Towards Operational Cost Minimization in Hybrid Clouds for Dynamic Resource Provisioning with Delay-Aware Optimization

Song Li, Yangfan Zhou, *Member, IEEE*, Lei Jiao, Xinya Yan, Xin Wang, *Member, IEEE*, and Michael Rung-Tsong Lyu, *Fellow, IEEE*

Abstract—Recently, hybrid cloud computing paradigm has be widely advocated as a promising solution for Software-as-a-Service (SaaS) providers to effectively handle the dynamic user requests. With such a paradigm, the SaaS providers can extend their local services into the public clouds seamlessly so that the dynamic user request workload to a SaaS can be elegantly processed with both the local servers and the rented computing capacity in the public cloud. However, although it is suggested that a hybrid cloud may save cost compared with building a powerful private cloud, considerable renting cost and communication cost are still introduced in such a paradigm. How to optimize such operational cost becomes one major concern for the SaaS providers to adopt the hybrid cloud computing paradigm. However, this critical problem remains unanswered in the current state of the art. In this paper, we focus on optimizing the operational cost for the hybrid cloud paradigm by theoretically analyzing the problem with a Lyapunov optimization framework. This allows us to design an online dynamic provision algorithm. In this way, our approach can address the real-world challenges where no *a priori* information of public cloud renting prices is available and the future probability distribution of user requests is unknown. We then conduct extensive experimental study based on a set of real-world data, and the results confirm that our algorithm can work effectively in reducing the operational cost.

Index Terms—Cloud computing, hybrid cloud, software-as-a-service

1 Introduction

Cloud computing has surged into popularity in the IT industry, which can provision computing resource with a cost-effective, elastic solution. In recent years, a hybrid cloud paradigm is widely advocated by the industry practitioners, where a software-as-a-service (SaaS) provider although owning a small local data center can extend its services into a public infrastructure-as-a-service (IaaS) cloud. With such a paradigm, a SaaS provider can scale up and down its computing capacity by renting different numbers of virtual machines (VMs) in the public cloud according to the dynamic user demand instead of relying only on the fixed capacity of local data center. This can handle the dynamics of user requests elegantly and cost-effectively.

There are diverse successful user cases. For example, OpenText, a leading enterprise content management software provider, employs the hybrid cloud model to demonstrate their software, which makes their salespeople more productive and results in increased company revenue [1]. Oxford University uses hybrid cloud to support their database services, in which researchers are able to use the database service in their virtual machines provided on the public cloud which is connected to their private cloud [2]. One of the world's leading game companies, SEGA, uses hybrid cloud to improve its development process [3].

In fact, more and more leading IaaS cloud solutions (e.g., Amazon EC2 [4] and VMWare vCloud [5]) are now aiming at such a hybrid cloud paradigm. A SaaS provider can now quickly and seamlessly adopt such a computing paradigm with a set of handy tools from the IaaS providers.

But charming as it looks, the cost-effectiveness of such a paradigm highly depends on how well the SaaS provider can optimize the cost caused by renting VMs from the public IaaS cloud. Acquiring public IaaS computing capacity may actually cause a considerable cost. Recent years have also witnessed a lot of cases where many enterprises (e.g. Zynga and Uber) even consider to shift much of their operations off from the IaaS clouds and back to their own data centers, because of the high expenditure of renting VMs [6], [7]. Unfortunately, we still lack a good understanding of such a cost optimization problem, not to mention that there are no tools available for the cost-down task. This is an urgent call for attention to the research community.

Minimizing the cost of hybrid cloud operation is actually a very challenging task. First of all, the end users will be driven away if a SaaS cannot meet the service level agreement (SLA). In other words, a SaaS provider has to maintain its computing capacity while limiting the number of the VMs to reduce the renting cost at the same time. However, the user requests are highly dynamic in nature. Their

- S. Li, Y. Zhou, X. Yan, and X. Wang are with the School of Computer Science, Fudan University, Shanghai, China, and with the Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, China. E-mail: yfzhou@cse.cuhk.edu.hk.
- Y. Zhou and M.R.-T Lyu are with the Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen, China.
- L. Jiao is with the University of Göttingen, Göttingen, Germany.

Manuscript received 2 Nov. 2014; revised 31 Dec. 2014; accepted 5 Dec. 2014. Date of publication 5 Feb. 2015; date of current version 12 June 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2015.2390413

patterns cannot be known and even accurately predicted in advance. Moreover, the communication cost between the local servers and the public IaaS cloud cannot be ignored, which unfortunately inherits the dynamics if the number of the renting VMs are dynamically tuned. Finally, the prices of VMs in the public IaaS cloud are typically varying and unpredictable [4]. All these dynamic factors can have a great impact on the cost, and hence bring great difficulty to the cost minimization task.

However, existing approaches (e.g., [8] and [9]) on deciding the cost-efficient computing capacity of the cloud generally requires *a priori* knowledge of the user demand and the VM prices, or an accurate prediction. They also do not consider the dynamics of user requests. As a result, they are not specifically tailored for optimizing the cost of hybrid cloud operation.

This work, in contrast, aims at solving the above real-world challenges. Via a comprehensive theoretical analysis, we tackle the cost minimization problem with a fast online algorithm for dynamic cloud resource provisioning in hybrid clouds. Our analysis assumes no *a priori* knowledge on future user requests and the VM prices, and also takes the communication cost into considerations. Via modeling the problem with Lyapunov optimization framework [10], we can approach the minimum time average cost by anatomizing it into three sub-problems, each of which can be solved efficiently. We further show that the cost can be minimized by exploiting the trade-off between the delay of handling a request and the cost. We conduct our experimental study with real-world data from Amazon EC2. The results verify that we can achieve a satisfactory optimization results in real-world scenarios.

The rest of this paper is organized as follows. Section 2 introduces the system model and formulate the cost minimization problem. Section 3 analyzes problem and models it with the Lyapunov optimization framework which carefully addresses the real-world challenges. An online dynamic provision algorithm (ODPA) is then proposed to solve the problem. We examine the theoretical properties of the ODPA in Section 4 and provide our our experimental results with real-world data sets in Section 5. Section 6 discusses the related work and the paper is concluded in Section 7.

2 Related Work

With the rapid growth of cloud computing industry, the cloud resource provisioning problem has also attracted many research efforts in cost-optimizing perspective [8], [9], [11], [12], [13], [14], [15], [16].

In [8] and [11], the authors build game-theoretic models in a competing market to decide the number of different types of VMs provisioned by the cloud to optimize the profit of the SaaS provider. These approaches require the prediction of the user requests. [8] considers the uncertainty of VM prices and user requests in optimizing the resource provisioning cost. It develops a stochastic programming model and focuses mainly on the strategy of resources reservation in advance based on a given probability distribution of the user requests. [14] present a bunch of scheduling policies with a resource prediction model to estimate the level of resources in the virtual cluster system. Instead of prediction, [13] propose to optimize the provisioning cost based also on stochastic programming models. In contrast, our method optimizes the cloud resource provisioning cost considering both the cloud and local data center along with an arbitrary user request distribution, unknown VM price information, by mainly solving the linear programming problem.

In [15], the authors propose Cura, a new MapReduce cloud service model, which automatically creates the best configuration of clusters for the jobs in order to approach a global resource optimization. In particular, the system uses a deadline-awareness method, which delays the execution of certain jobs, and helps the resource allocation to reach its global optimization and further reduce the cost. In contrast, coincidently, we also leverage the delay information of job execution, but use the Lyapunov optimization to approximate the ideal minimized cost with a delay-cost trade off.

Recently, the hybrid cloud paradigm has been widely advocated where a SaaS provider owns a small local data center, but can extend its services into a public IaaS cloud. Hybrid cloud are embraced by more and more leading industry practitioners. Examples include the Amazon EC2 [4], VMWare vCloud [5], IBM Hybrid Cloud Solution [17], and CloudSwitch [18]. Tak et al. [19] investigate the economic issues of the application deployment choice in the hybrid cloud. Hajjat et al. [20] propose the migration strategy of enterprise application to the cloud that optimizes the benefits and ensures the security policies. Guo et al. [9] design a cost-efficient VM migration algorithm to help local data center scale to the cloud with optimal monetary cost in the scenario of cloud bursting, in which a local data center works together with a public cloud to handle workload peaks [21], [22]. Our work optimizes the cost by taking many practical issues into considerations, include the communication cost and the dynamics of user requests and VM prices, with an SLA guarantee.

Finally, Lyapunov optimization technique has been widely adopted in routing and resource allocation in several types of applications. Neely [23] establishes a simple Lyapunov drift to achieve both system stability and performance optimization in time varying wireless networks. Amble et al. [24] design stable policies for routing requests based on Lyapunov optimization framework in content distribution networks. Recently, much research attention has been paid in applying Lyapunov optimization technique to control power management and resource allocation in data centers [25], [26], [27]. Inspired by this track of research efforts, we adopt Lyapunov optimization into the optimal cloud resource provisioning problem by tailoring the framework according to specifics of the hybrid cloud settings.

