Handling Flash Deals with Soft Guarantee in Hybrid Cloud

Yipei Niu[1] Fangming Liu*[1] Xincai Fei[1] Bo Li[2]
[1]Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology, China.
[2]The Hong Kong University of Science and Technology, Hong Kong.

***Abstract*—Flash deal applications, which offer significant benefits (e.g., discount) to subscribers within a short period of time, are becoming increasingly prevalent. Motivated by such transient profit, flash crowds of subscribers request services simultaneously. Considering the unique business logic, a hybrid cloud with soft guarantee, i.e., bounding the response time of delay-tolerant requests, has great potential to handle flash crowds. In this paper, to cost-effectively withstand flash crowds with soft guarantee, we propose a solution that makes smart decisions on scheduling requests in the hybrid cloud and adjusting the capacity of the public cloud. In respect of scheduling requests, we apply Sequential Quadratic Programming (SQP) to achieve soft guarantee. Furthermore, for adjusting capacity, we design an online algorithm to tune the scale of the public cloud towards jointly minimizing cost and response time, yet without a priori knowledge of request arrival rate. We prove that the online algorithm can obtain a competitive ratio of $1-6\epsilon$ against the optimal solution, where ϵ can be tuned close to 0. By conducting extensive trace-driven experiments in a website prototype deployed on `OpenStack Mitaka` and Amazon Web Service, our solution reduces response time by 15% compared with previous work under given budget.**

I. Introduction

With the increasing population of online e-commerce users and their interactions in the online forums, *flash deal* has become a common marketing strategy in which websites offer significantly discounted or limited products on sale for a short period. The rise of social networks, such as Facebook and Twitter, has further accelerated its growth, allowing popular deals to spread virally. Such a model has become a norm in many group buying websites, e.g., Groupon, and even in traditional e-commerce vendors as well, e.g., Prime Day of Amazon.

For example, on the Prime Day (July 15, 2015) that celebrates Amazon's 20th anniversary, thousands of lightning deals were offered, starting as often as every 10 minutes, sales on Amazon's Prime Day even exceeded Black Friday in 2014 [1]. In addition to Amazon's Prime Day, when Apple's new iPhones are released, the pre-orders exceeded two million in the first 24 hours [2]. Besides the discounted and pre-ordered products, the income brought by flash deals has appeared too as incentives. WeChat, the most popular mobile instant messaging platform in China, has offered virtual *red envelope* containing virtual money that can be cashed out. A red envelop is distributed to a group of WeChat users, but only the first some persons who catch the chance would be able to share the envelope and hence the money. Such an interesting and fast way of getting red envelopes soon becomes popular, and a total of 1.2 billion red envelopes (over $83 million) were unleashed on the Chinese New Year's Eve [3].

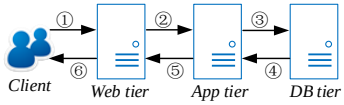

Fig. 1. Common workflow of flash deal applications. When web servers accept user requests, web servers need to wait for the results returned from the application tier and then send responses back to users.

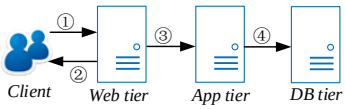

Fig. 2. Workflow of flash deal applications with soft guarantee. When web servers accept the user requests, instead of waiting for the results returned from the application tier, web servers send responses back to users. The requests are served asynchronously.

These flash deal applications demand for ultra fast responses from both vendors and consumers. Unfortunately, during short lifetimes of the flash deal applications, vendors have to handle dramatic flash crowds of subscriber requests, which may lead to long response time and even crash of applications. To put things into perspective, in the Chinese New Year's Eve, the times of shaking phones to request red envelopes reached a total of 11 billion and a peak of 810 million per minute [3]. Furthermore, the pre-orders of new iPhones exceeded two million in the first 24 hours, rendering the Apple Store in many regions to be unresponsive [2]. As a result, it is vital for flash deal providers to maintain fast response time under flash crowds.

Yet, given the incentive comes from the discounts (or money in the red envelop case), it is possible for subscribers to accept service degradation, i.e., postpone serving their requests. Moreover, considering the simple business logic, it is feasible to handle such flash crowds with soft guarantee, i.e., guaranteeing the postponed requests to be served within certain deadline. As illustrated in Fig. 1, a user sends a request for a red envelop (discounted or limited product) to the web tier of a flash deal application. Normally, the application and database tiers process the request and return a result

*The Corresponding Author is Fangming Liu (fmliu@hust.edu.cn). This work was supported in part by the National Natural Science Foundation of China under Grant 61520106005, in part by the National 863 Hi-Tech Research and Development Program under grant 2015AA01A203, and in part by the National 973 Basic Research Program under Grant 2014CB347800.

to the web tier. Finally, the web tier sends a response to the user. With soft guarantee, as shown in Fig. 2, as soon as the web tier receives and verifies the request, it sends a response to the user immediately, which significantly reduces the interactive response time. However, the response only contains the information that the user needs to wait for the result. Then the request is sent to the asynchronous process where we postpone serving it. After the application finishes serving the request asynchronously, the user will get the result whether she wins the red envelop or the product.

To improve interactive response time, the requests are scheduled to two processes, i.e., the interactive and the asynchronous processes. Correspondingly, how to schedule requests to maintain fast interactive response time as well as bound execution time of requests served asynchronously, becomes a non-trivial problem.

From the perspective of flash deal providers, although service degradation potentially relieves the pressure on the infrastructure, they still have to address the dramatic surge in user requests. To avoid long response time and even crashes led by the flash crowds, a hybrid cloud solution is a preferred strategy, which is adopted by 82 percent of enterprises [4]. As shown in Fig. 3, leveraging a hybrid cloud, the flash deal providers outsource excessive workloads to the public cloud via a dedicated connection offered by cloud providers, such as AWS Direct Connect. However, given the short duration, the flash deal providers can hardly have a *priori* knowledge of request arrival rate. As a result, it is challenging for the flash deal providers to make online decisions on adjusting the capacity of the public cloud to achieve a cost-effective solution during the long run of applications, i.e., maintaining fast response time as well as incurring less outsourcing cost.

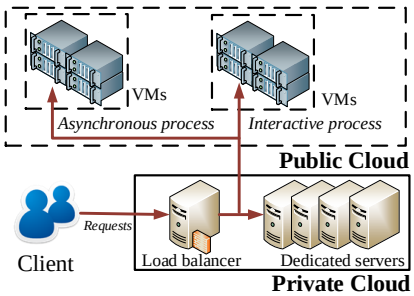

Fig. 3. When flash deals arrive, the load balancer in the private cloud outsources the excessive workloads to the public cloud. To facilitate communication between the private and public clouds, most cloud providers offer a dedicated connection for cloud tenants, such as AWS Direct Connect.

