Cost-Efficient VM Configuration Algorithm in the Cloud using Mix Scaling Strategy

Li Lu, Jiadi Yu, Yanmin Zhu, Guangtao Xue, Shiyou Qian, Minglu Li
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
Email: {luli_jtu, jiadiyu, yzhu, xue-gt, qshiyou, mlli}@sjtu.edu.cn

***Abstract*—Benefiting from the pay-per-use pricing model of cloud computing, many companies migrate their services and applications from typical expensive infrastructures to the cloud. However, due to fluctuations in the workload of services and applications, making a cost-efficient VM configuration decision in the cloud remains a critical challenge. Even experienced administrators cannot accurately predict the workload in the future. Since the pricing model of cloud provider is convex other than linear that often assumed in past research, instead of typical scaling out strategy. In this paper, we adopt mix scale strategy. Based on this observation, we model an optimization problem aiming to minimize the VM configuration cost under the constraint of migration delay. Taking advantages of Lyapunov optimization techniques, we propose a mix scale online algorithm which achieves more cost-efficiency than that of scale out strategy. Experimental results shows that the mix scale algorithm saves 30.8% and 31.1% cost while controlling migration delay in a tolerable range under different workload respectively.**

I. Introduction

Cloud computing has become a popular technology nowadays. Under the attractive pay-per-use pricing model, users could utilize cloud computing just like using water and electricity [1]. Hence, companies could flexibly manage their infrastructures to save costs. Also, elastic scalability is possible. According to a report [2], Brickfish, a software provider of social media in Chicago, migrated their service from typical rented servers to NaviCloud, and their cost decreased from 700 thousand dollars to 200 thousand per year.

The first step to utilize cloud computing is to assess their applications and predict the amount of requests that may come (i.e., arrival rate), which is the basis of other steps. However, even experienced administrators, cannot do this step very well, which may lead to low-quality services. Further, the corresponding revenue of this company may decrease. Thus, if cloud providers can provide recommendation to users, the efficiency would improve significantly. Many popular cloud platforms, such as Azure [3] and AppEngine [4], also suffer from the difficulty of configurations.

Many researchers design algorithms to make cost-efficient VM configuration decision. Calheiros ed al. [5] design an adaptive VM provisioning algorithm to handle the homogeneous VM configurations problems. Sharma et al. [6] take several provisioning mechanisms into consideration to achieve higher efficiency. Except for these works, Sharrukh et al. [7] [8] design a combinatorial auction-based mechanism for dynamic VM provisioning. Jung et al. [9] propose a system named Mistral, which manage power consumption and performance in host level. In fact, many cloud providers often provide simple strategies to their users, such as AWS Auto Scaling [10]. However, all these work do not take both cost-efficiency and migration delay into consideration the same time.

In VM configuration problem, there are still some complex challenges. First, initial applying application to the cloud properly depends on accurate prediction of future workload, which is difficult. Then, dynamic workload has significant influence on sequent VM configurations. Finally, different from scaling out strategy, the VM configurations are hard to decide in mix scaling strategy. Instead, since migration delay is introduced, the tradeoff between migration delay and VM renting cost should be taken into consideration.

In this paper, we propose an online algorithm for cost-efficient VM configurations using mix scaling strategy. Mix scaling strategy means heterogeneous flavors are used in VM configurations. Since the pricing model of cloud providers is usually convex, mix scaling strategy achieves higher cost-efficiency. The objective of the VM configuration problem is to minimize unit cost, while constraints are to control the delay in a tolerable range and to satisfy performance requirements. Taking advantages of Lyapunov optimization techniques, the VM configuration problem is transformed to a one-slot optimization problem. The one-slot optimization problem minimize both VM configuration cost and migration delay, where a weight can be tuned to change the emphasis. Users could tune the weight in the objective to change their focus on different objectives.

