Delay-aware Cost Optimization for Dynamic Resource Provisioning in Hybrid Clouds

Song Li[*], Yangfan Zhou[†], Lei Jiao[‡], Xinya Yan[*], Xin Wang[*], Michael R. Lyu[†]
[*]School of Computer Science, Fudan University, China
[†]Shenzhen Research Institute, The Chinese University of Hong Kong
[‡]University of Göttingen, Germany

***Abstract*—Hybrid cloud computing paradigm has recently be widely advocated, where Software-as-a-Service (SaaS) providers can extend their local services into the public clouds seamlessly. In this way, dynamic user request workload to a SaaS can be elegantly handled with the rented computing capacity in public cloud. However, although a hybrid cloud may save cost compared with the private cloud, it still introduces considerable renting cost and communication cost. How to optimize such an operational cost becomes one major concern for the SaaS providers to adopt such a 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 model by theoretically analyzing the problem with a Lyapunov optimization framework, and accordingly providing 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 experimental study based on a set of real-world data, and the results confirm that our algorithm can work well in reducing the cost.**

I. Introduction

Cloud computing has surged into popularity in the IT industry as it can provide a cost-effective elastic solution to computing resource provisioning. Recently, 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.

More and more leading IaaS cloud solutions (*e.g.*, Amazon EC2[1] and VMWare vCloud [2]) 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 back to their own data centers, because of the high expenditure of renting VMs [3][4]. 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 daunting task. Most importantly, 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. But the user requests are highly dynamic in nature, which cannot be known and 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 [1]. All these dynamic factors can have a great impact on the cost, and hence bring great challenges to cost minimization.

However, existing approaches (*e.g.*, [5] and [6]) 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 the future user requests and the VM prices, and also takes the communication cost into considerations. Via modeling the problem with Lyapunov optimization framework [7], 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.

978-1-4799-5054-6/14 $31.00 © 2014 IEEE
DOI 10.1109/ICWS.2014.35

The rest of this paper is organized as follows. Section II introduces the system model and formulate the cost minimization problem. Section III 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 IV and provide our our experimental results with real-world data sets in Section V. Section VI discusses the related work and the paper is concluded in Section VII.

II. System model and problem formulation

A. System model

We consider an SaaS provider operates with a small local data center (or *local servers* in our following discussions) and it can extend its service capacity via renting VMs of a public IaaS cloud. 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 [1][8]. 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).

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.

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. Each coarse-grained interval can be further divided into many small decision slots with length T', where the numbers of on-demand VMs and on-spot VMs can be reset at the beginning of each T' interval (without loss of generality, T=mT' where m is a natural number. We name such a decision slots with length T' a *fine-grained* decision time slot. 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.

B. User requests and SaaS service model

Consider the users access the SaaS provider with an arrival rate $\lambda(\tau)$, where τ is a natural number denoting each of the fine-grained decision time slot. Note that $\lambda(\tau)$ 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 period of time d^{max} before it is scheduled. Consider the requests can be stored in a queue denoted by $Q(\tau)$ in each fine-grained decision time slot. In each find-grained interval, the user requests in the queue are served with the rate $\mu(\tau)$. Therefore, the queue has the following update equation:

$$Q(\tau+1) = \max\{Q(\tau)-\mu(\tau),0\}+\lambda(\tau). \qquad (1)$$

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 in a coarse-grained decision interval is :

$$\mu(\tau) = (L(\tau)+R(\tau)+D(\tau)+S(\tau))\cdot c, \qquad (2)$$

where $L(\tau)$, $R(\tau)$, $D(\tau)$, and $S(\tau)$ denote the number of the local server VMs, reserved VMs, on-demand VMs, and on-spot VMs in time slot τ, 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.

