}(\mathbf{w}) - C(\mathbf{s}). \quad (14)$$

For comparison, we apply two other commonly-used and straightforward resource-provisioning strategies as benchmarks. One is the greedy-provisioning strategy, where the minimum number of VMs with the highest ECU that have at least n vCPUs will be scheduled to serve a request. The other is the random-provisioning strategy, where VMs that have at least n vCPUs will be randomly chosen and scheduled to serve a request.

The average profit for serving 100 customer requests as a function of the desired response time are shown in Fig. 6. The rated revenue per unit of workload is assumed $r = 0.005$ (namely 0.5 cent per MB). The three curves represent the results obtained from utilizing the GP strategy, the RP algorithm and the MPC strategy, respectively. Each point in the three curves is averaged by 1,000 realizations. As expected, the average profit increases as tD becomes larger. It is clear that the MPC strategy outperforms the other two strategies in terms of generating more profit for the business under any t_D. It is also interesting to find that the GP strategy outperforms the RP strategy when $t_D < 19{,}000$ seconds but falls behind when $t_D < 19{,}000$ seconds. The reason here is that the RP strategy is not able to guarantee the service performance. As a result, a substantial proportion of the requests may be completed beyond t_D when the desired response time is relatively short, and this leads to a large loss in revenue. Meanwhile, when the desired response time is relatively long, the main factor to determine the revenue is the provisioning cost. Hence the performance of the GP strategy becomes worse as t_D increases. In summary, the MPC strategy overcomes the inherent shortcomings of the other two resource-provisioning strategies and emerges the best performance in terms of generating the most profit.

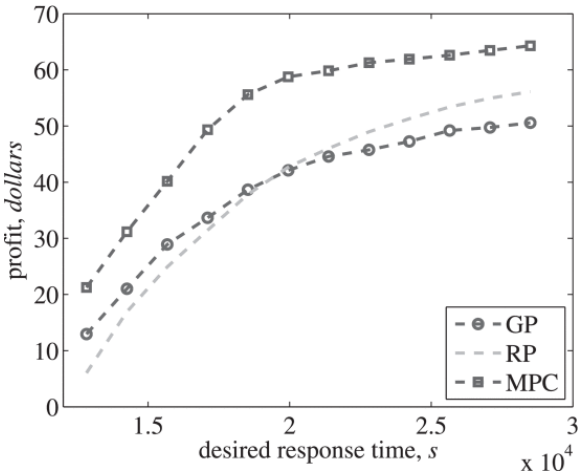

Fig. 6. The average profit for serving 100 customer requests as a function of the desired response time.

The results shown in Fig. 6 are highly helpful for businesses negotiating with their customers on some critical attributes of the SLAs, e.g., such as the charge plan and the desired response time for serving the requests. For convenience, in Fig. 7 we show a flowchart for efficiently serving dynamically arriving customer requests in an elastic virtual cloud-service infrastructure with on-demand VMs.

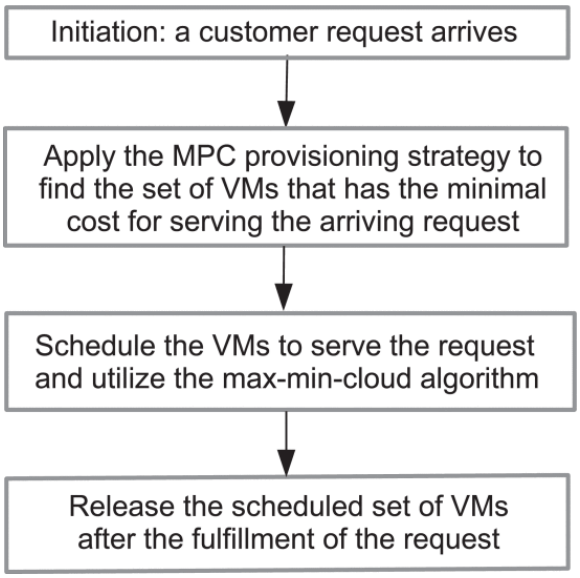

Fig. 7. Flowchart for efficiently serving dynamically arriving customer requests when the virtual cloud-service infrastructure is elastic by utilizing on-demand VMs.

6.3 Fixed Virtual Cloud-Service Infrastructure with Reserved VMs

Next, we consider another practical scenario when all the VMs in the virtual cloud-service infrastructure are reserved, and no on-demand VMs will be added to the virtual infrastructure. Such cloud service is typically used when the response time is not the first concern by the customers, and the arrival of the customers follows a certain pattern. The advantages of utilizing reserved VMs compared to the on-demand VMs include less cost (e.g., a significant discount up to 75 percent in Amazon EC2 [23]) for obtaining the same amount of computing power per unit time and the ease of maintenance and management of the VMs in the virtual infrastructure.