To handle flash deals cost-effectively with soft guarantee in the hybrid cloud, our contributions lie in two folds. On the one hand, to handle request-scheduling problem, we design the Workload Distribution Algorithm (WDA) which is used to distribute requests between the private and public clouds to minimize response time. Afterwards we design the Service Degradation Algorithm (SDA) to achieve soft guarantee, which schedules partial requests to the asynchronous process and maintains their execution time within an acceptable range, so as to further reduce the response time. Both WDA and SDA are designed by applying Sequential Quadratic Programming (SQP) [5], which converges within only tens of iterations. On the other hand, in respect of capacity adjustment, we design the Capacity-Adjusting Algorithm (CAA) by adapting one-time learning algorithm [6], so as to tune the scale of the public cloud to obtain a cost-effective hybrid cloud solution, i.e., maximizing the performance-cost ratio as well as controlling cost within budget. Through substantial theoretical analysis, we prove that CAA can obtain a competitive ratio of $1-O(\epsilon)$, where ϵ is a small control parameter and can be tuned close to 0.

In experiments, we build a private cloud with `OpenStack Mitaka` [7] under `nova-network` [8] and rent 20 AWS EC2 instances as the public cloud. Deploying a three tier website prototype on the hybrid cloud platform, we conduct real world trace-driven experiments to verify the efficacy of the proposed solution. Compared with a previous solution in [9], ours reduces response time by 15% on average under given budget.

II. Design Objectives and Solutions

A. Design Objectives

Fast interactive response time and soft guarantee. Concerning the bursty, fluctuating, and unpredictable flash deals, it is challenging to make online decisions on scheduling requests between the private cloud and the public cloud, so as to achieve fast interactive response time. Moreover, to further reduce interactive response time, we attempt to schedule partial requests which are already assigned to the public cloud to the asynchronous process, as shown in Fig. 3. Although flash deals can accept service degradation, we need to determine a maximum number of postponed requests and ensure their response time within an acceptable range, i.e., soft guarantee.

Cost-effectiveness. Besides achieving fast interactive response time as well as soft guarantee, it is necessary to design a cost-effective hybrid cloud solution, which spends the least money on achieving the best performance. Here we introduce a performance-cost ratio (PC ratio for short), which is obtained by dividing the reciprocal of average response time with outsourcing cost, to measure the cost-effectiveness of a hybrid cloud solution. By maximizing the PC ratio, we can obtain a cost-effective solution.

B. Our Solutions

Scheduling requests in two sub-stages. To achieve the first design objective, scheduling currently arrived requests under given cloud capacity contains two sub-stages. The first is scheduling requests between the private and public clouds, namely, workload distribution. In this sub-stage, we determine how many requests to be served immediately in the public cloud without taking service degradation into consideration. The other is scheduling requests assigned to the public cloud to the asynchronous process where requests get served within certain deadline.

Adjusting capacity under given budget. Given that flash deal providers lease the public cloud to handle excessive

TABLE I
KEY PARAMETERS

Notation	Definition
λ	The average arrival rate of requests in a queueing system
μ	The average service capacity of a queue
ρ	The traffic intensity of a queue, denoted as $\frac{\lambda}{\mu}$
f	The average response time of requests in a queue
d_t	The average response time during the t^{th} time slot
n_t	The number of running EC2 instances during the t^{th} time slot

flash deals, the capacity of the public cloud plays a critical role in maximizing PC ratio during the long run of flash deal applications, i.e., reducing response time and controlling cost. Hence, our solution needs to make online decisions on adjusting the capacity of the public cloud, which in turn impacts on the strategy for scheduling requests. In terms of adjusting cloud capacity, most public cloud providers provision auxiliary services, which enable public clouds to scale up or down automatically. For instance, the corresponding service in AWS EC2 [10] is known as Auto Scaling [11]. However, such automatic adjustment in scale makes it difficult to control cost. Therefore, to carefully control outsourcing cost, cloud tenants need to adjust the capacity of the public cloud proactively. According to the scaling policy in AWS EC2, the tenants can adjust the number of running EC2 instances manually [12]. For cloud tenants, when leasing the public cloud, they mostly set a budget in advance to limit outsourcing cost. Hence, we introduce a budget from the perspective of cloud tenants. Correspondingly, our solution tunes the scale of the public cloud to maximize the PC ratio, and finally controls outsourcing cost under a given budget.

III. SYSTEM MODEL

A. Queue-Based Model for Flash Deal Applications

1) Single-tier architecture: For flash deal applications deployed in clouds, we initially assume that the application is developed as a single-tier architecture, as in previous study [13]–[16]. We also assume that the request arrival is a Poisson process [15], [17], [18] and the service time of requests is generally distributed.

As such, we model the application as an M/G/1/PS queue, which is motivated by early studies in this field. For example, in [17], Pacifici *et al.* employ an M/M/1 queueing model to compute response time of web requests; in [18], Villela *et al.* use an M/G/1/PS model for each server in a tier, and similar queueing model is used in [15] for an e-commerce application. In the queueing model, when the request arrival rate is near the service capacity, the average queueing delay approaches infinity, implying that the application becomes unresponsive. The key parameters are summarized in Table I.

Assuming that the service time x is i.i.d and its probability distribution function is $F(x)$. The performance of the application, particularly the average response time, can be calculated as:

$$f^{ST}(\lambda) = \int_0^\infty \frac{x}{1-\rho} dF(x) = \frac{E[X]}{1-\rho}, \tag{1}$$

where $\rho = \frac{\lambda}{\mu}$ is traffic intensity of the queueing system and $E[X]$ is the expectation of processing times in the single tier system.

2) Multi-tier architecture: In practice, most applications however are developed as a multi-tier architecture. For each tier, we model it as an M/G/1/PS queue as in early studies [19]–[21]. Hence, we next extend the model by assuming the application deployed in the cloud is of K-tiered.

Lemma 1. *The request leaving process of the k^{th} tier, i.e., the request arrival process of the $(k+1)^{th}$ tier, still follows Poisson process, and the arrival rate $\lambda_{k+1} = \lambda_k$ when the queueing system is stable.*

We prove the lemma in our technical report [22].

Based on Lemma 1, by summing up the average response time of each tier f_k, the average response time of requests f^{MT} which are served by the K-tier system can be estimated as:

$$f^{MT}(\boldsymbol{\lambda}) = f^{MT}(\lambda_1, \lambda_2, ..., \lambda_K) = \sum_{k=1}^{K} f_k^{ST}(\lambda_k), \tag{2}$$

where ρ_k is the traffic intensity of the k^{th} tier and denoted as $\frac{\lambda_k}{\mu_k}$.

3) Soft guarantee: As aforementioned in Sec. I, considering the delay-tolerant characteristics of flash deals, we postpone serving certain requests and bounding their execution time within a deadline, i.e., soft guarantee. As a result, we next extend the multi-tier architecture with soft guarantee.