3 System Model and Problem Formulation

3.1 System Model

In this section, we focus on the system model of the problem we are facing. We consider a SaaS provider operating with a small local data center (or *local servers* in our following discussions) and The provider can also extend its service

TABLE 1
Notation

Symbols	Descriptions
T	Minimum renting period of reserved VMs
τ_i	The ith decision slot for the on-demand VMs and the on-spot VMs
τ	The length of each decision slot for the on-demand VMs and the on-spot VMs. $T = \mathrm{m}\tau$ where m is an integer.
$R(\tau_i)$	The number of reserved VMs at time slot τ_i
R^{max}	The upper bound of the number of the reserved VMs
$D(\tau_i)$	The number of on-demand VMs at time slot τ_i
D^{max}	The upper bound of the number of the on-demand VMs
$S(\tau_i)$	The number of on-spot VMs at time slot τ_i
S^{max}	The upper bound of the number of the on-spot VMs
$L(\tau_i)$	The number of VMs in the local servers at time slot τ_i
L^{max}	The upper bound of the number of the VMs in the local servers
$W(\tau_i)$	The lower bound of the number of VMs at time slot τ_i
$\lambda(\tau_i)$	The arrival rate of user requests at time slot τ_i
$\mu(\tau_i)$	The service rate of user request at time slot τ_i
d^{max}	The maximum allowed queueing time of each user request
$Q(\tau_i)$	The backlog of the user request queue
Q^{max}	The maximum of the queue backlog
$Z(\tau_i)$	The backlog of the virtual queue
Z^{max}	The maximum of the virtual queue backlog
ϕ	The cost of running the local servers
$P_r(\tau_i)$	The price of reserved VMs at τ_i
$P_d(\tau_i)$	The price of on-demand VMs at τ_i
$P_s(\tau_i)$	The price of on-spot VMs at τ_i
$M(\tau_i)$	The communication cost between VMs in the local servers and those in the cloud
$\beta(\tau_i)$	The service response time at time slot τ_i
β^{max}	The service response time upper bound requirment
ϵ	The parameter bounding growing rate of the virtual queue
V	Parameter for the trade-off between queueing delay and the cost

capacity via renting the VMs from a public IaaS cloud. This is a typical hybrid cloud model.

The public IaaS cloud will in general provision three types of VM services for the SaaS provider. The first is the *reserved* VM service, which is long-term service with a fixed VM numbers. We let T denote the minimum allowed renting period of such services. T is typically several days or months [4], [28]. The second is the *on-demand* VM service, where the number of VMs can be set instantaneously by the SaaS provider. The last is the *on-spot* VM services, which the SaaS provider can bid for. The price of the reserved service per unit is typically the lowest. The on-demand one is often the most expensive but it is charged in a pay-as-you-go fashion. The price of the on-spot one is dynamic according to the user bid (reflecting the user demand). The above is a typical IaaS provision scheme for current public IaaS cloud (e.g., the Amazon EC2).

Since renting the public cloud resources incurs monetary cost, the SaaS provider should reduce such cost. The key problem to the SaaS provider is how to decide the number of VMs it need to rent so that the performance requirement is satisfied and the cost is minimized. We use the symbols R, D, and S to denote the numbers of VMs of the above three services an SaaS provider will rent respectively. Also let L denote the numbers of the VMs in the local servers the SaaS provider owns. Let R^{max}, D^{max}, S^{max} and L^{max} denote the upper bound of the numbers of each VM type, and let W denote the lower bound of the numbers of all of VMs.

In the real-world scenario, the SaaS provide has to make decisions on the number of the VMs in some degrees of granularity. Since the allowed minimum renting period of reserved VMs is T, we consider the SaaS provider will divide its operation period into a sequence of time intervals with length T and determine the number of the reserved VMs at the beginning of each interval. We name such a decision interval with length T a coarse-grained decision interval.

The numbers of the other two types of VMs (the on-demand ones and the on-spot ones) can be decided in a relatively short-term manner. Without loss of generality, we consider the operation time can be divided into many non-overlapping small decision slots with length τ. Let τ_1, $\tau_2, \ldots, \tau_i, \ldots$ (i is a non-negative integer) denote such slots in sequence. The length of each τ_i is τ. For example, in Amazon EC2, τ is one hour, which means that the SaaS provider can make the decisions of the on-demand VMs and the on-spot ones per hour. We name such a decision slots with length τ a *fine-grained* decision time slot.

Without loss of generality, suppose $T = m\tau$. In other words, the length of the coarse-grained decision interval is m times of the fine-grained decision time slot. The SaaS provide can decide the number of the reserved VMs in the beginning of time slot τ_0, τ_m, $\tau_{2m}, \ldots, \tau_{jm}, \ldots$ where j is a non-negative integer.

In this way, how to determine the numbers of the above four types of VMs for the SaaS provider can be divided into a two-scale decision process. Finally, Table I provides a summary of the symbols used in this paper.

3.2 User Requests and SaaS Service Model

Consider the users access the SaaS provider with an arrival rate $\lambda(\tau_i)$. Note that $\lambda(\tau_i)$ can bear an *arbitrary* distribution in realistic scenarios.

The user requests to the SaaS provider usually have deadlines, allowing the request to wait for a maximum periods of time d^{max} before it is scheduled. Consider the requests can be stored in a queue denoted by $Q(\tau_i)$ in each fine-grained decision time slot. In each find-grained interval, the user requests in the queue are served with the rate $\mu(\tau_i)$. Therefore, the queue has the following update equation:

$$Q(\tau_{i+1}) = \max\{Q(\tau_i) - \mu(\tau_i), 0\} + \lambda(\tau_i). \tag{1}$$

To serve the user requests, the SaaS provider allocates each request a certain number of VMs according to the request requirement, consisting of those in the local servers and those in the public IaaS cloud. The service rate in each time slot τ_i in a coarse-grained decision interval is

$$\mu(\tau_i) = (L(\tau_i) + R(\tau_i) + D(\tau_i) + S(\tau_i)) \cdot c, \tag{2}$$

where $L(\tau_i)$, $R(\tau_i)$, $D(\tau_i)$, and $S(\tau_i)$ denote the number of the local server VMs, reserved VMs, on-demand VMs, and on-spot VMs in time slot τ_i, respectively. c is the service capacity of a single VM, *i.e.*, the number of requests that can be served by a VM per time slot. Without loss of generality, we assume that VMs are homogenous in terms of service capacity (otherwise, the capacity of VMs can be normalized theoretically). We consider the real-world case that the allocated VMs may communicate with each other. In practice, the intra-communications among the local servers or in the cloud *per se* are free, while inter-communications between the local servers and the IaaS cloud will be charged.

3.3 Cost Model

We consider that the cost of the SaaS provider includes running the local servers, purchasing three types of VMs, and the communication cost between the VMs across the cloud and local servers, as follows:

$$\begin{aligned} Cost(\tau_i) = \phi + R(\tau_i)P_r(\tau_i) + D(\tau_i)P_d(\tau_i) \\ + S(\tau_i)P_s(\tau_i) + M(\tau_i), \end{aligned} \tag{3}$$

where ϕ is the cost of running the local servers, which is a constant without loss of generality, and $M(\tau_i)$ denotes the communication cost between the VMs in the local servers and those in the cloud. $R(\tau_i)$, $D(\tau_i)$, and $S(\tau_i)$ are the numbers of the reserved, on-demand, and on-spot VMs in time slot τ_i respectively, and $P_r(\tau_i)$, $P_d(\tau_i)$, and $P_s(\tau_i)$ are their prices respectively. Note that $R(\tau_i)$ and $P_r(\tau_i)$ remain constant in each coarse-grained decision interval, *i.e.*, from τ_{jm} to $\tau_{(j+1)m-1}$ where j is a non-negative integer.

3.4 Problem Formulation

The SaaS provider can optimize its cost via a two-scale decision process: decide the reserved VM number in each coarse-grained decision interval with length T and decide the numbers of the on-demand and on-spot VMs in each fine-grained decision time slot τ_i in the interval, such that the time-average expecting cost is minimized. Formally,

$$\min lim_{t\to\infty} \frac{1}{t}\sum_{i=0}^{t-1} \mathbb{E}[Cost(\tau_i)] \tag{4}$$

s.t.

$$\begin{aligned} 0 \le L(\tau_i) \le L^{max}, 0 \le R(\tau_i) \le R^{max}, \\ 0 \le D(\tau_i) \le D^{max}, 0 \le S(\tau_i) \le S^{max}. \end{aligned} \tag{5}$$

$$L(\tau_i) + R(\tau_i) + D(\tau_i) + S(\tau_i) \ge W(\tau_i) \tag{6}$$

$$\forall \tau_i, Q(\tau_i) < Q^{max} \tag{7}$$

$$0 \le \beta(\tau_i) \le \beta^{max}. \tag{8}$$

The symbol $\mathbb{E}$ denotes the statistical expectation. The detailed descriptions of Equations (5)-(8) are as follows.