In this case, the virtual cloud-service infrastructure and the cost for operating such cloud service is fixed. When a customer submits his/her request, the business can only utilize the available VMs in the fixed virtual infrastructure to serve the request. The business also has the right to reject a customer's request [37] if there is insufficient computing resource (i.e., number of available VMs). We also assume that the business can earn full revenue from a customer as long as his/her request is accepted.

Note that the MPC strategy proposed in Section 6.1 needs to be slightly modified by adding one more constraint in order to suit the fixed virtual infrastructure:

(d) $s_j \leq N_j, j = 1, \ldots, M$, that is, the number of VMs in $\mathbf{s}$ is limited by the number of available VMs, denoted by N_j for $j = 1, \ldots, M$, in the virtual cloud-service infrastructure.

It is possible that there may be no solution to this optimization problem when constraint (d) is added. This implies that there is insufficient number of available VMs to serve the request and such request will be rejected.

Clearly when fixed virtual infrastructure is utilized, an efficient resource-provisioning strategy aims to serve as many customers as possible. Hence, we use the rejection rate of customers to be served as a metric to evaluate the performance of a resource-provisioning strategy. Next, we present an experimental study to illustrate the efficacy of the MPC strategy in the case only the reversed VMs are utilized to serve the dynamically arriving customers.

Suppose that the fixed virtual cloud-service infrastructure consisted of 150 reserved VMs, comprising 15 c4.xlarge VMs, 30 c3.large VMs, 45 c1.medium VMs and 60 m1.medium VMs. Customers submit their requests

sequentially following a Poisson process with arrival rate λc. The rejection rate of the ith customer (averaged over 10,000 realizations) is shown in Fig. 8 for $\lambda_c = 8.28$ hour^{-1}. The three curves represent the results obtained by utilizing the GP strategy, the RP strategy and the MPC strategy, respectively. It can be observed that the rejection rate starts increasing after the service of about 30 requests and then becomes stable after serving 50 to 60 requests. The MPC strategy leads to the minimum steady-state rejection rate compared to the other two strategies.

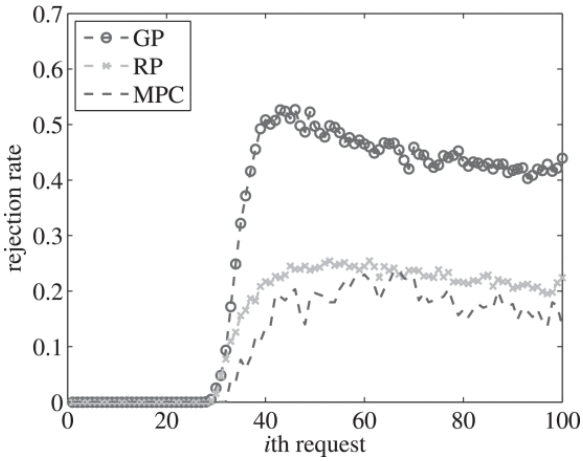

Fig. 8. The rejection rate for serving 100 customers as a function of the index of the customers.

In Fig. 9, we further show the steady-state rejection rate for customer requests as a function of the arrival rate, λ_c. It is noted that the MPC strategy again outperforms the other two strategies in terms of reducing the steady-state rejection rate for any λ_c. The GP strategy has the worst performance in this scenario. The difference in the performance of the three strategies is subtle only when $\lambda_c < 5$ hour^{-1} and $\lambda_c > 50$ hour^{-1}. This is simply because that either none or all of the customer requests will be rejected if the average time between the arrivals of two requests (namely $1/\lambda_c$) is relatively too long or too short compared to the average response time of a request, respectively.