Our main contributions are:

- We formulate the VM configurations problem as an optimization problem, and both migration delay and cost are taken into consideration.
- We design a cost-efficient mix scale online algorithm to transform VM configurations problem to a cost-migration delay tradeoff problem taking advantages of Lyapunov optimization techniques.
- We conduct simulations and demonstrate the efficiency of our proposed algorithm. The result shows that our algorithm would achieve more than 30% cost-efficiency than scaling out strategy.

In the rest of the paper, we first introduce the VM configuration problem, and formulate COst-Migration Delay Tradeoff (COMDT) problem to handle the migration delay (Section

978-1-4673-8999-0/17/$31.00 ©2017 IEEE

II). Then, we transform the COMDT problem to an one-slot optimization problem using Lyapunov optimization technique (Section III). Furthermore, we propose a Cost-Efficient Mix Scale Online algorithm to solve the one-slot problem (Section IV). Next, we conduct simulations to evaluate the performance of our algorithms (Section V). Finally, we make a conclusion (Section VI).

II. Problem Definition

In this section, we describe the formulation of the VM configuration problem and the COst-Migration Delay Tradeoff (COMDT) problem.

A. VM Configuration Problem

Since full-user-customized resources are not beneficial for management, the resources of cloud provider are usually presented in VM form. A cloud provider often provides various types of VMs (also called as flavors), the number of which K is defined by providers. TABLE I shows five different flavors in AWS. For flavor i, users pay c_i for usage in a unit time. Also, different flavors have different performance because of different hardware configuration. We take the maximum service rate as the metric to evaluate the performance of different flavors.

The objective of VM configuration problem is to minimize the cost that users pay for provisioning VMs. To ensure the response time of applications does not exceed a tolerable range, the sum service rate of provisioned VMs should be larger than the arrival rate of application requests. Thus, the VM configuration problem is

$$\begin{aligned} \min \quad & \sum_{i=1}^{K} x_i c_i \\ s.t \quad & \sum_{i=1}^{K} x_i \mu_i \geq \lambda \\ & x_i \in N \quad i = 1, 2, \ldots, K, \end{aligned} \tag{1}$$

where x_i is the number of the provisioned flavor i, c_i and μ_i are the unit cost and maximum service rate of the flavor i respectively, and λ is the arrival rate of requests.

When we introduce the dynamic workload λ and consider heterogeneous flavors in the VM configuration problem, there are several complex situations. The old configuration x^{old} is the VM configuration in the $t-1$ time slot, while the new configuration x^{new} is that in the t time slot (both x^{old} and x^{new} are vectors). There are three situations as follows.

(1) $x^{new} - x^{old} \geq 0$: the old VM configuration cannot satisfy performance requirement. Also, the number of each flavor in the new configuration x_i^{new} is all larger than that in the old configuration x_i^{old}. Therefore, users just provision more VMs of each flavor without migrating any data.

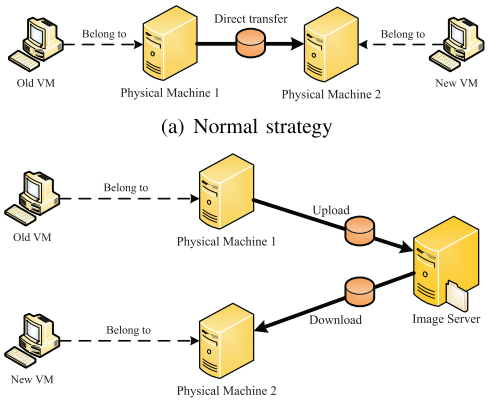
(2) $x^{new} - x^{old} \leq 0$: although the old configuration could satisfy performance requirement, it is not a cost-effective configuration. Also, each flavor in the new configuration x_i^{new} is less than that in the old configuration x_i^{old}, users just shutdown them without migrating any data.

(3) $x^{new} - x^{old} \neq 0$: some flavors in the new configuration x_i^{new} are larger than old one x_i^{old} while others in the new configuration x_j^{new} are less than old one x_j^{old}. It leads to a new problem: old VMs may contain some data for starting new VMs. Therefore, migration of these data becomes necessary, and the migration delay is introduced to the system.