Equation (5). The SaaS provider can bound the numbers of VMs of the four types by L^{max}, R^{max}, D^{max}, and S^{max} respectively. For instance, the number of reserved VMs can be bounded if the provider just want to purchase a small number of reserved VMs due to its service type, or it prefers to limit the purchasing cost.

Equation (6). In each time slot τ_i, the provisioned VMs must satisfy the capability requirement $W(\tau_i)$ ensuring that the requests can be served in a quality required by the SaaS provider.

Equation (7). Each request in the queue has a maximum delay, i.e., the request should be served before the deadline. To guarantee this, the backlog of queue $Q(\tau_i)$ should satisfy this equation, where Q^{max} is the maximum backlog. The Section 3.2 will show more details on this technique.

Equation (8). An SLA contract is established between the SaaS provider and the its users, specifying a given response time requirement β^{max}. Thus, to ensure that the SLA contract is not violated, the response time $\beta(\tau_i)$ should be bounded.

4 Online Dynamic Allocation Algorithm

To achieve the minimum time-averaged cost in Equation (4) is a huge challenge, since the SaaS provider cannot have the knowledge of the future user requests and VM prices in advance in the real practice.

In this section, we first discuss how we tackle the resource requirement heterogeneity of user requests, which is typical in real-world scenarios. Then we discuss our virtual queue notion to bound the request delay so as to meet the SLA contract. Based on these considerations, we then build the Lyapunov optimization model [10] and convert the cost minimization problem into a solvable one. We design an online dynamic provision algorithm to solve this problem. ODPA is able to approach the minimum time average cost without any *a priori* knowledge of the future user request workload and the future IaaS VM prices.

4.1 Heterogeneity-Aware Sub-Queues

One of the major challenges to build a queueing model for a hybrid cloud is to handle the heterogeneity of user requests to the SaaS. In realistic scenarios, different user requests may involve different computational resources of VMs, and incur different resource requirements. For example, a request may start a computation-intensive job and require more CPU capacity but less I/O capacity.

Consider the local servers of the SaaS provider that is typically limited in resource. A request may wait in the queue because one of the resource requirements cannot be entertained as the resource is a bottleneck. For example, in case that the CPU of the local servers is extensively exploited, the computation-intensive requests would have to be blocked even if another resource such as the memory is still quite sufficient.

Hence, we should take care of such heterogeneity of user requests. We consider the queue is occupied with heterogeneous user requests and tackle this real-world challenge with a queue anatomy approach. To the best of our knowledge, we are the first to take the request heterogeneity into the Lyapunov optimization queueing model.

To address this issue, we map the queue $Q(\tau_i)$ into a set of *sub-queues* $Q_k(\tau_i)$, $k = 1, 2, \ldots, n$. The operation of the sub queue is described as follows. Each sub-queue $Q_k(\tau_i)$ is corresponding to a certain type of resource (suppose there are n types of resources), like vCPU number, memory size requirement, and so on. One request is composed of the requests of each resources. For example, a user request need 2 vCPUs and 4 GB memory in total to finish its services. Among those resources, there must be one resource that is the bottleneck resource, which is the most scarce one in the cloud and is about to affect the service performance. Thus, when a request arrives, its bottleneck resource first enters a sub queue, and its requests for other resources also enters every sub-queue $Q_k(\tau_i)$ respectively. When the request is queueing in $Q(\tau_i)$, we consider it is queueing for each type of resource in $Q_k(\tau_i)$. The requst leaves $Q(\tau_i)$ only if it can be served by the bottleneck resource p and leave the corresponding sub-queue $Q_p(\tau_i)$ at the same time. In this case, the request also leave the other sub-queues.

Therefore, the service rate of $Q(\tau_i)$ (as well as that of each sub-queue $Q_k(\tau_i)$) is determined by the service rate of the bottleneck sub-queue $Q_p(\tau_i)$. We name the bottleneck sub-queue the prime sub-queue and the others the accompany sub-queues. Note that it is also easy to alter the prime sub-queue to another one if the bottleneck computing resource changes. We thus model the heterogeneity of user requests with such sub-queues.

4.2 Delay-Aware Virtual Queue

As previously mentioned, user request has a deadline d^{max} to meet after it enters $Q(\tau_i)$. To handle this situation, we apply *ϵ-persistent service queue* technique [29] to bound the worst-case delay of the dequeuing operations, so as to ensure that a request can be served before its deadline.

Let $Z_k(\tau_i)$ denote a queue associated with $Q_k(\tau_i)$. Its update equation is as follows:

$$Z_k(\tau_i+1) = \max\{Z_k(\tau_i) - \mu_k(\tau_i) + \epsilon_k 1_{Q_k(\tau_i)>0}, 0\}, \quad (9)$$

where $1_{Q_k(\tau_i)>0}$ is an indicator function taking the value 1 if $Q_k(\tau_i) > 0$, and 0 otherwise. ϵ_k is a parameter that controls the growing rate of the delay-aware virtual queue Z_k, which has an impact on the queueing time of a request (We will discuss the details on ϵ_k later). Note that queue Z_k is only a virtual queue, which is not the real request queue as queue Q_k. The reason that we bring in virtual queue is that it is useful to help us to bound the deadline the the requests, and is able to be integrated in the Lyapunov optimization framework.

Queue Z_k has the same service rate as that of queue Q_k, but has a different growing process. According to Eq. (9), queue Z_k grows if and only if queue Q_k is not empty.

Recall that queue Q_k is bounded (Eq. (7)), i.e., $\forall\tau_i, Q(\tau_i) < Q_k^{max}$, where Q_k^{max} is the maximum backlog of Q_k. According to Eq. (9), we can also find that virtual queue Z_k is bounded, *i.e.*, $\forall\tau_i, Z(\tau_i) < Z_k^{max}$, where Z_k^{max} is the maximum backlog of Z_k. To ensure that the requests are served before the deadline, we have the following lemma.

Lemma 1. *Given that $Q_k(\tau_i) \le Q_k^{max}$ and $Z_k(\tau_i) \le Z_k^{max}$, the user requests have the maximum delay of d_k^{max}, in which:*

$$d_k^{max} = \lceil (Q_k^{max} + Z_k^{max})/\epsilon_k \rceil \quad (10)$$

Proof. We prove this by contradiction, *i.e.*, assuming that requests $\lambda_k(t)$ are served at $t+d'$, where $d' > \lceil (Q_k^{max} + Z_k^{max})/\epsilon_k \rceil$. Since queues are served in the FIFO manner, during the time interval $[t+1, t+d']$, arrived requests are not served, *i.e.*, the served requests are only the ones arrived before t, and thus we have

$$\sum_{\tau_i=t+1}^{t+d'} \mu_k(\tau_i) \le Q_k(t) - \lambda_k(t) < Q_k(t) < Q_k^{max}. \quad (11)$$

According to Equation. (9), we have $Z_k(t+1) \ge Z_k(t) - \mu_k(t) + \epsilon_k$. By summing it during the time interval $[t+1, t+d']$, we further have $Z_k(t+d') - Z_k(t) \ge -\sum_{\tau_i=t+1}^{t+d'} \mu_k(\tau_i) + d'\epsilon_k$. Since it is already known that $Z_k(t) \ge 0$ and $Z_k(t+d') \le Z_k^{max}$, we have

$$\sum_{\tau_i=t+1}^{t+d'} \mu_k(\tau_i) \ge d'\epsilon_k - Z_k^{max} \quad (12)$$

Combine Equations (11) and (12), we get

$$d' < (Q_k^{max} + Z_k^{max})/\epsilon_k. \quad (13)$$

Equation (13) is contradictory with the assumption. Thus Lemma 1 is proved. □

For parameters ϵ_k and ϵ_l for sub-queues k and l respectively, we have the following relationship.

Theorem 1. *Given that $\forall k$, $Q_k(\tau_i) \le Q_k^{max}$ and $Z_k(\tau_i) \le Z_k^{max}$, $\forall k, l, k \ne l$, the following equation holds:*

$$\frac{\epsilon_k}{\epsilon_l} = \frac{Q_k^{max} + Z_k^{max}}{Q_l^{max} + Z_l^{max}}. \quad (14)$$

Proof. Since the property of the sub-queues implies that the elements with the same index has the same dequeue time, we have $d_k^{max} = d_l^{max}$. Then using Eq. (10) induced by Lemma. 1, we can get Eq. (14). □

4.3 Lyapunov Optimization

At each time slot τ_i, we select and control the prime queue $Q_p(\tau_i)$ (i.e., p is the bottleneck resource index), and other sub-queues are also controlled following the bottleneck one.