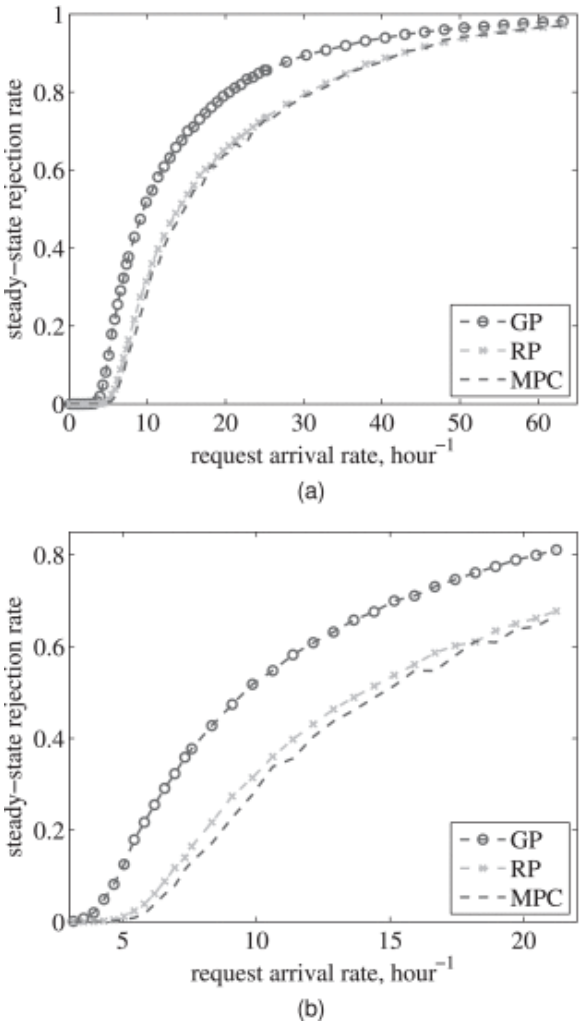

Fig. 9. (a) The steady-state rejection rate (smaller is better) under three resource-provisioning strategies as a function of arrival rate of the customer requests, and (b) the zoomed-in version of left half of Fig. 9a.

As seen in Fig. 9, the GP strategy in general yields the worst performance by having the largest steady-state rejection rate among the three strategies. It is also noted that the MPC strategy outperforms the RP strategy about 2-5 percent in the sense of reducing the steady-state rejection rate when $5 < \lambda_c < 20$ hour^{-1}. The superiority of the MPC strategy is not as prominent as in the previous scenario when the virtual cloud-service infrastructure is elastic and on-demand VMs are utilized. However, the MPC strategy still gives tremendous benefits for this particular scenario, since the cloud service based upon reserved VMs is typically running for long terms, i.e., several years. Reducing the steady-state rejection rate by 5 percent directly implies an increase in the generated revenue by 5 percent. Overall, investigation of the steady-state rejection rate in the cloud (as shown in Fig. 9) can help businesses better understand how much VMs they need to purchase in order to maintain a high quality of service and satisfy their customers, which implies keeping the rejection rate to a minimum. For convenience, we show in Fig. 10a flowchart for efficiently serving dynamically arriving customer requests in a fixed virtual cloud-service infrastructure with reserved VMs.

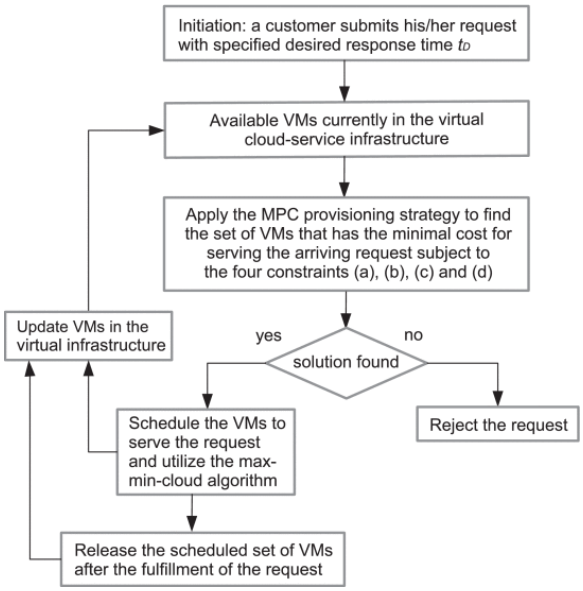

Fig. 10. Flowchart for efficiently serving dynamically arriving customer requests when the virtual cloud-service infrastructure is fixed by utilizing the reserved VMs.

SECTION 7 Conclusions and Future Extensions

In this paper we have proposed a novel probabilistic framework to model the service of customers in the cloud. The model considers essential features and concerns in modern cloud services including the multi-tenancy characteristic, the heterogeneity of VMs in the virtual infrastructure and the stochastic response times for serving a request with general probability distribution. To this end, the percentile and mean of the stochastic response times of customer requests have been analytically characterized in closed form. These two quantities are widely-used metrics for evaluating the performance of cloud services in the research community as well as cloud SLAs.