For the first two situations, users just provision more VMs to meet performance demand, or shutdown useless VMs for cost-efficiency. But for the third situation, we should migrate data from old VMs to new VMs, in which the migration delay is not tolerable.

TABLE I
VM Configurations and Prices in AWS

Flavor	Configurations	Price/h	Price/core
m4.large	2 vCPUs, 8G RAM	$0.979	$0.490
m4.xlarge	4 vCPUs, 16G RAM	$1.226	$0.307
m4.2xlarge	8 vCPUs, 32G RAM	$2.553	$0.319
m4.4xlarge	16 vCPUs, 64G RAM	$5.057	$0.316
m4.10xlarge	40 vCPUs, 160G RAM	$12.838	$0.321



(a) Normal strategy

(b) Real strategy

Fig. 1. Different strategy while starting a VM

B. COst-Migration Delay Tradeoff (COMDT) Problem

To tackle with the relationship between the migration delay and VM configuration cost, we formulate a COst-Migration Delay Tradeoff (COMDT in short) problem to minimize the VM configuration cost while constraining migration delay in a tolerable range.

Migration in cloud computing is distinguished into the live migration and shutdown migration. Since the shutdown migration scheme is widely adopted in popular cloud platforms, we just consider the delay of shutdown migration. In the shutdown migration, users first shutdown the old VM, then transfer the VM image to the new physical machine, and finally start the new VM. Intuitively, VM image is directly transferred between two physical machines, as Fig. 1(a) shows. However, in real cloud environments, the migration process is like Fig. 1(b) shows. In shutdown migration, the VM image is first uploaded to the image server, and downloaded to the new machine. Therefore, the migration delay is defined as

$$\alpha = 2\frac{\mathcal{D}}{b} + s, \tag{2}$$

where $\mathcal{D}$ is the image size, b is the bandwidth, and s is the start time of a new VM. Since upload and download bandwidths are the inner bandwidths in the cloud, and there is not significant difference between them, i.e., b is a constant.

In Eq. (2), the image size $\mathcal{D}$ is related to VM configurations between two constitute time slots. Since all old disk images should be uploaded to the image server, the sum transferring image size equals to the volume of old images. Thus, the sum transferring image size $\mathcal{D}$ is defined as

$$\mathcal{D}(t) = d(t) \sum_{x_i(t-1)>x_i(t)} |x_i(t-1) - x_i(t)|, \tag{3}$$

where $d(t)$ is the average image size in the time slot t.

The objective of COMDT problem is to minimize the cost paid by users, and the only difference from original problem is that the cost is related to time. As time goes on, the paying cost would tend to infinite which leads to the fact that the cost is beyond measure. Hence, minimizing time-averaged cost is the objective. On the other hand, the migration delay is inevitable. The migration delay should be constrained in a tolerable range. Similar to the objective, migration delay is also monotonically increasing in terms of time. Thus, the migration delay constraint is also time-averaged. Thus, the COMDT problem is formulated as follows.

$$\min \quad \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i=1}^{K} x_i(t) c_i \tag{4}$$

$$s.t \quad \sum_{i=1}^{K} x_i(t)\mu_i \geq \lambda(t) \qquad \forall t \tag{5}$$

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \alpha(t) \leq \mathcal{T} \tag{6}$$

$$x_i(t) \in N \qquad \forall t, i, \tag{7}$$

where $x_i(t)$ is the VM configuration, $\lambda(t)$ is the workload, $\alpha(t)$ is the generated migration delay in time slot t, and $\mathcal{T}$ is the maximum tolerable delay.

Thus, the COMDT problem is formulated. However, the objective and migration delay constraint are time-averaged, which means typical optimization techniques cannot be adopted to solve the COMDT problem. In the next section, we discuss the method to the COMDT problem.

III. Solution to COMDT Problem

In this section, we describe the solution to the COMDT problem based on Lyapunov optimization techniques [11]. Many researches [12]–[14] adopt Lyapunov optimization techniques to solve time-averaged problems. Taking advantage of the Lyapunov optimization techniques, the COMDT problem is transformed to a one-slot optimization problem, which is solved by typical optimization techniques.