Obviously, not all the requests can accept such service degradation. Furthermore, scheduling too many requests to the asynchronous process also leads to bad user quality of experience. Hence, we assume that the number of requests which to be postponed and assigned to the asynchronous process is α. Based on the sojourn time of each request in the application, we classify the requests to be scheduled to the asynchronous process into a set of different priority classes, indexed by $k \in \{1, 2, ..., K\}$. Hence, we have:

$$\textstyle\sum_{k=1}^{K} \alpha_k \leq \alpha, \tag{3}$$

where α_k is the number of requests scheduled to the asynchronous process in the k^{th} tier. As such, we model the asynchronous process as a priority queue. We denote the average response time of the asynchronous process as follows [23]:

$$f^{AP}(\boldsymbol{\alpha}) = \frac{C \sum_{k=1}^{K} \alpha_k}{(1 - \sum_{k=1}^{K-1} \frac{\alpha_k}{\mu^{AP}})(1 - \sum_{k=1}^{K} \frac{\alpha_k}{\mu^{AP}})}, \tag{4}$$

where $C = \frac{1}{2}(E^2[X_k] + Var[X_k])$ and μ^{AP} is the service rate of the priority queue. Moreover, to avoid the sojourn time approaching infinity, we need to enforce that:

$$\sum_{k=1}^{K} \frac{\alpha_k}{\mu^{AP}} < 1. \tag{5}$$

B. Performance of Flash Deal Applications in Hybrid Cloud

Before presenting the performance model of the hybrid cloud, we consider a discrete time model $t \in \{1, 2, ..., m\}$, whose length equals to the average delay of starting up one

cloud instance (e.g., an Amazon's EC2 instance [24]). During each time slot, the number of EC2 instances in the system therefore remains constant. Based on the time model and the system model, we now estimating the average response time in the hybrid cloud.

1) Response time in the private cloud: Considering the capacity of the private cloud is limited and difficult to be flexibly adjusted, we assume the application is single-tiered and model it as an M/G/1/PS queue. Based on Eq. (1), the performance of the application deployed in the private cloud during the t^{th} time scale, particularly the average response time, can be calculated as:

$$d_t^V = f^{ST}(\lambda_t^V),$$

where λ_t^V is the number of requests assigned to the private cloud during the t^{th} time slot.

Remark: Based on the formulation above, when ρ_t^V, which is calculated as $\frac{\lambda^V}{\mu^V}$, is near 1, the average response time approaches infinity. Hence, the number of requests assigned to the private cloud has profound impact on performance, which explains why a queue should be used here for modeling the private cloud instead of a constant. Moreover, our efforts mainly are devoted to the public cloud, we therefore also model it as single-tiered, rather than multi-tiered for simplicity.

2) Transmission time in the dedicated tunnel: As discussed above, the capacity of the private cloud is limited, and so the hybrid cloud users leverage a public cloud to handle bursty and immense flash deals. Using a dedicated tunnel, the hybrid cloud users can offload the excessive requests to the public cloud. For example, AWS Direct Connect [25] is a dedicated network connection from users' premises to AWS. Using the AWS Direct Connect, the users can establish a private tunnel between the public cloud and the private cloud, which in many cases can reduce the network costs, increase the throughput, and provide more stable network experience than public Internet based connections. Here we still model the tunnel as an M/G/1/PS queue, which has an average transmission time as

$$d_t^R = \int_0^\infty \frac{x}{1-\rho_t^R} dF(x) = \frac{E[X^R]}{1-\rho_t^R},$$

where $\rho_t^R = \frac{\lambda_t^R}{\mu_t^R}$ is traffic intensity of the private cloud.

3) Response time in the public cloud: Since the public cloud plays a critical role in handling flash deals, we now focus on the model of the application deployed in the public cloud. Most applications are developed as a 3-tier architecture, and we model them as M/G/1/PS queues.

The arrival of requests in the k^{th} tier at the asynchronous process, as well as the remaining request in the interactive process, follows Poisson process. And the request arrival rates are α_k and $\lambda_k - \alpha_k$, respectively [26].

When we extend the multi-tier applications with service degradation, based on the Eq. (2) and Eq. (4), the respective average response time of interactive d_t^{IP} and asynchronous processes d_t^{SG} can be denoted as:

$$d_t^{IP}(\boldsymbol{\lambda}_t, \boldsymbol{\alpha}_t) = f^{MT}(\boldsymbol{\lambda}_t - \boldsymbol{\alpha}_t),$$
$$d_t^{SG}(\boldsymbol{\alpha}_t) = f^{AP}(\boldsymbol{\alpha}_t).$$

C. Problem Formulation

1) Request-scheduling problem:

Sub-problem 1: workload distribution. When distributing requests between the private and public clouds, there exist two different cases: i) the capacity of the hybrid cloud exceeds the request arrival rate; ii) the capacity of the hybrid cloud is lower than the request arrival rate.

For the former case, Equation (1) can be used to estimate the average response time, which unfortunately is not applicable to the latter. To this end, we propose a general model to estimate the response time in both cases, as below.

$$d_t^H = \frac{\lambda_t^V}{\lambda_t} d_t^V + \frac{\lambda_t^U}{\lambda_t}(d_t^{IP} + d_t^R) + \frac{\lambda_t - \lambda_t^V - \lambda_t^U}{\lambda_t} \cdot D,$$

where D is a predefine maximum response time and d_t^H is the average response time of requests in the hybrid cloud. Here we assume that D is larger than the maximum response time of all the requests in the interactive process. As a result, based on the formulation, it is obvious that d_t^H is smaller than D.

Since it takes a certain time to start up an EC2 instance, the capacity of the public cloud cannot change immediately to handle the coming flash deals, i.e., the number of EC2 instances is constant during the t^{th} time slot. Given the capacity of the hybrid cloud, we need to distribute requests between the private and public clouds. The objective is to make sure that the application can provision the best services, i.e., try to maintain fast response time. That is

$$\begin{aligned} \min \quad & d_t^H(\lambda_t^V, \lambda_t^U) \\ \text{s.t.} \quad & \lambda_t^V + \lambda_t^U - \lambda_t \le 0, \end{aligned} \tag{6}$$

where λ^V and λ^U are the request arrival rate of the private and public clouds, respectively.

Remark: This problem is a linearly constrained convex problem. We use the Lagrange-Newton SQP method [27] to solve it in Sec. IV-A. For the requests redirected to the public cloud, we further determine how many requests to be assigned to the asynchronous process.

Sub-problem 2: service degradation. For requests redirected to the public cloud, i.e., λ_t^U, the postponed requests are classified into different sets based on their sojourn times in the hybrid cloud and each set is associated with a priority. Requests with long sojourn time in the application have high priority, which means that they are served early. For the postponed requests, we define a deadline of the execution time. We need to ensure that all the postponed requests are completed before the deadline, which can be denoted as follows:

$$d_t^{SG}(\boldsymbol{\alpha}_t) \le L, \tag{7}$$

where L is the predefined deadline. d_t^{SD} is the average execution time of postponed requests, which can be derived in Sec. III-B3 .