Let $\Theta(\tau_i)$ denote the vector $[Q_p(\tau_i), Z_p(\tau_i)]$. We define a *Lyapunov function* as follows:

$$L(\Theta(\tau_i)) \triangleq \frac{1}{2}\left[Q_p(\tau_i)^2 + Z_p(\tau_i)^2\right]. \quad (15)$$

The one-slot conditional *Lyapunov drift* is defined as

$$\Delta(\Theta(\tau_i)) \triangleq \mathbb{E}\{L(\Theta(\tau_i+1)) - L(\Theta(\tau_i)) | \Theta(\tau_i)\}. \quad (16)$$

Following the *drift-plus-penalty* algorithm [30], our aim is to make decisions on the state of VMs to minimize the upper bound of the following drift-plus-penalty expression given the current system state:

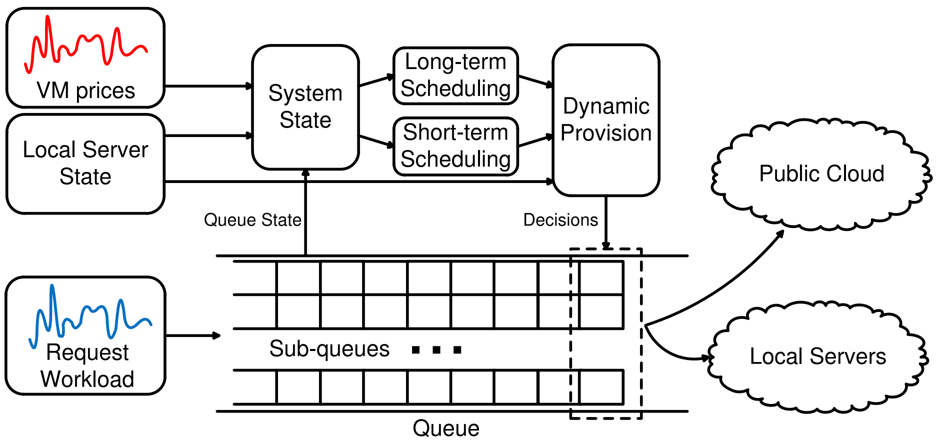


Fig. 1. Architecture of the system and ODPA.

$$\Delta(\Theta(\tau_i)) + V\mathbb{E}\{Cost(\tau_i)|\Theta(\tau_i)\}, \tag{17}$$

where V is a parameter determined by the SaaS provider to achieve a best tradeoff between queueing delay and the cost, to be discussed later.

The following theorem derives the upper bound of the drift-plus-penalty expression.

Theorem 2. *Assume that* $0 \le \lambda_p \le \lambda_p^{max}$, $0 \le \mu_p \le \mu_p^{max}$, *and* $Q_p(\tau_i) < Q_p^{max}$, *the drift-plus-penalty expression satisfies:*

$$\begin{aligned}\Delta(\Theta(\tau_i)) &+ V\mathbb{E}\{Cost(\tau_i)|\Theta(\tau_i)\}\\ &\le B + V\mathbb{E}\{Cost(\tau_i)|\Theta(\tau_i)\}\\ &+ Q_p(t)\mathbb{E}\{(\lambda_p(\tau_i) - \mu_p(\tau_i))|\Theta(\tau_i)\}\\ &+ Z_p(t)\mathbb{E}\{(\epsilon_p - \mu_p(\tau_i))|\Theta(\tau_i)\},\end{aligned} \tag{18}$$

where B is

$$B = \frac{1}{2}\max\{(\mu_p^{max})^2, \epsilon_p^2\} + \frac{1}{2}((\mu_p^{max})^2 + (\lambda_p^{max})^2). \tag{19}$$

Proof. According to Equation (1), we have

$$\begin{aligned}Q_p(\tau_i+1)^2 &= (\max\{Q_p(\tau_i) - \mu_p(\tau_i), 0\} + \lambda_p(\tau_i))^2\\ &\le Q_p(\tau_i+1)^2 + \mu_p(\tau_i)^2 + \lambda_p(\tau_i)^2 + 2Q_p(\lambda_p(\tau_i) - \mu_p(\tau_i)).\end{aligned}$$

Then we can obtain the following:

$$\begin{aligned}&\frac{Q_p(\tau_i+1)^2 - Q_p(\tau_i)^2}{2}\\ &\le Q_p(\tau_i)(\lambda_p(\tau_i) - \mu_p(\tau_i)) + \frac{(\lambda_p(\tau_i) - \mu_p(\tau_i))^2}{2}\\ &\le Q_p(\tau_i)(\lambda_p(\tau_i) - \mu_p(\tau_i)) + \frac{1}{2}((\mu_p^{max})^2 + (\lambda_p^{max}))^2.\end{aligned}$$

According to Equation (9), we have

$$Z_p(\tau_i+1) \le Z_p(\tau_i) - \mu_p(\tau_i) + \epsilon_p.$$

Squaring both sides yields

$$Z_p(\tau_i+1)^2 \le (Z_p(\tau_i) - \mu_p(\tau_i) + \epsilon_p)^2$$

and then we have the following:

$$\begin{aligned}&\frac{Z_p(\tau_i+1)^2 - Z_p(\tau_i)^2}{2}\\ &\le Z_p(\tau_i)(\epsilon_p - \mu_p(\tau_i)) + \frac{(\mu_p(\tau_i) - \epsilon_p)^2}{2}\\ &\le Z_p(\tau_i)(\epsilon_p - \mu_p(\tau_i)) + \frac{1}{2}\max\{(\mu_p^{max})^2, \epsilon_p^2\}.\end{aligned}$$

We can get the following based on the above inductions:

$$\begin{aligned}&L(\Theta(\tau_i+1)) - L(\Theta(\tau_i))\\ &= \frac{Q_p(\tau_i+1)^2 - Q_p(\tau_i)^2}{2} + \frac{Z_p(\tau_i+1)^2 - Z_p(\tau_i)^2}{2}\\ &\le B + Q_p(\tau_i)(\lambda_p(\tau_i) - \mu_p(\tau_i)) + Z_p(\tau_i)(\epsilon_p - \mu_p(\tau_i)).\end{aligned}$$

By taking the expectation and adding $V\mathbb{E}\{Cost(\tau_i)|\Theta(\tau_i)\}$ to both sides, we can obtain Equation (18). □

Rewriting Equation (18) by substituting $\mu(\tau_i)$ and Cost (τ_i) in Equations (2) and (3), we turn the drift-plus-penalty upper bound minimization problem into the following problem **P0**:

$$\begin{aligned}\min &\mathbb{E}\{R(\tau_i)(VP_r(\tau_i) - Q_p(\tau_i) - Z_p(\tau_i))|\Theta(\tau_i)\}\\ &+ \mathbb{E}\{D(\tau_i)(VP_d(\tau_i) - Q_p(\tau_i) - Z_p(\tau_i))|\Theta(\tau_i)\}\\ &+ \mathbb{E}\{S(\tau_i)(VP_s(\tau_i) - Q_p(\tau_i) - Z_p(\tau_i))|\Theta(\tau_i)\}\\ &+ \mathbb{E}\{L(\tau_i)(-Q_p(\tau_i) - Z_p(\tau_i)) + VM(\tau_i)|\Theta(\tau_i)\}\end{aligned} \tag{20}$$

s.t. Equations (5), (6), (7), (8).

4.4 Online Dynamic Provision Algorithm

By analyzing the optimization problem with a Lyapunov optimization framework, we are able to tackle the complicated time-average cost minimization problem into **P0**. **P0** can then be decoupled into three parts, namely, the *long-term scheduling*, the *short-term scheduling*, and the *dynamic provisioning*. It can then be handled with a separation-of-concern approach: Every part can be solved individually with efficient algorithms while the time-average cost can be minimized. Fig. 1 overviews the framework of ODPA. What follows elaborates these three key parts.

4.4.1 Long-Term Scheduling

The SaaS provider optimizes its cost at the beginning of each coarse-grained interval with length T.

The number of reserved VMs to be rented can be decided with the following optimization problem **P1**:

$$\begin{aligned} \min_{R(\tau_i)} \; & R(\tau_i)(VP_r(\tau_i) - Q_p(\tau_i) - Z_p(\tau_i)) \\ \text{s.t. } & \text{Equations } (5),\ (6),\ (7),\ (8). \end{aligned} \tag{21}$$

It is easy to see that the objective and constraints of **P1** are all linear. Hence, **P1** can be efficiently solved with linear programming.