C. 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.

$$Cost(\tau) = \phi + \frac{R(t)}{T}P_r(t) + D(\tau)P_d(\tau) + S(\tau)P_s(\tau) + M(\tau), \qquad (3)$$

where ϕ is the cost of running the local servers, which is a constant without loss of generality, and $M(\tau)$ denotes the communication cost between the VMs in the local servers and those in the cloud. We consider the real-world case that the allocated VMs may communicate with each other. In practice, the communications among the local servers or in the cloud per se are free, while communications between the local servers and the IaaS cloud will be charged. $R(t)$, $D(\tau)$, and $S(\tau)$ are the numbers of the reserved, on-demand, and on-spot VMs in time slot τ respectively, and $P_r(t)$, $P_d(\tau)$, and $P_s(\tau)$ are their prices respectively. t denotes the first fine-grained time slot of a coarse-grained time interval. Note that the number of the reserved VMs $R(t)$ and the price $P_r(t)$ are decided at t and are not variable during the entire coarse-grained interval.

D. 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 in the interval, such that the time-average expecting cost is minimized. Formally,

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

s.t.

$$0 \le L(\tau) \le L^{max}, 0 \le R(\tau) \le R^{max}, \\ 0 \le D(\tau) \le D^{max}, 0 \le S(\tau) \le S^{max}. \tag{5}$$

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

$$\forall \tau, Q(\tau) < Q^{max}, \bar{Q} < \infty \tag{7}$$

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

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.

Equation (6): In each time slot τ, the provisioned VMs must satisfy the capability requirement $W(\tau)$ ensuring that the requests can be served.

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 $Q(\tau)$ should satisfy this equation, where Q^{max} is the maximum backlog. The section III.B 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 $\beta(\tau)$ should be bounded.

III. 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. Based on these considerations, we then build the Lyapunov optimization model [7] and convert the cost minimization problem into a solvable one. We design an online dynamic provision algorithm (ODPA) 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.

A. 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.

We map the queue $Q(\tau)$ into a set of *sub-queues* $Q_i(\tau)$, $i = 1, 2, ..., n$. Each sub-queue $Q_i(\tau)$ is corresponding to a certain type of resource (suppose there are n types of resources). When a request enters $Q(\tau)$, it also enters every sub-queue $Q_i(\tau)$. When the request is queueing in $Q(\tau)$, we consider it is queueing for each type of resource in $Q_i(\tau)$. It leaves $Q(\tau)$ only if it can be served by the bottleneck resource k and leave the corresponding sub-queue $Q_k(\tau)$. In this case, the request also leaves the other sub-queues.

Therefore, the service rate of $Q(\tau)$ (as well as that of each sub-queue $Q_i(\tau)$) is determined by the service rate of the bottleneck sub-queue $Q_k(\tau)$. 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 type of bottleneck computing resource changes. We thus model the heterogeneity of user requests with such sub-queues.

B. Delay-aware Virtual Queue

User request has a deadline d^{max} to meet after it enters $Q(\tau)$. To handle this situation, we apply *ϵ-persistent service queue* technique [9] to bound the worst-case delay of the dequeuing operations, so as to ensure that a request can be served before its deadline.

Let $Z_i(\tau)$ denote a queue associated with $Q_i(\tau)$. Its update equation is as follows.

$$Z_i(\tau+1) = \max\{Z_i(\tau) - \mu_i(\tau) + \epsilon_i 1_{Q_i(\tau)>0}, 0\}, \tag{9}$$

where $1_{Q_i(\tau)>0}$ is an indicator function taking the value 1 if $Q_i(\tau) > 0$, and 0 otherwise. ϵ_i is a parameter that controls the growing rate of the delay-aware virtual queue Z_i, which has an impact on the queueing time of a request (We will discuss the details on ϵ_i later).

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

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

Lemma 1: Given that $Q_i(\tau) \le Q_i^{max}$ and $Z_i(\tau) \le Z_i^{max}$, the user requests have the maximum delay of d_i^{max}, in which:

$$d_i^{max} = \lceil (Q_i^{max} + Z_i^{max})/\epsilon_i \rceil \tag{10}$$

Proof: The proof is omitted due to space limitation. It can be found in [10] ■

For parameters ϵ_i and ϵ_j for sub-queues i and j respectively, we have the following relationship.