A. Problem Transformation Using Lyapunov Optimization

During the time slot t, the migration delay changes a little, which seems like a queue with arrival and departure. Thus, we introduce a virtual queue to represent the changing part of the migration delay. $Q(t) = 0$ is defined as the initial state of the virtual queue, and it is updated as follows.

$$Q(t+1) = \max\{Q(t) + \alpha(t) - \mathcal{T}, 0\}, \tag{8}$$

where $\alpha(t)$ is the migration delay and $\mathcal{T}$ is the maximum tolerable delay. And $Q(t)$ is obtained from the backlog of the virtual queue. If the migration delay constraint (6) is satisfied, $\lim_{T\to\infty} \frac{Q(t)}{T} = 0$, which means the constraint (6) enforces the virtual queue stable.

Theorem 1. *The constraint (6) of the COMDT Problem,* $\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \alpha(t) \leq \mathcal{T}$ *is satisfied if and only if the virtual queue is stable, i.e.,* $\lim_{T\to\infty} \frac{Q(t)}{T} = 0$.

Proof: According to Eq. (8), we have:

$$Q(t+1) \geq Q(t) + \alpha(t) - \mathcal{T},$$

then applying Eq. (2) to it, summing up both sides of the equality above over time slots $t \in \{0, T-1\}$, and then divided by T, we have

$$\frac{Q(T-1) - Q(0)}{T} \geq \frac{2}{T \cdot b} \sum_{t=0}^{T-1} \mathcal{D}(t) + s - \mathcal{T}.$$

Finally, letting $T \to \infty$, and applying the initial status of virtual queue, i.e., $Q(0) = 0$, we have

$$\lim_{T\to\infty} \frac{Q(T-1)}{T} \geq \frac{2}{b} \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathcal{D}(t) + s - \mathcal{T},$$

then applying constraint (6), the necessity is proved. When the length of virtual queue is large, it means that the migration delay has far exceeded the time that user can tolerant.

Since the queue is stable, i.e., $\lim_{T\to\infty} \frac{Q(t)}{T} = 0$, we have:

$$\frac{2}{b} \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathcal{D}(t) + s - \mathcal{T} \leq \lim_{T\to\infty} \frac{Q(T-1)}{T} = 0.$$

Therefore, the sufficiency is proved. ■

According to Theorem 1, the constraint (6) is transformed to the stability of the virtual queue. For each time slot t, we define the Lyapunov function $L(t)$ as the square of the current virtual queue backlogs:

$$L(t) = \frac{1}{2} Q(t)^2. \tag{9}$$

To ensure constraint (6) is satisfied, the virtual queue should be enforced toward a lower congestion state. As a result, we define the Lyapunov drift to observe the amount that the Lyapunov function changes between two consistent time slots.

$$\Delta L(t) = E\{L(t+1) - L(t) | Q(t)\}, \tag{10}$$

since $L(t)$ is a function of $Q(t)$, Eq. (10) could be written as

$$\Delta L(Q(t)) = E\{L(Q(t+1)) - L(Q(t)) | Q(t)\}. \tag{11}$$

Thus, the Lyapunov drift is defined as a tool to measure the stability of the virtual queue and is used to enforce the migration delay constraint satisfied.

Theorem 2. *For any time slot t, given any possible VM configuration decisions, the Lyapunov drift $\Delta L(t)$ is bounded as follows.*

$$\Delta L(Q(t)) \leq M + Q(t)E\{2\frac{\mathcal{D}(t)}{b} + B|Q(t)\}, \tag{12}$$

where $M = \frac{1}{2}(2\frac{\mathcal{D}_{max}}{b} + s - \mathcal{T})^2$, and $B = s - \mathcal{T}$.