The problem can be formulated as follows:

$$\begin{aligned} \min \quad & d_t^{IP}(\boldsymbol{\alpha}_t) \\ \text{s.t.} \quad & (3), (5), (7). \end{aligned} \tag{8}$$

Remark: This problem is a nonlinearly constrained convex problem. Furthermore, more complicated than the workload

distribution problem, the constraint (7) is fractional, we therefore apply the Quasi-Newton SQP methods [28] to solve it in Sec. IV-A.

2) Capacity-adjusting problem: In practice, when leveraging public clouds, cloud tenants always have a budget b, which limits the cost incurred by the system, i.e., only a limited number of EC2 instances can be leased. Hence, we define that the number of EC2 instances a tenant can boot is n, where $0 < n < N$ and N is the maximum number of EC2 instances a tenant plan to lease.

During the t^{th} time slot, the tenants need to decide how many EC2 instances to run, so as to handle bursty and immense flash deals smoothly as well as control the outsourcing cost. The decision on leasing n EC2 instances can be denoted as $\boldsymbol{x}_t^{(n)}$, where all N entries are 0 except that the n^{th} one is 1.

Meanwhile, the real capacity of an application deployed in the hybrid cloud cap_t is defined as $\frac{1}{d_t^H}$. As a result, the performance-cost ratio $\pi_t^{(n)}$ is denoted as:

$$\pi_t^{(n)} = \frac{cap_t}{a_t \cdot n_t} = \frac{1}{a_t d_t^H \boldsymbol{J}^{\mathrm{T}} \boldsymbol{x}_t^{(n)}}, \boldsymbol{J} = (1, 2, ..., N)$$

where a_t is the price of an EC2 instance and d_t^H is the average response time of requests in the hybrid cloud. The PC ratio means how many requests are served with unit amount of money in each time unit, implying whether the application provisions cost-effective services.

Due to the unpredictable flash deals and potentially changing price, $\boldsymbol{\pi}_t$ and a_t cannot be known in advance. The decision $\boldsymbol{x}_t$ must be made based on the two variables. Given the previous $t-1$ decisions $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{t-1}\}$, and input $\{\boldsymbol{\pi}_i, a_i\}_{i=1}^t$, the decision $\boldsymbol{x}_t$ has to satisfy

$$\textstyle\sum_{i=1}^t a_i \boldsymbol{J}^{\mathrm{T}} \boldsymbol{x}_i = \sum_{i=1}^t \boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x}_i \leq b,$$

where $\boldsymbol{c}_i^{\mathrm{T}} = a_i \cdot \boldsymbol{J}^{\mathrm{T}}$.

During the t^{th} time slot, we need to maximize the PC ratio and control cost within budget, which can be formulated as follows:

$$\max \quad \sum\nolimits_{t=1}^m \boldsymbol{\pi}_t^{\mathrm{T}} \boldsymbol{x}_t \tag{9}$$

$$\text{s.t.} \quad \sum\nolimits_{i=1}^t \boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x}_i \leq b, \quad i = 1, 2, ..., m, \tag{10}$$

where $\boldsymbol{\pi}_t = (\pi_t^{(1)}, \pi_t^{(2)}, ..., \pi_t^{(n)}, ..., \pi_t^{(N)})$ and $\boldsymbol{x}_t \in \{\boldsymbol{x}_t^{(1)}, \boldsymbol{x}_t^{(2)}, ..., \boldsymbol{x}_t^{(n)}, ..., \boldsymbol{x}_t^{(N)}\}$.

Remark: The problem above is an online knapsack problem, which is known as NP-hard. Moreover, the problem is a multi-dimension version, whose solution is a matrix, making it more complicated to handle. Finally, considering unpredictable and fluctuating flash crowds, it becomes much more challenging to control cost within budget. In the next section, we will demonstrate an effective online solution to it without a *priori* knowledge of request arrivals.

IV. Design of Algorithms

A. Algorithms for Scheduling Requests

For optimization with linear or quadratic objective functions, there exist a number of effective solutions, e.g., the active set and the Goldfarb-Idnani method. The objective functions of sub-problems (6) and (8) however are more complicated than linear and quadratic ones, and we therefore use SQP method to solve these sub-problems.

Based on Newton-Lagrange SQP method [27], we first define the scalar-valued Lagrangian function for the workload distribution problem (6) as follows:

$$\mathcal{L}(\boldsymbol{\lambda}, u) = d_t^H(\boldsymbol{\lambda}) + ug(\boldsymbol{\lambda}),$$

where $g(\boldsymbol{\lambda}) = \lambda_t^V + \lambda_t^U - \lambda_t$.

Based on Karush-Kuhn-Tucker (KKT) conditions, it follows that we only need to solve the quadratic problem (QPL) below:

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{l}^T \nabla d_t^H(\boldsymbol{\lambda})\boldsymbol{l} + g^T(\boldsymbol{\lambda})\boldsymbol{l} \\ \text{s.t.} \quad & g(\boldsymbol{\lambda}) + \boldsymbol{e}^T\boldsymbol{l} \leq 0, \end{aligned}$$

where $\boldsymbol{l}$ is the step length when searching the solution to the QPL and $\boldsymbol{e}$ is an all-one vector. The algorithm is demonstrated as follows:

Algorithm 1 Workload Distribution Algorithm

1. Initialize $\boldsymbol{\lambda}^{(0)}$ and $k = 0$;
2. Determine a solution $\boldsymbol{l}_k$ and a corresponding Lagrange multiplier $u^{(0)}$ of $(QPL)_k$;

if $l_k > 0$ **then**
  $\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \boldsymbol{l}_k$;
  Set $k = k + 1$ and repeat step 1.
end if

After determining how many requests to be redirected to the public cloud, we then solve the service degradation problem. We use the Quasi-Newton SQP method to solve it, which is proved to be global convergence in [28]. First, we define the Lagrangian function as follows:

$$\mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{u}) = d_t^{IP}(\boldsymbol{\alpha}) + \boldsymbol{u}^T \boldsymbol{g}(\boldsymbol{\alpha}),$$

where $\boldsymbol{g}(\boldsymbol{\alpha})$ is derived from constraints (3), (5), (7), respectively. $g_1(\boldsymbol{\alpha}), g_2(\boldsymbol{\alpha}) \leq 0$ and $g_3(\boldsymbol{\alpha}) = 0$.

Then, we define the penalty function as follows:

$$\Phi_r(\boldsymbol{\alpha}) = f(\boldsymbol{\alpha}) + \tfrac{1}{r_1} g_1(\boldsymbol{\alpha}) + \tfrac{1}{r_2} g_2(\boldsymbol{\alpha}) + \tfrac{1}{r_3} |g_3(\boldsymbol{\alpha})|,$$

where $r_1, r_2, r_3 > 0$.