Theorem 1: Given that $\forall i$, $Q_i(\tau) \leq Q_i^{max}$ and $Z_i(\tau) \leq Z_i^{max}$, $\forall i, j$, $i \neq j$, the following equation holds.

$$\frac{\epsilon_i}{\epsilon_j} = \frac{Q_i^{max} + Z_i^{max}}{Q_j^{max} + Z_j^{max}} \tag{11}$$

Proof: The proof can be also found in [10] ■

C. Lyapunov Optimization

At each time slot τ, we select and control the prime queue $Q_k(\tau)$ (*i.e.*, k is the bottleneck resource index), and other sub-queues are also controlled following the bottleneck one.

Let $\Theta(\tau)$ denote the vector $[Q_k(\tau), Z_k(\tau)]$. We define a *Lyapunov function* as follows.

$$L(\Theta(\tau)) \triangleq \frac{1}{2}[Q_k(\tau)^2 + Z_k(\tau)^2] \tag{12}$$

The 1-slot conditional *Lyapunov drift* is defined as:

$$\Delta(\Theta(\tau)) \triangleq \mathbb{E}\{L(\Theta(\tau+1)) - L(\Theta(\tau))|\Theta(\tau)\} \tag{13}$$

Following the *drift-plus-penalty* algorithm[11], 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:

$$\Delta(\Theta(\tau)) + V\mathbb{E}\{Cost(\tau)|\Theta(\tau)\}, \tag{14}$$

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 $0 \leq \lambda_k \leq \lambda_k^{max}$, $0 \leq \mu_k \leq \mu_k^{max}$, and $Q_k(\tau) < Q_k^{max}$, the drift-plus-penalty expression satisfies:

$$\begin{aligned} \Delta(\Theta(\tau)) &+ V\mathbb{E}\{Cost(\tau)|\Theta(\tau)\} \\ &\leq B + V\mathbb{E}\{Cost(\tau)|\Theta(\tau)\} \\ &+ Q_k(t)\mathbb{E}\{(\lambda_k(\tau) - \mu_k(\tau))|\Theta(\tau)\} \\ &+ Z_k(t)\mathbb{E}\{(\epsilon_k - \mu_k(\tau))|\Theta(\tau)\}, \end{aligned} \tag{15}$$

where B is:

$$B = \frac{1}{2}\max\{(\mu_k^{max})^2, \epsilon_k^2\} + \frac{1}{2}((\mu_k^{max})^2 + (\lambda_k^{max})^2) \tag{16}$$

Proof: The proof can be also found in [10] ■

Rewriting Equation (15) by substituting $\mu(\tau)$ and Cost(τ) in Equations (2) and (3), we turn the drift-plus-penalty upper bound minimization problem into the following problem **P0**.

$$\begin{aligned} \min\ & \mathbb{E}\{\frac{R(t)}{T}(VP_r(t) - Q_k(t) - Z_k(t))|\Theta(t)\} \\ &+ \mathbb{E}\{D(\tau)(VP_d(\tau) - Q_k(\tau) - Z_k(\tau))|\Theta(\tau)\} \\ &+ \mathbb{E}\{S(\tau)(VP_s(\tau) - Q_k(\tau) - Z_k(\tau))|\Theta(\tau)\} \\ &+ \mathbb{E}\{L(\tau)(-Q_k(\tau) - Z_k(\tau)) + VM(\tau)|\Theta(\tau)\} \end{aligned} \tag{17}$$

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

where $\tau \in [t,\ t+T\text{-}1]$, and t is the beginning fine-grained time slot of the coarse-grained interval T.