Proof: First, applying Eq. (9) to Eq. (11), the Lyapunov drift is written as:

$$\begin{aligned}\Delta L(Q(t)) &= E\{L(Q(t+1)) - L(Q(t))|Q(t)\} \\ &= \frac{1}{2}E\{Q^2(t+1) - Q^2(t)|Q(t)\}.\end{aligned} \tag{13}$$

Obviously, $\max^2\{a, 0\} \leq a^2$. Combined with it, Eq. (8) and $B = s - \mathcal{T}$, we have

$$\begin{aligned}Q^2(t+1) \leq Q^2(t) + (2\frac{\mathcal{D}_{max}}{b} + B)^2 \\ + 2Q(t)(2\frac{\mathcal{D}(t)}{b} + B),\end{aligned} \tag{14}$$

where D_{max} is the maximum disk image size.

Applying Eq. (14) to Eq. (13), the Lyapunov drift is further transformed as:

$$\Delta L(Q(t)) \leq M + Q(t)E\{2\frac{\mathcal{D}(t)}{b} + B|Q(t)\},$$

where M is defined above. ■

According to Theorem 2, we know that there is an upper bound of the Lyapunov drift. To minimize the upper bound, we can enforce the migration delay constraint satisfied. According to Lyapunov optimization framework, we then formulate the objective of the one-slot optimization problem with Eq. (7) and (12).

$$\begin{aligned}VC(t) + \Delta L(Q(t)) \leq M + VC(t) \\ + Q(t)E\{2\frac{\mathcal{D}(t)}{b} + B|Q(t)\},\end{aligned} \tag{15}$$

where $C(t)$ is $\sum_{i=1}^{K} x_i(t)c_i$ for simplicity.

Since M is a constant, it could be eliminated from our objective. Hence, the objective of the one-slot optimization problem is:

$$\begin{aligned}\min \quad & VC\left(t\right) + Q\left(t\right)\left(2\frac{\mathcal{D}\left(t\right)}{b} + B\right) \\ s.t. \quad & constraints\ (5)\ (7).\end{aligned} \tag{16}$$

Thus, the COMDT problem is transformed to one-slot optimization problem. For each time slot $t \in [0, T-1]$, we minimize both the migration delay and VM configuration cost. Also, the weight V is added to control the emphasis on the migration delay or VM configuration cost.

B. Optimal Analysis

The weight V in problem (16) represents a tradeoff between the VM configuration cost and migration delay. In this subsection, we analyze the optimality of the one-slot optimization problem. First, we assume that one user sets a fixed V such that $V > 0$ in the algorithm, independent of current queue status. Hence, during any time slot $t \in \{0, 1, \ldots, T-1\}$, we have

$$\frac{1}{T}\sum_{t=0}^{T-1} C\left(t\right) = P^*, \tag{17}$$

where P^* denotes the optimal cost in theory. And based on constraint (6), we get

$$E\{\alpha(t)\} \leq \mathcal{T}. \tag{18}$$

Since when the time slot tends to infinity Eq. (18) is satisfied, there exists a non-negative ϵ, making Eq. (19) satisfied:

$$E\{\alpha(t)\} \leq \mathcal{T} - \epsilon. \tag{19}$$

From the objective (16), it implies that the right side of Eq. (15) is minimized. Taking advantage of the definition of the Lyapunov drift $\Delta L(Q(t))$, we sum up both side of Eq. (15) over time slot $t \in \{0, 1, \ldots, T-1\}$, which eliminates all inner terms, and then divide by T. Also, applying the optimal cost value P^* to the right side, we get

$$\begin{aligned}\frac{V}{T}\sum_{t=0}^{T-1} C\left(t\right) + \frac{L(Q(T)) - L(Q(0))}{T} \\ \leq M + VP^* - \frac{\epsilon}{T}\sum_{t=0}^{T-1} Q(t).\end{aligned}$$

Making $T \to \infty$, since $L(Q(T))$ would not be infinity and $L(Q(0))$ is a constant, $\frac{L(Q(T)) - L(Q(0))}{T} = 0$.