4.4.2 Short-Term Scheduling

The SaaS provider decides the numbers of the local servers VMs, the on-demand VMs, and the on-spot VMs in every fine-grained time slot τ_i of each coarse-grained interval. The system state includes $Q_p(\tau_i)$, $Z_p(\tau_i)$, $\lambda_p(\tau_i)$, the pricing information $P_d(\tau_i)$, $P_s(\tau_i)$, and the remained resource capacities of local servers, *e.g.*, remained CPU, memory, and etc.

With the knowledge of long-term scheduling (the solution to **P1**), the numbers of the various VMs can be optimized via solving the following problem **P2**:

$$\begin{aligned} \min_{D(\tau_i),S(\tau_i),L(\tau_i)} \; & D(\tau_i)(VP_d(\tau_i) - Q_p(\tau_i) - Z_p(\tau_i)) \\ & + S(\tau_i)(VP_s(\tau_i) - Q_p(\tau_i) - Z_p(\tau_i)) \\ & + L(\tau_i)(-Q_p(\tau_i) - Z_p(\tau_i)) \\ \text{s.t. } & \text{Equations } (5),\ (6),\ (7),\ (8). \end{aligned} \tag{22}$$

Similar to **P1**, **P2** is also a linear optimization problem which can be easily solved with linear programming.

We now theoretically analyze the performance of ODPA scheduling. We first give the worst case delay of the requests and then find the performance bound of the algorithm.

Lemma 2. *If ODPA is implemented and given that* $0 < P_r(t) \le P_r^{max}$, $0 < P_d(\tau_i) \le P_d^{max}$, $0 < P_s(\tau_i) \le P_s^{max}$, $0 \le \epsilon_k \le \mu_k^{max}$, *and* $V > 0$, *we have:*

(i). The length of Q_k *and* Z_k *can be bounded by the following:*

$$\begin{aligned} Q_k^{max} &= V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max}, \\ Z_k^{max} &= V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_k. \end{aligned} \tag{23}$$

(ii). The worst case delay d_k^{max} *of the requests in sub-queue* i *is:*

$$d_k^{max} = \left\lceil \frac{2V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max} + \epsilon_k}{\epsilon_k} \right\rceil. \tag{24}$$

(iii). The worst case delay d_k^{max} *and* d_l^{max} *of sub-queue* k *and* l *are equal,* $\forall k, l, k \neq l$*:*

$$d_k^{max} = d_l^{max}. \tag{25}$$

Proof. (i). To prove this, we need to prove $\forall \tau_i$, $Q_k(\tau_i) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max}$ and $Z_k(\tau_i) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_k$. We prove this by induction. For $\tau_i = 0$, we have $Q_k(0) = 0 \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max}$ and $Z_k(0) = 0 \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_k$. Assume that the equations hold for τ_i, we induce that they also hold for $\tau_i + 1$. We first consider Q_k. (1) if $Q_k(\tau_i) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\}$. We can obtain the following:

$$\begin{aligned} Q_k(\tau_i + 1) &= \max\{Q_k(\tau_i) - \mu_k(\tau_i), 0\} + \lambda_k(\tau_i) \\ &\le Q_k(\tau_i) + \lambda_k(\tau_i) \\ &\le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k(\tau_i) \\ &\le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max}. \end{aligned}$$

(2) In the other case, if $V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} < Q_k(\tau_i) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max}$. In this case, the following holds:

$$Q_k(\tau_i) + Z_k(\tau_i) \ge Q_k(\tau_i) > V\max\{P_r^{max}, P_d^{max}, P_s^{max}\}.$$

Based on this inequation, according to Equation (20), we can see that $R(\tau_i)$, $D(\tau_i)$, $S(\tau_i)$ and $L(\tau_i)$ would choose its maximum value respectively, and thus $\mu_k(\tau_i) = (L^{max} + R^{max} + D^{max} + S^{max}) \cdot c = \mu_k^{max}$. In case (2.1), if $Q_k(\tau_i) - \mu_k^{max} \le 0$, refers to Equation (1), we have

$$\begin{aligned} Q_k(\tau_i + 1) &= \lambda_k(\tau_i) \le \lambda_k^{max} \\ &\le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max}. \end{aligned}$$

(2.2) If $Q_k(\tau_i) - \mu_k^{max} > 0$. Recall that we assumed the queue Q_k is finite(Equation (7)) and thus in the optimization process $\mu_k^{max} \ge \lambda_k^{max}$ holds, we have

$$\begin{aligned} Q_k(\tau_i + 1) &= Q_k(\tau_i) - \mu_k^{max} + \lambda_k^{max} \\ &\le Q_k(\tau_i) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max}. \end{aligned}$$

Hence, we prove that

$$Q_k(\tau_i + 1) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_k^{max},$$

and thus prove the case of Q_k.

We next consider Z_k, which is similar to Q_k. (1) if $Z_k(\tau_i) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\}$, we have

$$\begin{aligned} Z_k(\tau_i + 1) &= \max\{Z_k(\tau_i) - \mu_k(\tau_i) + \epsilon_k 1_{Q_k(\tau_i)>0}, 0\} \\ &\le Z_k(\tau_i) + \epsilon_k \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_k. \end{aligned}$$

(2) In the other case, if $V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} < Z_k(\tau_i) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_k$. Similar to the case of Q_k, $\mu_k(\tau_i) = \mu_k^{max}$. (2.1) if $Z_k(\tau_i) \le \mu_k^{max} - \epsilon_k$, refers to Equation (9), we have

$$Z_k(\tau_i + 1) = 0 \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_k.$$

(2.2) If $Z_k(\tau_i) > \mu_k^{max} - \epsilon_k$, and according to the assumption that $0 \le \epsilon_k \le \mu_k^{max}$, we have

$$\begin{aligned} Z_k(\tau_i + 1) &= Z_k(\tau_i) - \mu_k^{max} + \epsilon_k \le Z_k(\tau_i) \\ &\le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_k. \end{aligned}$$

Hence, we prove that

$$Z_k(\tau_i) \le V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_k,$$

and thus prove the case of Z_k.

(ii). Combine Equations (10) and (23), we can easily get Equation (24).

(iii). The settings of sub-queues infer that the elements in the sub-queues have the same time interval for enqueue and dequeue operation for each request, and thus the maximum deadlines are the same. □

Based on this lemma, we derive the following theorem.

Theorem 3. *ODPA can bound the time average cost as follows:*

$$lim_{t\to\infty}\frac{1}{t}\sum_{i=0}^{t-1}\mathbb{E}[Cost(\tau_i)] \le \frac{1}{t}\sum_{i=0}^{t-1}c^*_{\tau_i} + \frac{B}{V}. \quad (26)$$

Proof. We define $c^*_{\tau_i}$ as the minimum ideal value of $\mathbb{E}[Cost(\tau_i)]$ if the request information is already known before time slot τ_i, subjects to $\mu_p(\tau_i) - \lambda_p(\tau_i) \le 0$ and $\mu_p(\tau_i) - \epsilon_p \le 0$. Inspired by [31], we define the 1-slot sample path drift $\Delta_1(\Theta(\tau_i))$ without expectation as following:

$$\Delta_1(\Theta(\tau_i)) \triangleq L(\Theta(\tau_{i+1})) - L(\Theta(\tau_i)).$$

Similar to the previous Lyapunov analysis, we have

$$\Delta_1(\Theta(\tau_i)) \le B - Q_p(\tau_i)(\mu_p(\tau_i) - \lambda_p(\tau_i)) - Z_p(\mu_p(\tau_i) - \epsilon_p).$$

By adding $V\mathbb{E}[Cost(\tau_i)]$ to both sides, and using the optimizing process discussed in ODPA which finds the optimum μ^*_p, we can obtain

$$\begin{aligned}\Delta_1(\Theta(\tau_i)) + V\mathbb{E}[Cost(\tau_i)] &\le B + c^*_r \\ &- Q_k(\tau_i)(\mu^*_p(\tau_i) - \lambda_p(\tau_i) \\ &- Z_k(\tau_i)(\mu^*_p(\tau_i) - \epsilon_k) \\ &\le B + Vc^*_r.\end{aligned}$$

By summing the above equation over t slots, and dividing by kV, we can get

$$\frac{L(\Theta(\tau_t) - L(\Theta(0))}{tV} + \frac{1}{t}\sum_{i=0}^{t-1}\mathbb{E}[Cost(\tau_i)] \le \frac{1}{t}\sum_{i=0}^{t-1}c^*_r + \frac{B}{V}.$$

Since $L(\Theta(t)) \ge L(\Theta(0)) = 0$ and we extend t to the infinity, we can prove the theorem:

$$lim_{t\to\infty}\frac{1}{t}\sum_{i=0}^{t-1}\mathbb{E}[Cost(\tau_i)] \le \frac{1}{t}\sum_{i=0}^{t-1}c^*_{\tau_i} + \frac{B}{V},$$

□

where $\frac{1}{t}\sum_{i=0}^{t-1}c^*_{\tau_i}$ is the minimum possible time average cost if the future information of subsequent slots is known, B is defined in Equation (19).