Based on the algorithm in [5], we just need to solve the following quadratic programming (QP) problem:

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{l}^T B_k \boldsymbol{l} + \boldsymbol{g}_k^T \boldsymbol{l} \\ \text{s.t.} \quad & h_i^{(k)}(l) \leq 0, \\ & h_i^{(k)}(l) = 0, \end{aligned}$$

where $B_k = \nabla^2 d_t^{IP}(\boldsymbol{\alpha}^{(k)}) + \sum_{i=1}^{3} \alpha \nabla^2 g_i(\boldsymbol{\alpha}^k)$, $h_i^{(k)}(l) = g_i^{(k)} + g_i^{'}(\boldsymbol{\alpha}^k)l$, and l is the step length when searching solution.

Finally, the algorithm can be demonstrated as follows.

Algorithm 2 Service Degradation Algorithm

1. Initialize $\boldsymbol{x}^{(0)}$, B_0 and $k = 0$;
2. Determine a solution $\boldsymbol{l}_k$ and a corresponding Lagrange multiplier $u^{(0)}$ of $(QP)_k$;
3. Determine a $\theta_k \in [0, 1]$ with $\Phi_r(\alpha^{(k)} + \theta_k l_k) = \min\{\Phi_r(\alpha^{(k)} + \theta_k l_k) | 0 \leq \theta \leq 1\}$ and set $\alpha^{(k+1)} = \alpha^{(k)} + \theta_k l_k$;
4. Update B_k to B_{k+1} and set $k = k + 1$. Go to step 2.

B. A Near-Optimal Algorithm for Adjusting Capacity

Here we use a one-time learning algorithm [6] for capacity adjustment. In [6], the one-time learning algorithm is applied to single-dimension online problems, whose solution is a vector. However, our capacity-adjusting problem is a multi-dimension version. As a result, by adapting the one-time learning algorithm for our multi-dimension problem, we design an online algorithm and present substantial theoretical analysis on its effectiveness.

First, we define a partial linear problem on the input during $\{0, s\}$, where $s = \epsilon m$ and $0 < \epsilon < 1$. The partial linear problem defined only on the input during $\{0, s\}$ is denoted as follows:

$$\begin{aligned} \max \quad & \sum_{t=0}^{s} \boldsymbol{\pi}_t^T \boldsymbol{x}_t \\ \text{s.t.} \quad & \sum_{t=0}^{s} \boldsymbol{c}_t^T \boldsymbol{x}_t \leq (1-\epsilon)\epsilon b, \\ & \boldsymbol{x}_t \in K, t \in [0, s]. \end{aligned} \tag{11}$$

where $\epsilon = \frac{s}{m}$. Next we have the corresponding dual problem:

$$\begin{aligned} \min \quad & (1-\epsilon)\epsilon b \cdot p + \sum_{t=0}^{s} y_t \\ \text{s.t.} \quad & c_{tj} \cdot p + y_t \geq \pi_{tj}, j \in [1, N] \\ & p, y_t \geq 0, t \in [0, s]. \end{aligned} \tag{12}$$

Let $(\hat{p}, \hat{\boldsymbol{y}})$ represent the optimal solution to problem (12). For any given p, we define the strategy of adjusting the number of VMs as follows:

$$x_{tj}(p) = \begin{cases} 1, & \pi_{tj} > p \cdot c_{tj}, \\ 0, & \pi_{tj} < p \cdot c_{tj}. \end{cases} \tag{13}$$

Among all the elements of $\boldsymbol{x}_t$ that equal 1, we set them as 0 except which makes the value of $\pi_{tj} - p \cdot c_{tj}$ the maximum, i.e.,

$$x_{tj}(p) = \begin{cases} 1, & j = \arg\max_{j \in N}\{\pi_{tj} - p \cdot c_{tj}\}, \\ 0, & else. \end{cases} \tag{14}$$

Finally, we propose our algorithm as follows:

Algorithm 3 Capacity-Adjusting Algorithm

```
Initialize p̂ and max;
for each t = s+1, s+2, ..., m do
  for each j = 1, 2, ..., N do
    Let x_tj(p) = 1, if it makes π_tj − p·c_tj the maximum;
  end for
end for
```

Remark: In this algorithm, we first need to solve a small scale partial linear problem (11) and obtain the dual solution $(\hat{p}, \hat{\boldsymbol{y}})$. Then, during time slot $t \in \{s, s+1, ...m\}$, we use $\hat{p}$ learned from problem (12) to decide the VM adjusting strategy $\boldsymbol{x}_t$. Intuitively, when the scale of problem (12) is large, the constraint (10) can be enforced with high possibility. On the contrary, if the scale of the partial linear problem is small, $\hat{p}$ learned from it can hardly enforce the constraint (10). Hence, we now demonstrate the optimality of the algorithm as well as the correlation between the budget b and ϵ.

Assumption 1. *For any p, there can be at most 1 column of $\boldsymbol{\pi}_t \in \{\boldsymbol{\pi}_i | i \in [1, m]\}$, i.e., π_{tj}, such that $\pi_{tj} = p \cdot c_{tj}$ or $\pi_{tj} - p \cdot c_{tj} = \pi_{tl} - p \cdot c_{tl}$.*

Based on the proof in [29], with probability of 1, no p can satisfy 2 entries of $\boldsymbol{\pi}_t$ such that $\pi_{tj} = p \cdot c_{tj}$ or $\pi_{tj} - p \cdot c_{tj} = \pi_{tl} - p \cdot c_{tl}$ simultaneously by perturbing $\boldsymbol{\pi}_t$ with a random variable . Furthermore, the effect of this perturbation can be made arbitrarily small. Agrawal *et al.* have also made such an assumption and proposed an effective algorithm in [6].

Lemma 2. *For all $t \in [0, m]$, under Assumption 1, $\boldsymbol{x}_t(p^*)$ and $\boldsymbol{x}_t^*$ differs no more than 1 value of t.*

We prove the lemma in our technical report [22].

Remark: Lemma 2 demonstrates the gap between the optimal solution $\boldsymbol{x}_t(p^*)$ of the dual problem and the optimal solution $\boldsymbol{x}_t^*$ of the primal. However, in the algorithm, we use the solution $\hat{p}$ learned from the partial problem (11) instead of the optimal solution p^*. Hence, we next demonstrate that $\hat{p}$ is accurate enough as a substitute.

Lemma 3. *The primal solution derived using sample dual price $\hat{p}$ is a feasible solution to the linear problem (9) with high probability of $1 - \epsilon$ that*

$$\sum_{t=1}^{m} \boldsymbol{c}_t^T \boldsymbol{x}_t(\hat{p}) \leq b,$$

given that $b \geq \frac{3m \ln(N/\epsilon)}{\epsilon^3}$.

We prove the lemma in our technical report [22].