Note that whatever T is, $\frac{V}{T}\sum_{t=0}^{T-1} C(t) > 0$. Therefore, we tend T of both side to infinity, and have

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} Q(t) \leq \frac{M + VP^*}{\epsilon}. \tag{20}$$

What's more, we know $-\frac{\epsilon}{T}\sum_{t=0}^{T-1} Q(t) \leq 0$ is satisfied whatever T is. Hence, we have

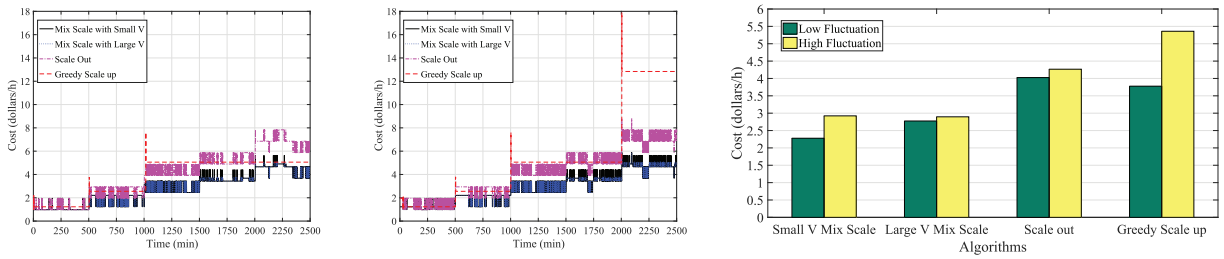
$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} C(t) \leq \frac{M}{V} + P^*. \tag{21}$$

From Eq. (20) and (21), we see that under a larger weight V, the time-average VM configuration cost of our algorithm is pushed closer to the optimal value P^* from Eq. (21). However, according to Eq. (20), the migration delay constraint (6) is not satisfied. On the contrary, Under a smaller weight V, we handle the constraint (6), while no longer cost-effective. Therefore, users can tune the weight V to emphasize the objective between VM configuration cost and migration delay.

IV. Cost-Efficient Mix Scaling Online Algorithm

In this section, we propose a cost-efficient mix scale online algorithm to solve the one-slot optimization problem.

Algorithm 1 shows the Cost-efficient Mix Scaling Online Algorithm. We first solve the original VM configuration and get an optimized configuration (Line 1). Then we check three situations by comparing this configuration with the old configuration (Line 2). Before users deploy their application,



(a) Instant cost under low-fluctuation (b) Instant cost under high-fluctuation (c) Average cost

Fig. 2. Cost evaluations under different fluctuation TPC-W workload

Algorithm 1 Cost-Efficient Mix Scaling Online Algorithm

Input: x^{old}: the old VM configuration
c: the cost of all types of VMs
μ: the maximum service rate of all types of VMs
K: the number of VM types
λ: the arriving rate
$\mathcal{D}$: the volume of disk image in this time slot
b: the network bandwidth
s: the startup time of a new VM
$\mathcal{T}$: the maximum tolerable delay
Output: x^{new}: the new VM configuration of this time slot

1: Get a new configuration x_{new} by solving original problem
2: $x^{dif} = x^{new} - x^{old}$
3: **if** for all K element in $x^{dif} \geq 0$ **or** for all K element in $x^{dif} \leq 0$ **then**
4: **return** x^{new}
5: **end if**
6: Here we should ask user for a tuning parameter V
7: $x^{new} = x^{old}$
8: **for** i:= 1 : K **do**
9: $x^{tmp} = x_i^{new} + 1$
10: **while** x^{tmp} cannot satisfy constraint (5) **do**
11: $x^{tmp} = x_i^{tmp} + 1$
12: **end while**
13: $x\_list_i = x^{tmp}$
14: **end for**
15: Calculate all costs of elements in x_list_i and sort x_list_i based on these costs in ascending order
16: $x^{new} = x\_list_1$
17: **return** x_{new}

the arrival rate is a unknown parameter. We let users select the type of deployed applications, and use the average arriving rate of this kind of application as the predicted value. If the comparison satisfies the first two situations, the optimized configuration is the result (Line 3 - Line 5). Then system provisions more VMs or shutdown useless VMs. Otherwise, we construct a one-slot optimization problem. Since the weight V is the tradeoff between the migration delay and cost, we first ask users for it to determine their objective (Line 6). Then we greedily calculate all configuration candidates which satisfy all constraints (Line 8 - Line 14). Since we increase each flavor just by one, and judge whether it satisfies the constraint, the configuration results would be close to the optimized result. Finally, we calculate the cost of all configuration candidates, and sort them in a descending order, then take the first one as our new configuration (Line 15 - Line 16).