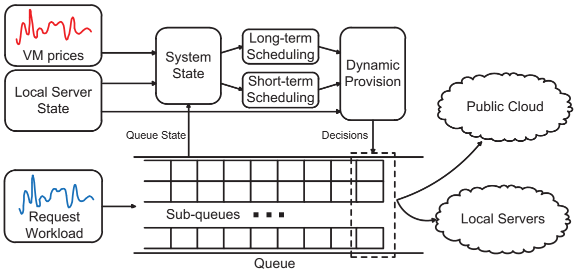


Fig. 1. Architecture of the system and ODPA.

D. Online Dynamic Provision Algorithm (ODPA)

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. Figure 1 overviews the framework of ODPA. What follows elaborates these three key parts.

1) Long-term scheduling: The SaaS provider optimizes its cost at the beginning of each coarse-grained interval with length T. Let t denote the *first* fine-grained time slot of the coarse-grained interval. The current system state then includes $Q_k(t)$, $Z_k(t)$, $\lambda_k(t)$, and the reserved VM pricing information $P_r(t)$.

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

$$\begin{aligned} \min_{R(t)}\ & \frac{R(t)}{T}(VP_r(t) - Q_k(t) - Z_k(t)) \\ \text{s.t. } & \text{Equation } (5)(6)(7)(8) \end{aligned} \tag{18}$$

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

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 of each coarse-grained interval. The system state includes $Q_k(\tau)$, $Z_k(\tau)$, $\lambda_k(\tau)$, the pricing information $P_d(\tau)$, $P_s(\tau)$, 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**.

$$\begin{aligned}\min_{D(\tau),S(\tau),L(\tau)} \quad & D(\tau)(VP_d(\tau) - Q_k(\tau) - Z_k(\tau)) \\ & + S(\tau)(VP_s(\tau) - Q_k(\tau) - Z_k(\tau)) \\ & + L(\tau)(-Q_k(\tau) - Z_k(\tau)) \\ & \text{s.t. Equations } (5)(6)(7)(8)\end{aligned} \tag{19}$$

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

3) Dynamic provisioning: We now consider the communication cost $M(\tau)$. 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. 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.

$$\begin{aligned}\min \quad & V \cdot M(\tau) \\ \text{s.t. } & \text{constraint}(8)\end{aligned} \tag{20}$$

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

Theorem 3: 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 k, 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 $|\{e \mid \forall e \in E'\}|$ is equal to k 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 [12]. 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 +∞ 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 k in G. S is also a cut set of size k in G', *i.e.*, $S = S'$, since edges of new added clique have +∞ weights and they cannot be in the cut set of G'. Conversely, suppose $S' \subseteq E'$ is a cut set of size k 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 k in G if and only if S' is a cut set of size k 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. ■

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 $k_A$ and $k_B$ respectively.
Output: The partition $A$ and $B$ of size $k_A$ and $k_B$ that
    approximately minimize the communication cost between
    the two partitions.
repeat
    compute the gain for all VMs;
    $i = 1$;
    repeat
        select $v_i$ that has maximum gain $g[i]$ from unlocked
        VMs and satisfies the balance and SLA contract;
        if no such VM then
            break;
        end if
        update gains of adjacent VMs with $v_i$;
        lock VM $v_i$;
        $i = i + 1$;
    until all VMs are locked
    find k that maximize $g_{max} = \sum_{i=1}^{k} g[i]$;
    if $g_{max} > 0$ then
        move $v_1$, $v_2$, ..., $v_k$ VMs to the other partition;
        unlock all VMs;
        reset $g$, $g_{max}$;
    end if
until $g_{max} \leq 0$
```

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 [13] 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 updates the adjacent VMs until all VMs selected. It then executes the first k 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.

$$g(i) = E_i - I_i - P_i, \tag{21}$$

where E_i and I_i are the external cost and internal cost of VM i 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_i 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.

$$r \cdot |U| - 1 \leq |A| \leq r \cdot |U| + 1, \tag{22}$$

Algorithm 2 Online Dynamic Provision Algorithm (ODPA)