Remark: So far we showed that $\boldsymbol{x}_t(\hat{p})$ is a feasible solution, such that it can enforce $\sum_{t=1}^{m} \boldsymbol{c}_t^T \boldsymbol{x}_t(\hat{p}) \leq b$ with a probability of at least $1 - \epsilon$ on condition that $b \geq \frac{3m \ln(N/\epsilon)}{\epsilon^3}$. However, simply satisfying the constraint may make the objective value far deviate from the optimum. Therefore, we then show that $\boldsymbol{x}_t(\hat{p})$ is indeed a near-optimal solution.

Lemma 4. *The primal solution constructed using sample dual price $\hat{p}$ is a near-optimal solutions to the linear problem (12) with high probability of $1 - \epsilon$ that*

$$\sum_{t=1}^{m} \boldsymbol{\pi}_t^T \boldsymbol{x}_t(\hat{p}) \geq (1 - 3\epsilon)OPT,$$

given that $b \geq \frac{3m \ln(N/\epsilon)}{\epsilon^3}$.

We prove the lemma in our technical report [22].

Remark: Lemma 4 indicates that with probability of $1 - \epsilon$, the online solution $\boldsymbol{x}_t(\hat{p})$ taken over entire period $\{1, 2, ..., m\}$ is near optimal on condition that $b \geq \frac{3m \ln(N/\epsilon)}{\epsilon^3}$. Then we try to prove the competitive ratio of the Capacity-Adjustment Algorithm.

Proposition 1. *For any $\epsilon > 0$, the Capacity-Adjusting Algorithm is $1 - 6\epsilon$ competitive for the online linear problem (9).*

We prove the proposition in our technical report [22].

Remark: By now, we have proved that the capacity-adjusting algorithm can derive a competitive ratio of $1-6\epsilon$ against the optimal value such that the condition $b \geq \frac{3m\ln(N/\epsilon)}{\epsilon^3}$ holds. Intuitively, this condition may make b become so large that any $\boldsymbol{x}_t$ can enforce the constraint (10). However, in practice, cloud tenants always set the budget b in advance. Hence, the condition $b \geq \frac{3m\ln(N/\epsilon)}{\epsilon^3}$ can provide a guideline for cloud tenants to adjust ϵ and b.

V. Performance Evaluation

In this section, we evaluate the efficacy of our algorithms, i.e., Workload Distribution Algorithm (WDA), Service Degradation Algorithm (SDA), and Capacity-Adjusting Algorithm (CAA).

Performance metrics. We take the average response time as the key performance metric for WDA, SDA, and CAA. For SDA, postponed requests are scheduled to the asynchronous process, and hence we add the execution time of the postponed requests as another performance metric.

Performance-cost ratio. When evaluating CAA, we calculate the performance-cost (PC) ratio to judge whether it is cost-effective.

Budget-cost ratio. To evaluate whether CAA can control the outsourcing cost effectively, we consider the budget-cost (BC) ratio, which is the ratio between cost and budget, as a key metric.

A. Setup

1) Testbed: We build a private cloud on two servers with `OpenStack Mitaka` [7] under `nova-network` [8]. Each server has an Intel Xeon 2.6 Ghz (8 cores with hyper-threading) CPU and a 1 GbE NIC connected to a 1 GbE switch port. The servers run Linux 2.6.32 kernel, among which one acts as a controller node and the other acts as a compute node. At the same time, we rent 20 EC2 large type instances on AWS as the public cloud part, whose details are listed in Table II. The `OpenStack` platform provides easy access to manage the EC2 instances in the public cloud, which is further used to integrate the private and public resources into a hybrid cloud. As such, we build a hybrid cloud scenario.

TABLE II
Parameters of EC2 large instance

Type	vCPU	ECU	Memory	Storage (GB)	Pricing
Detail	4	13	16GiB	EBS Only	0.239 USD/Hour

In the evaluation, we use the online traffic in U.S. on Cyber Monday measured by Akamai [30]. Data was collected from U.S. retailers by Akamai's Net Usage Index (NUI) and Real User Monitoring (RUM), which is plotted in Fig. 4.

2) Implementation: We develop a 3-tiered website prototype to capture the characteristics of flash deal applications. All the three tiers are deployed in one VM, so that we can tune the capacity of the website horizontally by adding more VMs. In each VM, the web tier is deployed by an Apache HTTP server which is used to maintain HTTP connections and retrieve static files. Behind the web tier, we deploy two Tomcat 9.0 servers as the application tier, which equally share sessions and serve requests dispatched by APJ connector in Apache 2.4. These Tomcat servers execute a Servlet that queries records of a table from a MySQL database. Considering flash deals mostly involve query operations, we do not take data consistency into consideration in the database tier.

To produce HTTP requests for the website, we design an HTTP request generator with HttpClient 4.5.2 according to a function obtained by fitting the trace in Fig. 4. Each request is created as a thread and initialized with a timestamp of birth. Based on the current capacity of the hybrid cloud, a load balancer determines how to schedule the newly arrived requests according to WDA and SDA. Then the load balancer redirects the requests to each running VM. After scheduling requests, based on CAA, the load balancer also adjusts the number of running EC2 instances in AWS through EC2 API command line. By now one round of optimization is completed.

B. Evaluating Algorithms for Scheduling Requests

To evaluate the effectiveness of the algorithms for scheduling requests, we do not take capacity adjustment into consideration. Instead, we set the capacity of the hybrid cloud as a constant, so as to clearly demonstrate the impact brought by the algorithms.

As mentioned in Sec. IV-A, after determining how many requests to be served by the public cloud with the Workload Distribution Algorithm (WDA), we adopt the Service Degradation Algorithm (SDA) to further reduce interactive response time. For comparison, we set WDA as the baseline algorithm. Meanwhile, when leveraging SDA, we need to set the number of postponed requests in advance, i.e., α, which is introduced in Sec. IV-A. Here we set $\frac{\alpha}{\lambda} = 0.1, 0.2, 0.3$ and corresponding cases are SDA-0.1, SDA-0.2, and SDA-0.3, respectively. Actually, WDA is a special case of SDA whose $\alpha = 0$. Moreover, we set the deadline L as 2000ms.

Interactive response time. Fig. 5 plots the response time of the interactive process under different values of α. By scheduling partial requests to the asynchronous process and postponing serving them, the values of response time of SDA-0.1 are much lower than WDA's in general. Specifically, for WDA, 60% of values of response time exceed 500ms. Compared with SDA-0.3, only 6% of values of response time are over 500ms. Such a phenomenon implies that the website under soft guarantee can better get over with the spikes of flash crowds. Obviously, scheduling more requests to the asynchronous process can reduce response time remarkably. However, excessive postponed requests may lead to long execution time, making their response time exceed the predefined deadline L. Hence, we plot the CDF of response time in the asynchronous process in Fig. 6.