V. Experiment Result

In this section, we conduct simulations based on OpenStack performance data and TPC-W workload data to evaluate the performance of Cost-Efficient Mix Scaling Online Algorithm.

A. Simulation Setup

We take the simulation data from following sources.

1) Application Workload: we measure requests arrival rate as the application workload and take the workload from TPC-W. TPC-W simulates the activities of a business oriented transactional web server [15] [16]. Hence, the workload of TPC-W is close to the realistic e-commercial website workload.

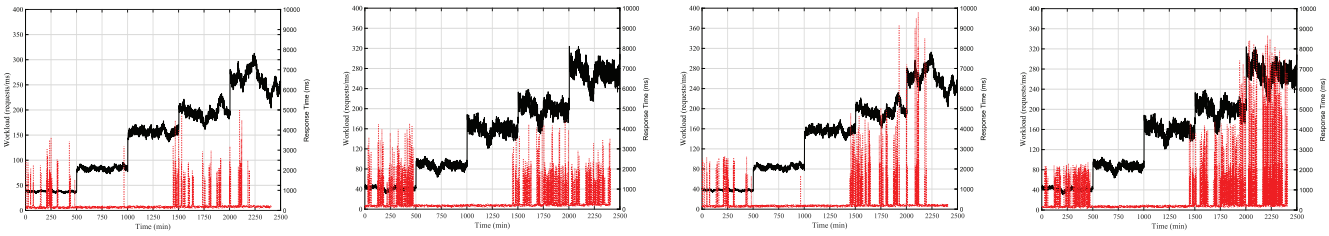
2) VM Capacity: OpenStack [17] is now the most popular cloud platform in the world. Hence, we build a OpenStack platform on a cluster, which contains 10 servers, and each server has 32 VCPUs, 62.9GB memory and 17.2TB storage disks. The flavors in the platform are configured the same as that in TABLE I. Also, all images contains a TPC-W server. With monitoring the resources usage of VMs, we get the maximum services rate of each flavor, i.e., μ.

3) VM Price: Since our platform is not a commercial platform, we choose to take a price model from AWS, which is the largest public cloud provider. Hence, we take the same price model in TABLE I.

B. Performance Evaluation under TPC-W workload

We make simulations to evaluate the performance of our algorithms compared with the scale out and greedy scale up algorithms under TPC-W workload.

Cost evaluations. Fig. 2(a) and Fig. 2(b) show the instant cost evaluation results under low-fluctuation and high-fluctuation workload respectively. The low-fluctuation workload means the workload fluctuation is small, while high-fluctuation operates the opposite. From Fig. 2(a) and Fig. 2(b), we see that the costs of scale out and greedy scale up algorithms are more stable than mix scale algorithms. This is because mix scale algorithms are more sensitive to the changing workload. There are also some difference between mix scale algorithms. After 1000 min, instant costs of two mix scale algorithms separate clearly. Larger V mix scale



(a) V=0.001 under low-fluctuation (b) V=0.001 under high-fluctuation (c) V=9999999 under low-fluctuation (d) V=9999999 under high-fluctuation

Fig. 3. Response time evaluations of mix scaling algorithms different fluctuation TPC-W workload

achieves higher cost-efficiency since it takes no consideration of migration delay. Furthermore, in Fig. 2(b), scale out and greedy scale up algorithms cost more in VM configurations.