Based upon the proposed cloud-service framework, we have devised a max-min-cloud algorithm for allocating the sub-applications (i.e., workloads) in an arriving request to VMs. We have rigorously proved the optimality of the max-min-cloud algorithm and further conducted extensive experiments and MC simulations to demonstrate its optimality under various scenarios. As a byproduct of the max-min-cloud algorithm, we also devised an efficient resource-provisioning strategy, termed the MPC strategy, for determining the appropriate amount of computing resources in the cloud required to serve dynamically arriving customer requests. Our practical case study shows that utilizing the MPC strategy can yield a 10-40 percent increase in profit to businesses compared to other resource-provisioning strategies when the cloud service is based upon on-demand VMs. On the other hand, when the cloud service is based upon reserved VMs with fixed cost, we have found that the MPC strategy outperforms the other strategies by accepting 2-20 percent more requests when customers are submitting their requests with a normal pace.

The results presented in this paper are not limited to the case when cloud-service performance is dependent merely on the ECU (i.e., vCPU speed) of the VMs. Namely, our framework can also be extended to scenarios when the cloud-service performance is determined and affected by other factors, such as network bandwidth and energy consumption. For example, consider the data transmission service in the cloud where data throughput is the key performance metric. In this case, the throughput is mainly dependent upon the network bandwidth of the VMs that are used to process and transfer data. To improve the utilization of the network bandwidth, the network bandwidth in a VM is typically shared by several workloads/tenants. Hence, we can replace ECU with network bandwidth in the proposed multi-tenant framework to characterize the cloud-service

performance. Here the max-min-cloud algorithm is also useful for optimizing the utilization of network bandwidth in the VMs. In cases when energy consumption of VMs dominates the total provisioning cost of the VMs, the MPC strategy can also be extended to include energy consumption of the VMs in the cost model.

Future extensions may include addressing the case when sub-applications are heavily consuming memory or storage space. In such a case, the memory/space capacity of VMs has to be taken into account and the max-min-cloud algorithm may become suboptimal. It is also of interest to examine the performance of the max-min-cloud algorithm by considering the performance interference phenomenon in VMs when a large number of sub-applications are executed in the same VM.

Acknowledgments

This work was made possible by the NPRP 5-137-2-045 grant from the Qatar National Research Fund (a member of the Qatar Foundation). The statements made herein are solely the responsibility of the authors. The authors are very grateful for this support.

References

1. K. Xiong, H. Perros, "Service performance and analysis in cloud computing", Proc. IEEE World Conf. Serv., pp. 693-700, 2009.
2. B. Yang, F. Tan, Y. Dai, S. Guo, "Performance evaluation of cloud service considering fault recovery", Proc. 1st Int. Conf. Cloud Comput., pp. 571-576, 2009.
3. H. Khazaei, J. Mišić, V. B. Mišić, "Performance analysis of cloud computing centers using M/G/m/m+r queueing systems", IEEE Trans. Parallel Distrib. Syst., vol. 23, no. 5, pp. 936-943, May 2012.
4. B. Yang, F. Tan, Y.-S. Dai, "Performance evaluation of cloud service considering fault recovery", J. Supercomputing, vol. 65, pp. 426-444, 2013.
5. S. Yeo, H. Lee, "Using mathematical modeling in provisioning a heterogeneous cloud computing environment", Computer, vol. 44, no. 8, pp. 55-62, 2011.
6. Z. Wang, M. M. Hayat, N. Ghani, K. B. Shaban, "A probabilistic multi-tenant model for virtual machine allocation in cloud systems", Proc. 3rd IEEE Int. Conf. Cloud Netw., pp. 339-343, 2014.
7. T. Braun et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", J. Parallel Distrib. Comput., vol. 61, no. 6, pp. 810-837, 2001.
8. G. Ritchie, J. Levine, "A fast effective local search for scheduling independent jobs in heterogeneous computing environments", 2003.
9. S. T. Maguluri, R. Srikant, L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters", Proc. IEEE INFOCOM, pp. 702-710, 2012.
10. G. Juve, E. Deelman, "Resource provisioning options for large-scale scientific workflows", Proc. IEEE 4th Int. Conf. e-Sci., pp. 608-613, 2008.
11. J. Yang, J. Qiu, Y. Li, "A profile-based approach to just-in-time scalability for cloud applications", Proc. IEEE Int. Conf. Cloud Comput., pp. 9-16, 2009.
12. S. Chaisiri, B.-S. Lee, D. Niyato, "Optimal virtual machine placement across multiple cloud providers", Proc. IEEE Asia-Pacific Serv. Comput. Conf., pp. 103-110, 2009.
13. S. Chaisiri, B.-S. Lee, D. Niyato, "Optimization of resource provisioning cost in cloud computing", IEEE Trans. Serv. Comput., vol. 5, no. 2, pp. 164-177, Apr. 2012.
14. I. Foster et al., "Cloud computing and grid computing 360-degree compared", Proc. Grid Comput. Environ. Workshop, pp. 1-10, 2008.
15. T. Sridhar, "Cloud computing—A primer", Internet Protocol J., vol. 12, no. 3, pp. 2-19, 2009.
16. Q. Zhang, L. Cheng, R. Boutaba, "Cloud computing: State-of-the-art and research challenges", J. Internet Serv. Appl., vol. 1, no. 1, pp. 7-18, 2010.