The above theorem shows the $[O(1/V), O(V)]$ trade-off between cost and delay. By increasing the value of V, we can get the near-optimal value of time average cost but bring in larger queue length and longer delay. We verify it with our experimental study with real-world data in what follows.

4.4.3 Dynamic Provisioning

We now consider the communication cost $M(\tau_i)$. Given the solutions of long-term and short-term scheduling, we should then decide the locations of the VMs to minimize the communication cost with SLA guarantee. As discussed before, although the intra communication of the local servers or the cloud is free, the communication between the local servers and the cloud will be charged. A good provision of VMs to the local servers and the IaaS cloud should be able to minimize the communication cost at each slot. The optimization problem is defined in the following **P3** formulation

$$\begin{aligned}\min \quad & V\cdot M(\tau_i) \\ \text{s.t.} \quad & \text{constraint}(8).\end{aligned} \quad (27)$$

Unfortunately, **P3** is not an easily tractable problem. We prove it NP-complete, and then resort to a fast online heuristic algorithm.

Theorem 4. *The problem of **P3** is NP-complete.*

Proof. By regarding the VMs as the vertices and the communication links between the local servers and the IaaS as the edges of the graph, the SaaS system can be modeled as a graph. Each edge is then weighted with its corresponding communication cost. The local servers and the IaaS clouds are then regarded as two fix-sized partitions of the graph. **P3** is to find the partition that minimize the edge weights across the two fixed unequal sized sets.

We now show that this problem belongs to NP. For a given graph $G = (V, E)$ and size $|V|$, we use the edge set $E' \subseteq E$ as a certificate for G. For each edge e in E', we can check whether its two endpoints are in different partitions and whether $|\{e|\forall e \in E'\}|$ is equal to $|V|$ in polynomial time. In other words, verifying the certificate can be solved in polynomial time, and hence **P3** belongs to NP.

We next prove **P3** NP-hard by reducing it from Minimum Graph Bisection problem (MGBP), a known NP-complete problem [32]. Given an instance $G = (V, E)$ ($|V| = 2n_1,\ n_1 \in N^+$) of MGBP, we reduce it to an instance of **P3**. Assuming $n_2 > n_1$, we form $G' = (V', E')$ by adding a clique of size $n_2 - n_1$ with $+\infty$ edge weights to G. This can be done in polynomial time. We then show this transformation is in nature a reduction.

Suppose $S \subseteq E$ is a cut set of size $|V|$ in G. S is also a cut set of size $|V|$ in G', *i.e.*, $S = S'$, since edges of new added clique have $+\infty$ weights and they cannot be in the cut set of G'. Conversely, suppose $S' \subseteq E'$ is a cut set of size $|V|$ in G'. Since clique edges cannot be in the cut set due to their positive infinity weights, we have $S' = S$.

Hence, we prove that S is a cut set of size $|V|$ in G if and only if S' is a cut set of size $|V|$ in G'. The reduction is then proved. As a result, **P3** is NP-hard. Since **P3** is NP as well as NP-hard, it is NP-complete. □

Next, we designed a fast heuristic algorithm that can efficiently solve **P3** illustrated in Algorithm 1. We modify the state-of-the-art Fiduccia-Mattheyses algorithm [33] by bringing the moving penalty between two partitions and the SLA contract into consideration.

The algorithm first computes the gain of all VMs in each iteration, greedily selects the VM of maximum gain and

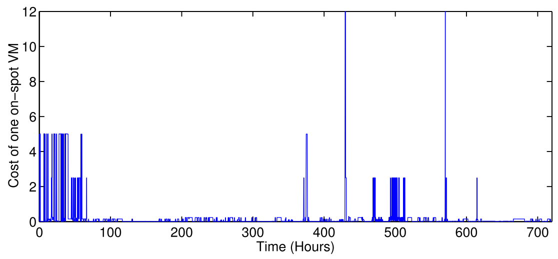


Fig. 2. Price of on-spot VMs.

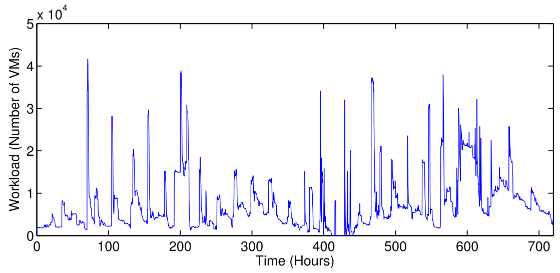


Fig. 3. User request workload.

updates the adjacent VMs until all VMs selected. It then executes the first x moves that maximize the sum of those gains. The gain, which is the cost reduction after moving one VM to the other partition, is defined as follows:

$$g(c) = E_c - I_c - P_c, \tag{28}$$

where E_c and I_c are the external cost and internal cost of VM c respectively, i.e., the communication cost with the VMs in the other partition and VMs in the same partition (if each communication is charged), and P_c is the possible extra penalty caused by the moving operation, *i.e.*, the violation of the communication cost. Note that the moving operation may cause the change of the partition size. To ensure that the partition size is fixed, each labelled VM should satisfy the balance constraint as follows:

$$r \cdot |U| - 1 \leq |A| \leq r \cdot |U| + 1, \tag{29}$$

where $|A| + |B| = |U|$, and $r = \frac{|A|}{|A|+|B|}$.

Algorithm 1. Minimizing Communication Cost

```
Input: The initial distribution of VMs $G = (V, E)$, and the initial
       partitions of local servers $A_0$ and cloud $B_0$ with fixed
       size $|V_{A_0}|$ and $|V_{B_0}|$ respectively.
Output: The partition $A$ and $B$ of size $|V_A|$ and $|V_B|$ that
        approximately minimize the communication
        cost between the two partitions.
 1: repeat
 2:   compute the gain for all VMs;
 3:   $c = 1$;
 4:   repeat
 5:     select $v_c$ that has maximum gain $g[c]$ from unlocked
        VMs and satisfies the balance and SLA contract;
 6:     if no such VM then
 7:       break;
 8:     end if
 9:     update gains of adjacent VMs with $v_c$;
10:     lock VM $v_c$;
11:     $c = c + 1$;
12:   until all VMs are locked
13:   find $x$ that maximize $g_{max} = \sum_{i=1}^{x} g[c]$;
14:   if $g_{max} > 0$ then
15:     move $v_1, v_2, \ldots, v_x$ VMs to the other partition;
16:     unlock all VMs;
17:     reset $g$, $g_{max}$;
18:   end if
19: until $g_{max} \leq 0$
```

The time complexity of Algorithm 1 is $O(n)$, and it only needs a very small number of passes to converge leading to a fast approximate algorithm.

Algorithm 2. Online Dynamic Provision Algorithm

```
Input: VM prices, local servers state, request workload.
Output: The provision strategy
 1: for each coarse-grained slot $T$ do
 2:   select the prime sub-queue;
 3:   solve linear programming problem P1;
 4:   for each fine-graind slot $\tau_i$ do
 5:     select the prime sub-queue;
 6:     solve linear programming problem P2;
 7:     algorithm1;
 8:     update queue state;
 9:   end for
10:   update queue state;
11: end for
```

Finally, we can now formally describe ODPA in Algorithm 2. ODPA involves solving two linear programming problems and Algorithm 1 with linear time complexity. Hence, it meets the efficiency requirement in real-world online application.

5 Experimental Study with Real-World Data Set

To further study the performance of ODPA, we conduct a set of experimental study with a large real-world data set.

The VM price data are recorded by tracking the prices of the prevalent IaaS cloud, the Amazon EC2 [4]. We wrote a python script to automatically record the prices of the three types of VMs, i.e., the reserved VMs, the on-demand VMs, and the on-spot VMs in each hour for one month from the Amazon EC2's official website. Fig. 2 shows the on-spot VM prices. The user request data are based on the real cloud request log RICC [34], which is a computing platform with various job running records. We mapped the request in every different months with the one-month VM prices from Amazon, and did one experiment for the data of every month. The results are averaged. We believe that in this way with real-life data sets, we can well capture the performance of the algorithm objectively. Fig. 3 plots the request data sampled for one month. We can see from the two figures that both data are highly dynamic, which is a real-world challenge to VM provision algorithm like ODPA.