Response time of the asynchronous process. As shown in Fig. 6, for all the cases, as α increases from 0.1 to 0.3, the proportion of requests, whose response time is within 2000ms, goes down from 98% to 63% correspondingly. Due to

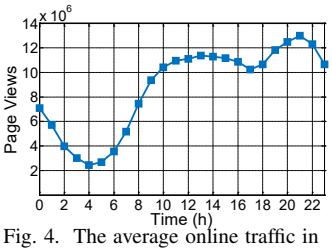

Fig. 4. The average online traffic in the U.S. on Cyber Monday.

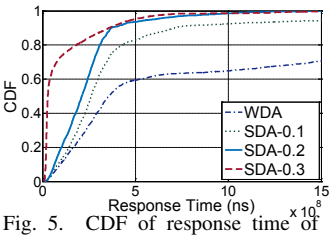

Fig. 5. CDF of response time of service degradation algorithms.

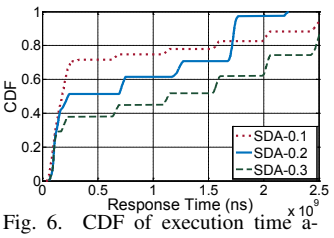

Fig. 6. CDF of execution time among postponed requests.

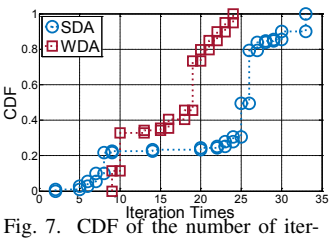

Fig. 7. CDF of the number of iterations of WDA and SDA.

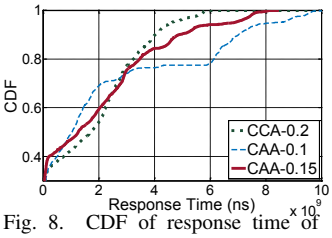

Fig. 8. CDF of response time of capacity-adjusting algorithms.

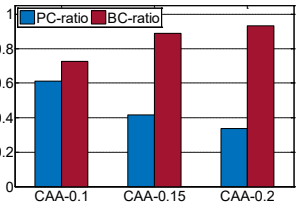

Fig. 9. PC and BC ratios of capacity-adjusting algorithms.

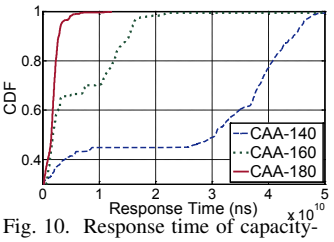

Fig. 10. Response time of capacity-adjusting algorithms.

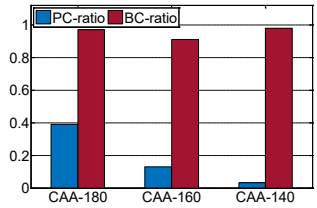

Fig. 11. PC and BC ratios of capacity-adjusting algorithms.

scheduling excessive requests to the asynchronous process, response time of SDA-0.3 is hardly controlled within predefined deadline, i.e., 2000ms. Interestingly, the trend of the lines in Fig. 6 shows distinct stages. It is because we attach different priorities to the postponed requests based on their sojourn time. Requests with longer sojourn time have a higher priority to get served, which leads to distinct classes of response time.

Convergence of WDA and SDA. Fig. 7 plots the CDF of the number of iterations to achieve convergence. It clearly shows that WDA can converge within 20 iterations and SDA is able to converge within 30 iterations for 80% of the respective total runs. Such results demonstrate the fast convergence of our solution.

C. Evaluating Capacity-Adjusting Algorithm

To evaluate the Capacity-Adjusting Algorithm (CAA), we tune ϵ and b to demonstrate its effectiveness. Furthermore, we also compare CAA with CEOA in [9] under different values of budget.

Impact of learning scale. To evaluate the impact of the learning scale of the partial linear problem, i.e. s, we set three cases of CAA on condition that budget keeps the same. The three cases are named as CAA-0.1, CAA-0.15 and CAA-0.2, where $\epsilon = \frac{s}{m} = 0.1, 0.15, 0.2$, respectively. Fig. 8 demonstrates the interactive response time of CAA under different values of ϵ. In general, around 75% of values of response time are less than 3500ms among the three cases. For the requests whose response times exceed 3500ms, as ϵ increases, response time goes down. Based on the analysis in Sec. IV-B, given a smaller ϵ, the partial linear problem has a lower budget. Hence, the price vector p learnt from the partial problem indicates the online version is under a tight budget, making response time increase slightly.

Fig. 9 plots the PC and BC ratios of CAA under different values of ϵ, respectively. As ϵ decreases, the PC ratio increases correspondingly. Such a trend is accordance with the competitive ratio of $1 - 6\epsilon$ derived in Proposition 1. Moreover, with ϵ increasing, the cost incurred by CAA decreases and stays within the budget. Such a fact verifies the conclusion in Lemma 3, i.e., the outsourcing cost can be controlled under the budget with a probability of $1 - \epsilon$.

Impact of budget. To evaluate the impact of budget, we set budget $b = 140(\$), 160(\$), 180(\$)$ and name the corresponding cases as CAA-140, CAA-160, and CAA-180, respectively. Fig. 10 plots the response time of CAA under different values of budget. When the budget increases from 140 to 180, the response time decreases significantly. In addition to performance, we plot the PC and BC ratios of CAA in Fig. 11. As shown in Fig. 11, with b increasing, the PC ratio goes up, which verifies Lemma 3 and Lemma 4. To obtain a competitive ratio of $1 - 6\epsilon$, the budget should satisfy the condition that $b \geq \frac{3m \ln(N/\epsilon)}{\epsilon^3}$. As a result, a large budget allows a smaller ϵ, which contributes to a more near-optimal value. In addition, based on the formulation of the partial linear problem (11), a larger budget is identical to a smaller ϵ, which makes the PC ratio approaching the optimum. In respect of cost, all the BC ratios are around 1, indicating that CAA is effective in controlling cost.

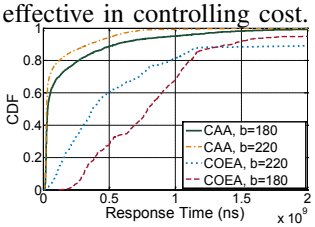

Fig. 12. CDF of response time of CAA and CEOA.

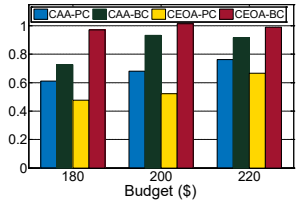

Fig. 13. PC and BC ratios of CAA and CEOA.

Comparison with CEOA. We next compare CAA with CEOA [9] under different values of budget. Fig. 12 plots the response time of CAA and CEOA when budget equals 180($) and 220($). We observe that when budget increases, the response time of CAA and CEOA decreases, respectively. Furthermore, under the same budget, the values of response time of CAA are lower than CEOA's.