17. B. P. Rimal, E. Choi, I. Lumb, "A taxonomy and survey of cloud computing systems", Proc. 5th Int. Joint Conf. INC IMS IDC, pp. 44-51, 2009.
18. M. Armbrust et al., "A view of cloud computing", Commun. ACM, vol. 53, no. 4, pp. 50-58, 2010.
19. L. Wu, S. K. Garg, R. Buyya, "SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments", Proc. 11th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput., pp. 195-204, 2011.
20. R. Buyya et al., "Cloud computing and emerging it platforms: Vision hype and reality for delivering computing as the 5th utility", Future Generation Comput. Syst., vol. 25, no. 6, pp. 599-616, 2009.
21. P. Patel, A. H. Ranabahu, A. P. Sheth, "Service level agreement in cloud computing", OOPSLA Cloud Comput. Workshops Orlando FL USA, 2009.
22. A. Iosup et al., "Performance analysis of cloud computing services for many-tasks scientific computing", IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 6, pp. 931-945, Jun. 2011.
23. [online] Available: http://aws.amazon.com/ec2/.
24. J. Dean, S. Ghemawat, "MapReduce: Simplified data processing on large clusters", Commun. ACM, vol. 51, no. 1, pp. 107-113, 2008.
25. [online] Available: http://hadoop.apache.org/.
26. X. Qiu et al., "Cloud technologies for bioinformatics applications", Proc. 2nd Workshop Many-Task Comput. Grids Supercomputers, 2009.
27. M. C. Schatz, B. Langmead, S. L. Salzberg, "Cloud computing and the DNA data race", Nature Biotechnology, vol. 28, no. 7, pp. 691-693, 2010.
28. B. Langmead, K. D. Hansen, J. T. Leek, "Cloud-scale RNA-sequencing differential expression analysis with Myrna", Genome Biol., vol. 11, no. 8, 2010.
29. W. Iqbal, M. N. Dailey, D. Carrera, P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud", Future Generation Comput. Syst., vol. 27, no. 6, pp. 871-879, 2011.
30. Y. Hu, J. Wong, G. Iszlai, M. Litoiu, "Resource provisioning for cloud computing", Proc. Conf. Center Adv. Stud. Collaborative Res., pp. 101-111, 2009.
31. W. Voorsluys, J. Broberg, S. Venugopal, R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation", Proc. 1st Int. Conf. Cloud Comput., pp. 254-265, 2009.
32. K. Boloor, R. Chirkova, Y. Viniotis, T. Salo, "Dynamic request allocation and scheduling for context aware applications subject to a percentile response time SLA in a distributed cloud", Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci., pp. 464-472, 2010.
33. W. Iqbal, M. Dailey, D. Carrera, "SLA-driven adaptive resource management for web applications on a heterogeneous compute cloud", Proc. 1st Int. Conf. Cloud Comput., pp. 243-253, 2009.
34. H. N. Van, F. D. Tran, J.-M. Menaud, "SLA-aware virtual resource management for cloud infrastructures", Proc. 9th IEEE Int. Conf. Comput. Inf. Technol., vol. 1, pp. 357-362, 2009.
35. K. H. Kim, A. Beloglazov, R. Buyya, "Power-aware provisioning of cloud resources for real-time services", Proc. 7th Int. Workshop Middleware Grids Clouds e-Sci., 2009.
36. P. Mell, T. Grance, "The NIST definition of cloud computing", Sep. 2011.
37. X. H. Guo et al., "SPIN: Service performance isolation infrastructure in multi-tenancy environment" in Service-Oriented Computing—ICSOC, Berlin, Germany:Springer, 2008.