We first study the optimization performance of ODPA via comparing it with two benchmark methods. The first benchmark method is one based on a conventional notion that schedules the requests immediately and purchases the

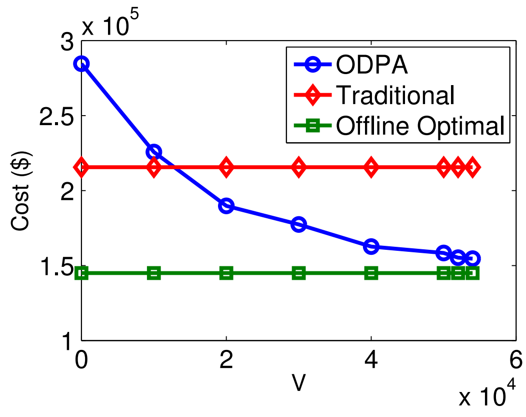


Fig. 4. Total cost with various V.

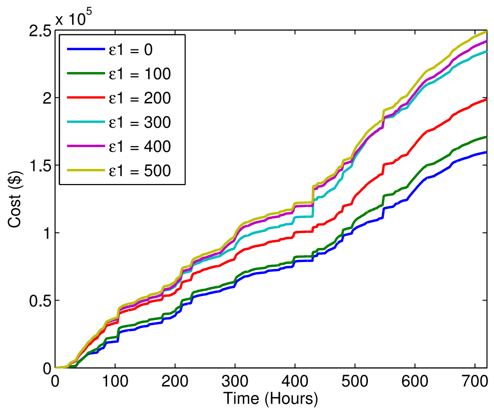


Fig. 6. The cost with various size of ϵ_1 ($V = 50{,}000$).

lowest price VMs to serve the requests regardless the changes of prices. We name it the *conventional* method, and this method is a commonly-used online algorithm.

Another benchmark method assumes that there is oracle and all the future information is perfectly known across the entire time horizon and achieves the ideal solution. We name it an *ideal offline* method. Note that the ideal offline method is an idealized best method, which is not feasible in reality since the future user request and IaaS price information cannot be known beforehand.

Detailed comparisons of ODPA with the two benchmark algorithms are provided in Section 5.1. Section 5.2 futher examines the delay property of ODPA and the trade-off between the delay and cost to verify the theoretical analysis in real-world scenarios.

5.1 Cost Saving of ODPA

First we study how ODPA perform in terms of cost minimization. According to the previous analysis, V and ϵ_i are two major parameters that influence the cost minimization. Moreover, the cost may vary with different proportions of the VMs in the local servers and public IaaS cloud. We evaluate these factors in what follows.

Fig. 4 shows the impact of V by setting it in range [0, 54,000]. We can see from Fig. 4 that the total cost decreases as V increases. This is consistent with our theoretical analysis. In particular, when V increases, the performance of ODPA gets closer to the ideal offline method. Fig. 5 further compares ODPA with the conventional method. It shows that ODPA can reduce up to 30 percent cost saving when V is large.

We then evaluate the impact of parameter ϵ_i. Note that it is not necessary to verify each ϵ_i since each pair of ϵ_i and ϵ_j can be converted to the other based on Theorem 1.

Therefore, without loss of generality, we considers only sub-queue 1 with its corresponding parameter ϵ_1, and study its impact on cost. Fig. 6 shows the results, where we can observe that the cost increases with the growing of ϵ_1.

Finally, the proportion of the requests handled by local servers can influence the cost, since renting less IaaS VMs can reduce the cost more. Fig. 7 shows that the cost decreases as the size of the local servers grows, in which the size is set at different levels of average workload.

When V is set large enough, e.g., $V = 50{,}000$, the performance of ODPA can closely approach the ideal offline method. Fig. 8 shows that even when the local servers handles most of the requests, ODPA is still able to reduce the cost by at least around 10 percent compared with the conventional method.

5.2 Delay Property of ODPA

Based on Theorem 3, there is a trade-off between cost and delay with ODPA. The delay is influenced by the parameter V and ϵ_i according to Lemma 2. Fig. 10 shows the impact of these factors on the delay.

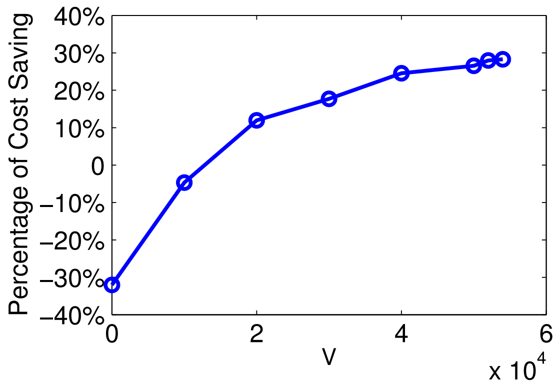


Fig. 5. Impact of Parameter V ($\epsilon_1 = 1$).

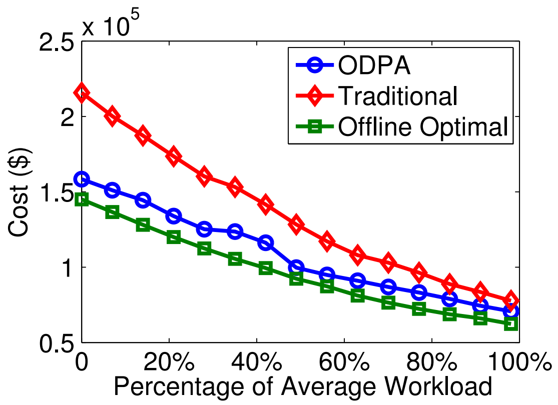


Fig. 7. Total cost with various size of local servers.

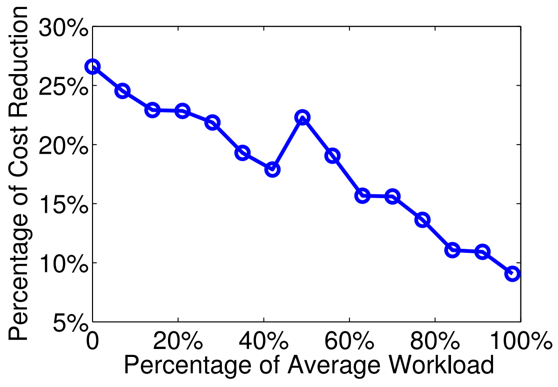


Fig. 8. Proportion of cost reduction.

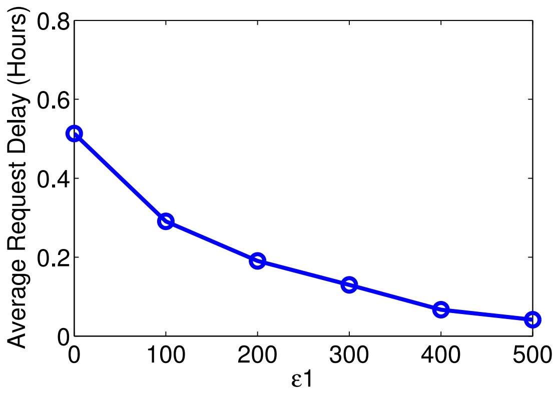


Fig. 10. Impact of ϵ_1 ($V = 50{,}000$).

We measure the delay as the average request delay, i.e., the average queueing time of the requests. The delay caused by parameter V is illustrated in Fig. 9, suggesting that larger V incurs larger delay. Also, the delay decreases when the parameter ϵ_1 increases, as shown in Fig. 10.

Together with Figs. 4 and 6, we can see that there is a trade-off between cost and delay when tuning the parameter V and ϵ_i. The experimental results are consistent with the theoretical analysis. ODPA can successfully save cost by exploiting the delay property. In practice, to meet the delay requirement, one can tune the parameter V, ϵ_i based on the Lemmas 1 and 2.

6 Conclusion

This paper investigates how to optimize the monetary cost of purchasing cloud VMs for the hybrid cloud computing paradigm. Our work assumes an arbitrary request arriving probability and no accurate *a priori* knowledge of VM prices in the public cloud. We specifically tailor a theoretical model based on Lyapunov Optimization framework according to the real-world challenges of this problem. We then develop an method to minimize the time average cost with an online dynamic allocation algorithm. Both the theoretical analysis and the experimental study based on real-world data trace demonstrate the advantages of the algorithm. The evaluation shows that the online dynamic provision algorithm can achieve much lower cost than the conventional method and approach the ideal offline optimal method closely.

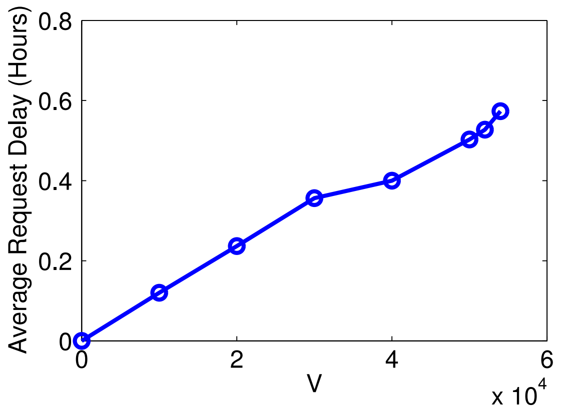


Fig. 9. Impact of V ($\epsilon_1 = 1$).