Fig. 13 illustrates the PC and BC ratios of CAA and CEOA under different values of budget. As shown in Fig. 13, when budget increases, the BC ratio of CEOA can be maintained around 1, which means that CEOA can effectively control cost with Lyapunov optimization techniques. However, the values of PC ratio are lower than CAA's, since the trade-off between

cost and performance is driven by a parameter V in CEOA. Through Lyapunov optimization techniques, CEOA minimizes the sum of cost and response time under a given value of V, instead of maximizing the PC ratio. Compared with CEOA, CAA reduces response time by 15% and improves the PC ratio by 19% on average, respectively.

VI. Related Work

To improve the performance of web applications, Kamra *et al.* present queue-based solutions that effectively bound the response time, together with an admission controller [15]. The focus of our paper is on improving the response time for flash deal applications in a hybrid cloud. Based on SQP methods, we design WDA and SDA for scheduling requests. Abdelzaher *et al.* design a system for QoS management with graceful service degradation [31]. To prevent web server from being overloaded, they reduce the number of embedded objects per page. We instead postpone serving delay-tolerant requests to improve the performance of interactive processes.

Complementary to existing work [32], [33], hybrid cloud is a promising solution to handle flash crowds, where cloud tenants outsource excessive workload to a public cloud as well as own a private cloud. Recent works have commonly focused on provisioning a cost-effective solution. In [9], Niu *et al.* study flash crowds of common e-commerce websites in a hybrid cloud. Different from [9], we focus on flash deal applications, where subscribers can accept service degradation. By postponing serving requests, our solution not only reduces response time but also controls cost within budget. Apart from motivation, our online algorithm also has a competitive ratio, which is stronger than the $O(V, \frac{1}{V})$ tradeoff achieved in [9].

VII. Conclusion

In this paper, we proposed a solution for flash deal applications to withstand flash crowds in a hybrid cloud. To achieve a cost-effective hybrid cloud solution under soft guarantee, we jointly handled request-scheduling and capacity-adjusting problems. Concerning scheduling requests, we achieved fast response time of the interactive process as well as guaranteed requests served in the asynchronous process within a predefined deadline. In terms of adjusting capacity, we tuned scale of the public cloud with the objectives of performance-cost ratio maximization as well as outsourcing cost minimization. By adapting an online learning algorithm, we obtained a competitive ratio of $1 - O(\epsilon)$ against the offline optimal solution. Extensive trace-driven experiments on a hybrid cloud platform showed that compared with previous work, our solution reduced response time by 15% on average and effectively maintained cost within the budget.

References

[1] Amazon Says Prime Day Was Bigger Than Black Friday And Will Be Held Again.

[2] Apple iPhone 6 preorders start at midnight, but problems plague Apple Store, other sites. [Online]. Available: http://www.cnet.com/news/apple-iphone-6-preorders/

[3] WeChat red envelope. [Online]. Available: https://en.wikipedia.org/wiki/WeChat_red_envelope

[4] "State of the cloud report," Report, 2015 RightScale, Inc., Feb. 2015. [Online]. Available: http://www.rightscale.com/lp/2015-state-of-the-cloud-report

[5] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, 1 1995.

[6] S. Agrawal, Z. Wang, and Y. Ye, "A dynamic near-optimal algorithm for online linear programming," *Operations Research*, vol. 62, no. 4, pp. 876–890, 2014.

[7] OpenStack Mitaka. [Online]. Available: http://www.openstack.org/software/mitaka/

[8] Nova Network. [Online]. Available: http://docs.openstack.org/admin-guide/compute-networking-nova.html

[9] Y. Niu, B. Luo, F. Liu, J. Liu, and B. Li, "When Hybrid Cloud Meets Flash Crowd: Towards Cost-Effective Service Provisioning," in *Proc. of IEEE INFOCOM*, 2015.

[10] Amazon EC2. [Online]. Available: http://aws.amazon.com/ec2/

[11] Auto Scaling. [Online]. Available: http://aws.amazon.com/autoscaling/

[12] Manual Scaling. [Online]. Available: http://docs.aws.amazon.com/autoscaling/latest/userguide/as-manual-scaling.html

[13] Y. Diao, J. Hellerstein, S. Parekh, H. Shaikh, M. Surendra, and A. Tantawi, "Modeling differentiated services of multi-tier web applications," in *Proc. of IEEE MASCOTS*, 2006.

[14] D. A. Menascé, D. Barbará, and R. Dodge, "Preserving qos of e-commerce sites through self-tuning: A performance model approach," in *Proc. of ACM EC*, ser. EC '01, 2001.

[15] A. Kamra, V. Misra, and E. Nahum, "Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites," in *Proc. of IWQOS*, 2004.

[16] D. Jiang, G. Pierre, and C.-H. Chi, "Autonomous resource provisioning for multi-service web applications," in *Proceedings of WWW*, 2010.

[17] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, "Performance management for cluster-based web services," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 12, pp. 2333–2343, Dec 2005.

[18] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," *ACM Trans. Internet Technol.*, vol. 7, no. 1, Feb. 2007.

[19] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Trans. Auton. Adapt. Syst.*, vol. 3, no. 1, pp. 1:1–1:39, Mar. 2008.

[20] Y. Diao, J. Hellerstein, S. Parekh, H. Shaikh, and M. Surendra, "Controlling quality of service in multi-tier web applications," in *Proc. of IEEE ICDCS*, 2006.

[21] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," *SIGMETRICS Perform. Eval. Rev.*, 2005.

[22] Technical Report. [Online]. Available: https://www.dropbox.com/s/ov5uo88nmygbg7a/TechnicalReport.pdf?dl=0

[23] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[24] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proc. of IEEE Cloud Computing (CLOUD)*, 2012.

[25] AWS Direct Connect. [Online]. Available: http://aws.amazon.com/directconnect/

[26] S. Ross, *Stochastic processes*. Wiley, 1996.

[27] R. Fletcher, *Practical Methods of Optimization; (2Nd Ed.)*. New York, NY, USA: Wiley-Interscience, 1987.

[28] S. Han, "A globally convergent method for nonlinear programming," *Journal of Optimization Theory and Applications*, vol. 22, no. 3, pp. 297–309, 1977.

[29] N. R. Devanur and T. P. Hayes, "The adwords problem: Online keyword matching with budgeted bidders under random permutations," in *Proc. of ACM EC*, 2009.

[30] "Akamai's 2014 online holiday shopping trends and traffic report," Report, 2016 Akamai Technologies, Inc., 2014. [Online]. Available: https://content.akamai.com/PG2112-Holiday-Recap-Report.html

[31] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: A control-theoretical approach," *IEEE TPDS*, vol. 13, no. 1, pp. 80–96, Jan. 2002.

[32] B. Li, G. Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin, "An empirical study of flash crowd dynamics in a p2p-based live video streaming system," in *IEEE GLOBECOM*, 2008.

[33] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao, "Flash crowd in p2p live streaming systems: Fundamental characteristics and design implications," *IEEE TPDS*, vol. 23, no. 7, pp. 1227–1239, July 2012.