```
Input: VM prices, local servers state, request workload.
Output: The provision strategy
for each coarse-grained slot $T$ do
    select the prime sub-queue;
    solve linear programming problem P1;
    for each fine-graind slot $\tau$ do
        select the prime sub-queue;
        solve linear programming problem P2;
        algorithm1;
        update queue state;
    end for
    update queue state;
end for
```

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

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.

Next, we will first theoretically analyze the algorithm performance in Section IV, and then build a simulation to have further studies.

IV. Performance Analysis

We now theoretically analyze the performance of ODPA. 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 $0 < P_r(t) \leq P_r^{max}$, $0 < P_d(\tau) \leq P_d^{max}$, $0 < P_s(\tau) \leq P_s^{max}$, $0 \leq \epsilon_i \leq \mu_i^{max}$, and $V > 0$, we have:

(i). The length of Q_i and Z_i can be bounded by the following:

$$\begin{aligned} Q_i^{max} &= V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_i^{max}, \\ Z_i^{max} &= V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \epsilon_i \end{aligned} \tag{23}$$

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

$$d_i^{max} = \lceil \frac{2V\max\{P_r^{max}, P_d^{max}, P_s^{max}\} + \lambda_i^{max} + \epsilon_i}{\epsilon_i} \rceil \tag{24}$$

(iii). The worst case delay d_i^{max} and d_j^{max} of sub-queue i and j are equal, $\forall i, j$, $i \neq j$:

$$d_i^{max} = d_j^{max} \tag{25}$$

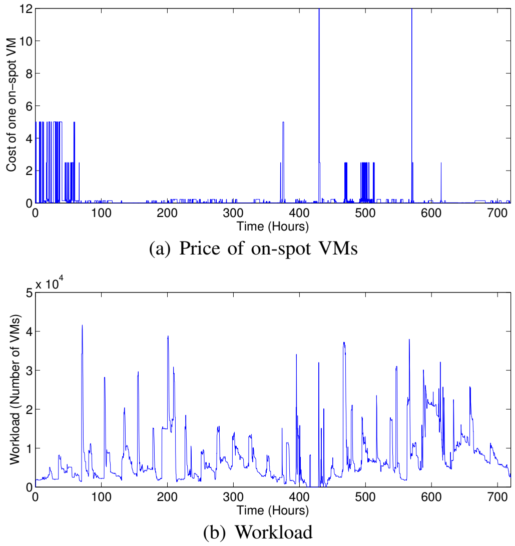
Proof: The proof can be also found in [10] ■

Based on this lemma, we derive the following theorem.

Theorem 4: ODPA can bound the time average cost as follows.

$$lim_{t\rightarrow\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\mathbb{E}[Cost(\tau)] \leq \frac{1}{t}\sum_{\tau=0}^{t-1}c_\tau^* + \frac{B}{V}, \tag{26}$$

Proof: The proof can be also found in [10] ■



(a) Price of on-spot VMs

(b) Workload

Fig. 2. Real traces

, where $\frac{1}{t}\sum_{\tau=0}^{t-1}c_\tau^*$ is the minimum possible time average cost if the future information of subsequent slots is known, B is defined in Equation (16).

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.

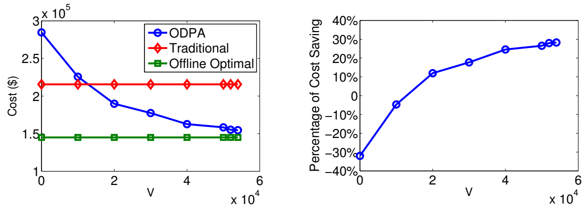
V. Experimental Study With Real-world Data Set

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 [1]. We program to record the prices of the reserved VMs, the on-demand VMs, and the on-spot VMs for one month. The user request data are based on the real cloud request log RICC [14]. Figures 2(a) and 2(b) plot the on-spot VM prices and the request data. We can see 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 lowest price VMs to serve the requests regardless the changes of prices. We name it the *conventional* method.

Another benchmark method assumes that 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.



(a) Total cost with various V (b) Proportion of cost reduction

Fig. 3. Impact of parameter V ($\epsilon_1 = 1$)

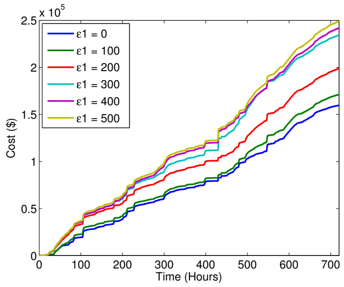


Fig. 4. The cost with various size of ϵ_1 ($V = 50000$)

Detailed comparisons of ODPA with the two benchmark algorithms are provided in Section V-A. Section V-B 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.

A. Cost saving of ODPA

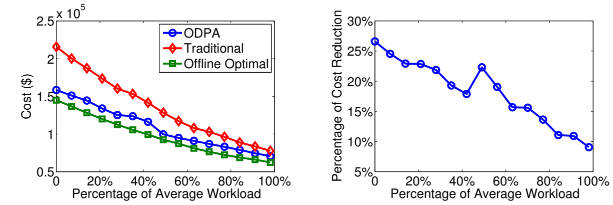
First we study how ODPA performs in terms of cost minimization. According to the previous analysis, V and ϵ_i are two major parameters that influence the cost minimization. Moreover, the cost may vary with different proportions of the VMs in the local servers and public IaaS cloud. We evaluate these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