Acknowledgments

This work was supported by the National Basic Research Program of China under 973 Project No. 2014CB347701, the National Natural Science Foundation of China under Project Nos. 61100077 and 61332010, the Program for New Century Excellent Talents in University under Project No. NCET-11-0113, the Shenzhen Basic Research Program under Project No. JCYJ20120619152636275, and the Research Grants Council of the Hong Kong Special Administrative Region, China under Project No. CUHK 415113. This work has presented in part in IEEE ICWS 2014. Yangfan Zhou is the corresponding author.

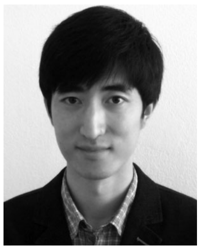
References

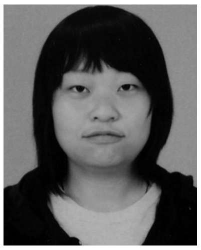
[1] (2014). “Opentext,” [Online]. Available: http://www.skytap.com/downloads/case-studies/skytap_casestudy_opentext.pdf
[2] (2014). “Oxford university,” [Online]. Available: http://www.vmware.com/a/customers/solution/10?sort=a&next=21
[3] (2014). “Sega,” [Online]. Available: http://www.vmware.com/a/customers/solution/10?sort=a&next=41
[4] (2014). “Amazon ec2,” [Online]. Available: http://aws.amazon.com/ec2
[5] (2014). “Vmware vcloud,” [Online]. Available: http://www.vmware.com/products/vcloud-hybrid-service/
[6] (2014). “Zynga,” [Online]. Available: http://zynga.com/
[7] (2014). “Uber,” [Online]. Available: https://www.uber.com/
[8] D. Ardagna, B. Panicucci, and M. Passacantando, “A game theoretic formulation of the service provisioning problem in cloud systems,” in *Proc. 20th Int. Conf. World Wide Web*, 2011, pp. 177–186.
[9] T. Guo, U. Sharma, T. Wood, S. Sahu, and P. Shenoy, “Seagull: intelligent cloud bursting for enterprise applications,” in *Proc. USENIX Annu. Tech. Conf.*, 2012, p. 33.
[10] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource Allocation and Cross-Layer Control in Wireless Networks*. Delft, The Netherlands, Now Publishers, 2006.
[11] D. Ardagna, B. Panicucci, and M. Passacantando, “Generalized nash equilibria for the service provisioning problem in cloud systems,” *IEEE Trans. Serv. Comput.*, vol. 6, no. 4, pp. 429–442, Oct.–Dec., 2012.
[12] S. Chaisiri, B.-S. Lee, and D. Niyato, “Optimization of resource provisioning cost in cloud computing,” *IEEE Trans. Serv. Comput.*, vol. 5, no. 2, pp. 164–177, Apr.–Jun. 2012.
[13] S. Chaisiri, R. Kaewpuang, B.-S. Lee, and D. Niyato, “Cost minimization for provisioning virtual servers in amazon elastic compute cloud,” in *Proc. IEEE Int. Symp. Model., Anal. Simulation Comput. Telecommun. Syst.*, 2011, pp. 85–95.
[14] T. Hacker and K. Mahadik, “Flexible resource allocation for reliable virtual cluster computing systems,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2011, pp. 1–12.
[15] B. Palanisamy, A. Singh, and B. Langston, “Cura: A cost-optimized model for mapreduce in a cloud,” in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, May 2013, pp. 1275–1286.

[16] K. Chard, K. Bubendorfer, and P. Komisarczuk, "High occupancy resource allocation for grid and cloud systems, a study with drive," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput.*, 2010, pp. 73–84.
[17] (2014). "Ibm hybrid cloud solution," [Online]. Available: http://www-01.ibm.com/software/tivoli/products/hybrid-cloud/
[18] (2014). "Cloudswitch," [Online]. Available: http://www.cloudswitch.com/
[19] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam, "To move or not to move: The economics of cloud computing," in *Proc. 3rd USENIX Conf. Hot Topics Cloud Comput.*, 2011, p. 5.
[20] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 243–254, 2010.
[21] (2014). "Hp cloud," [Online]. Available: https://www.hpcloud.com/solutions/bursting
[22] (2014). "Cisco business briefing document," [Online]. Available: https://www.cisco.com/en/US/solutions/collateral/ns341/ns991/ns995/IaaS_BDM_WP.pdf
[23] M. J. Neely, "Energy optimal control for time-varying wireless networks," *IEEE Trans. Inf. Theory*, vol. 52, no. 7, pp. 2915–2934, Jul. 2006.
[24] M. M. Amble, P. Parag, S. Shakkottai, and L. Ying, "Content-aware caching and traffic management in content distribution networks," in *Proc. IEEE INFOCOM*, 2011, pp. 2858–2866.
[25] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Proc. IEEE Netw. Oper. Manage. Symp.*, 2010, pp. 479–486.
[26] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data centers power reduction: A two time scale approach for delay tolerant workloads," in *Proc. IEEE INFOCOM*, 2012, pp. 1431–1439.
[27] W. Deng, F. Liu, H. Jin, and C. Wu, "SmartDPSS: Cost-minimizing multi-source power supply for datacenters with arbitrary demand," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 420–429.
[28] "Google compute engine," http://cloud.google.com/products/compute-engine/
[29] M. J. Neely, "Opportunistic scheduling with worst case delay guarantee in single and multi-hop network," in *Proc. INFOCOM*, 2011, pp. 1728–1736.
[30] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource Allocation and Cross-Layer Control in Wireless Networks*. Delft, The Netherlands: Now Publishers, 2006.
[31] M. J. Neely, A. S. Tehrani, and A. G. Dimakis, "Efficient algorithms for renewable energy allocation to delay tolerant consumers," in *Proc. 1st IEEE Int. Conf. Smart Grid Commun.*, 2010, pp. 549–554.
[32] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete graph problems," *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976.
[33] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Conf. Des. Autom.*, 1982.
[34] "The ricc log," [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html

Song Li received the BS degree from the School of Computer Science, Fudan University, Shanghai, China, in 2012. He is currently working toward the MS degree at the School of Computer Science, Fudan University. His research interests include cloud services and the performance of data center network.

Yangfan Zhou received the BSc degree from Peking University in 2000, and the MPhil and PhD degrees from The Chinese University of Hong Kong in 2006 and 2009, respectively. He is currently with Fudan University. His research interests include distributed computing and networking, particularly in mobile computing, cloud computing, and the Internet of Things. Before joining Fudan, he was a research staff member with The Chinese University of Hong Kong from 2009 to 2014. He has also been working as an engineer in information technology industry for many years, where he is now also active in technology consulting. He is a member of the IEEE.


Lei Jiao received the BSc and MSc degrees from Northwestern Polytechnical University, Xi'an, China, in 2007 and 2010, respectively, and the PhD degree from University of Göttingen, Göttingen, Germany, in 2014, all in computer science. Prior to his PhD study, he was a researcher with IBM Research, Beijing, China. His research interests span computer networks and distributed systems, with a recent focus on performance modeling, analysis, optimization, and evaluation.


Xinya Yan received the BS degree from Zhejiang University of Technology, Hangzhou, China, in 2012. She is currently working toward the MS degree at the School of Computer Science, Fudan University. Her current research interests include cloud resource provisioning and cloud pricing.

Xin Wang received the BS degree in information theory and MS degree in communication and electronic systems from Xidian University, China, in 1994 and 1997, respectively. He received the PhD degree in computer science from Shizuoka University, Japan, in 2002. In 1995 and 1998, he was working on China's pioneering telecom-level video conferencing systems and DVB-S systems with Huawei Inc., Shenzhen, China. He is currently a professor at Fudan University, Shanghai, China. His research interests include quality of network service, next-generation network architecture, mobile Internet and network coding. He is a member of CCF, IEEE, and ACM.

Michael Rung-Tsong Lyu received the BS degree in electrical engineering from National Taiwan University in 1981, the MS degree in computer engineering from the University of California, Santa Barbara, in 1985, and the PhD degree in computer science from the University of California, Los Angeles, in 1988. He is a professor at the Computer Science and Engineering Department of the Chinese University of Hong Kong. His research interests include software reliability engineering, software fault tolerance, distributed systems, data mining, social networks, machine learning, multimedia information retrieval, and mobile networks, where he has published over 450 papers. He has been an associate editor of the *IEEE Transactions on Reliability*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Services Computing*, *IEEE Access*, and *Wiley Software Testing, Verification & Reliability Journal*. He is a fellow of the IEEE and AAAS, and a Croucher Senior Fellow. He received the IEEE Reliability Society 2010 Engineer of the Year Award.