Fig. 2(c) shows the average cost evaluation under low-fluctuation and high-fluctuation workload respectively. Under low-fluctuation workload, the mix scale algorithm (with s-maller V) achieves 30.8% and 26.3% higher cost-efficiency than the scale out and greedy scale up algorithms. The mix scale algorithm (with larger V) achieves 31.1% and 26.5% higher cost-efficiency than the scale out and greedy scale up algorithms respectively. Under high-fluctuation workload, the cost-efficiency is higher. The mix scale algorithm (with smaller V) achieve 31.5% and 45.5% higher cost-efficiency than the scale out and greedy scale up algorithms.

Response time evaluations. Fig. 3(a) and Fig. 3(b) show the response time evaluation of small V mix scale algorithm under low-fluctuation and high-fluctuation workloads. Fig. 3(c) and Fig. 3(d) show the similar evaluation of large V mix scale algorithm. Since only mix scale algorithms introduce migration delay into the system, only mix scale algorithms are evaluated. In Fig. 3(a) and Fig. 3(b), mix scale algorithm tend to choose migrate VMs to achieve higher cost-efficiency under high-fluctuation workload. Thus, migrations occur more frequently and higher response time is brought. Small V mix scale reduces 35.85% response time under small-fluctuation workload than that under large-fluctuation workload. In Fig. 3(c) and Fig. 3(d), we see that large V mix scale reduces 42.59% response time under small-fluctuation workload than that under high-fluctuation workload.

Comparing Fig. 3(a) with Fig. 3(c), we see small V mix scale algorithm brings lower response time than large V mix scale algorithm under the same workload. Since small V mix scale takes migration delay into consideration, this mix scale controls migration delay in a tolerable range. Thus, the response time of small V mix scale is 30.93% lower than that of large V mix scale. Fig. 3(b) and Fig. 3(d) show that small V mix scale reduces 38.19% response time than that of large V mix scale under high-fluctuation workload.

VI. Conclusion

Taking advantages of pay-per-use pricing model in the cloud, many companies have migrated their services to the cloud. Making VM configuration decision cost-efficiently under dynamic workload becomes a critical challenge. In this paper, we propose a mix-scale online algorithm to make cost-efficient VM configurations based on Lyapunov optimization techniques. By simulations under TPC-W workload, we demonstrate that our solution achieves efficiency both in minimizing the VM configuration cost and controlling the delay in tolerable range.

Acknowledgment

This research was sponsored by National 863 Program (No.2015AA01A202).

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] TechTarget, "Special report: Tackling the cloud computing landscape in enterprise it," tech. rep., 2013.

[3] Microsoft, "Microsoft azure." [Online]. Available: https://www.azure.cn/home/features/what-is-windows-azure/, 2016.

[4] Google, "Google app engine." [Online]. Available: https://appengine.google.com/, 2016.

[5] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and qos in cloud computing environments," in *Proc. ICPP'11*, pp. 295–304, 2011.

[6] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proc. ICDCS'11*, pp. 559–570, June 2011.

[7] S. Zaman and D. Grosu, "Combinatorial auction-based dynamic vm provisioning and allocation in clouds," in *Proc. CloudCom'11*, pp. 107–114, 2011.

[8] S. Zaman and D. Grosu, "A combinatorial auction-based mechanism for dynamic vm provisioning and allocation in clouds," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 129–141, 2013.

[9] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *Proc. ICDCS'10*, pp. 62–73, 2010.

[10] Amazon, "AWS — Auto Scale." [Online]. Available: http://aws.amazon.com/autoscaling/, 2016.

[11] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[12] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li, "etime: Energy-efficient transmission between cloud and mobile devices," in *Proc. INFOCOM'13*, pp. 195–199, 2013.

[13] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, 2012.

[14] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[15] PHARM in the University of Wisconsin-Madison, "Java TPC-W implementation distribution." [Online]. Avaliable: http://pharm.ece.wisc.edu/tpcw/tpcw.tar.gz, 2016.

[16] TPC Organization, "TPC-W - Homepage." [Online]. Available: http://www.tpc.org/tpcw/, 2016.

[17] Rackspace Cloud Computing, "OpenStack Open Source Cloud Computing Software." [Online]. Available: http://www.openstack.org/, 2016.