Figure 3 shows the impact of V by setting it in range [0, 54000]. We can see from Figure 3(a) 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. Figure 3(b) further compares ODPA with the conventional method. It shows that ODPA can reduce up to 30% 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.

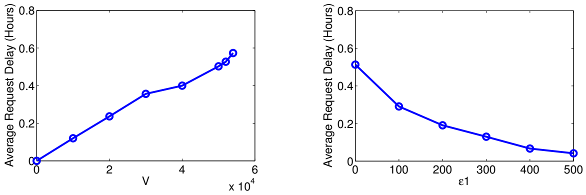
Therefore, without loss of generality, we considers only subqueue 1 with its corresponding parameter ϵ_1, and study its impact on cost. Figure 4 shows the results, where we can observe that the cost increases with the growing of ϵ_1.

Finally, the proportion of the requests handled by local servers can influence the cost, since renting less IaaS VMs can reduce the cost more. Figure 5(a) 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 enouth, *e.g.*, $V = 50000$, the performance of ODPA can closely approach the ideal offline method. Figure 5(b) 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% compared with the conventional method.



(a) Total cost with various size of local servers (b) Proportion of cost reduction

Fig. 5. Impact of proportion of workload of local servers ($V = 50000$, $\epsilon_1 = 1$). The size of local servers is set at different levels of average workload.



(a) Impact of V ($\epsilon_1 = 1$) (b) Impact of ϵ_1 ($V = 50000$)

Fig. 6. Delay of ODPA.

B. Delay property of ODPA

Based on Theorem 4, 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. Figure 6 shows the impact of these factors on the delay.

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 Figure 6(a), suggesting that larger V incurs larger delay. Also, the delay decreases when the parameter ϵ_1 increases, as shown in Figure 6(b).

Together with Figures 3(a) and 4, 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 Lemma 1 and Lemma 2.

VI. 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 [5][6][15][16][17][18]. In [5] and [15], 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. [5] considers the uncertainty of VM prices and user requests in optimizing the resource provisioning cost. [17] designs a deadline-awareness system which optimizes its global resource allocation and reduce the cost by delaying the excution of particular jobs. Moreover, [18] proposes to optimize the provisioning cost based also on a stochastic programming model.

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[1], VMWare vCloud [2], and IBM Hybrid Cloud Solution [19]. Tak *el al.* [20] investigate the economic issues of the application deployment choice in the hybrid cloud. Hajjat *et al.* [21] propose the migration strategy of enterprise application to the cloud that optimizes the benefits and ensures the security policies. Guo *et al.* [6] 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 [22][23].

Finally, Lyapunov optimization technique has been widely adopted in routing and resource allocation in several types of applications. Neely *et al.* [24] establishes a simple Lyapunov drift to achieve both system stability and performance optimization in time varying wireless 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]. 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.

VII. 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.

ACKNOWLEDGEMENT

This work was supported by the National Basic Research Program of China under 973 Project No. 2011CB302603, the National Natural Science Foundation of China under Project No. 61100077, the Program for New Century Excellent Talents in University under Project No. NCET-11-0113, and the Shenzhen Basic Research Program under Project No. JCYJ20120619152636275. Yangfan Zhou is the corresponding author.

REFERENCES

[1] "Amazon EC2," http://aws.amazon.com/ec2.

[2] "VMWare vCloud," http://www.vmware.com/products/vcloud-hybrid-service/.

[3] "Zynga," http://zynga.com/.

[4] "Uber," https://www.uber.com/.

[5] D. Ardagna, B. Panicucci, and M. Passacantando, "A game theoretic formulation of the service provisioning problem in cloud systems," in *Proc. of the 20th ACM WWW*, 2011.

[6] T. Guo, U. Sharma, T. Wood, S. Sahu, and P. Shenoy, "Seagull: intelligent cloud bursting for enterprise applications," in *Proc. of of the USENIX ATC*, 2012.

[7] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource allocation and cross-layer control in wireless networks*. Now Publishers Inc, 2006.

[8] "Google Compute Engine," http://cloud.google.com/products/compute-engine/.

[9] M. J. Neely, "Opportunistic scheduling with worst case delay guarantee in single and multi-hop network," in *Proc. of the INFOCOM*, 2011.

[10] S. Li, Y. Zhou, L. Jiao, X. Yan, X. Wang, and M.R.Lyu, "Delay-aware cost optimization of dynamic resource provisioning in hybrid cloud," Tech. Rep., School of Computer Science, Fudan University. http://www.cse.cuhk.edu.hk/ỹfzhou/tech-report-1402.pdf, Jan. 2014.

[11] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource allocation and cross-layer control in wireless networks*. Now Publishers Inc, 2006.

[12] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete graph problems," *Theoretical Computer Science*, vol. 1, no. 3, pp. 237–267, 1976.

[13] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. of the 19th Conference on Design Automation*, 1982.

[14] "The RICC log," http://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html.

[15] D. Ardagna, B. Panicucci, and M. Passacantando, "Generalized nash equilibria for the service provisioning problem in cloud systems," *IEEE Transactions on Services Computing*, vol. 6, no. 4, 2012.

[16] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

[17] B. Palanisamy, A. Singh, and B. Langston, "Cura: A cost-optimized model for mapreduce in a cloud," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013, pp. 1275–1286.

[18] S. Chaisiri, R. Kaewpuang, B.-S. Lee, and D. Niyato, "Cost minimization for provisioning virtual servers in amazon elastic compute cloud," in *Proc. of the IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, 2011.

[19] "IBM hybrid cloud solution," http://www-01.ibm.com/software/tivoli/products/hybrid-cloud/.

[20] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam, "To move or not to move: The economics of cloud computing," in *Proc. of the 3rd USENIX HotCloud*, 2011.

[21] 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 Computer Communication Review*, vol. 40, no. 4, pp. 243–254, 2010.

[22] "HP Cloud," https://www.hpcloud.com/solutions/bursting.

[23] "Cisco Business Briefing Document," https://www.cisco.com/en/US/solutions/collateral/ns341/ns991/ns995/IaaS_BDM_WP.pdf.

[24] M. J. Neely, "Energy optimal control for time-varying wireless networks," *IEEE Transactions on Information Theory*, vol. 52, no. 7, pp. 2915–2934, 2006.

[25] 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. of IEEE INFOCOM*, 2012, pp. 1431–1439.

[26] W. Deng, F. Liu, H. Jin, and C. Wu, "Smartdpss: Cost-minimizing multi-source power supply for datacenters with arbitrary demand," in *Proc. of ICDCS*, 2013.