When Hybrid Cloud Meets Flash Crowd: Towards Cost-Effective Service Provisioning

Yipei Niu[1] Bin Luo[1] Fangming Liu*[1] Jiangchuan Liu[2] Bo Li[3]
[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]Simon Fraser University, Canada [3]The Hong Kong University of Science and Technology, Hong Kong.

***Abstract*—With rapid development in online shopping, e-commerce websites are facing intensive user requests from an increasing number of customers. Especially in promotion seasons, these websites may encounter flash crowds which pull heavy pressure to private infrastructure and even make the website unavailable. Such severe flash crowds can be addressed by leveraging hybrid cloud solution, which relieves workloads of the private cloud by offloading the excessive user requests to the IaaS public cloud. However, the bursty and fluctuation of flash crowds bring challenges to distributing user requests with targets of delay-minimizing and cost-saving. In this paper, we apply the queueing theory to evaluate the average response time and explore the tradeoff between performance and cost in the hybrid cloud. By taking advantage of Lyapunov optimization techniques, we design an online decision algorithm for request distribution which achieves the average response time arbitrarily close to the theoretically optimum and controls the outsourcing cost based on a given budget. The simulation results demonstrate that in a hybrid cloud, our solution can reduce the cost of e-commerce services as well as guarantee performance when encountering flash crowds.**

I. Introduction

Recently, e-commerce websites, which provision cheaper commodities and instant shopping, have become prevalent in our daily lives. In consequence of attracting sale discount, e-commerce websites see surges in page visits and deals during promotion seasons. Based on the statistics data of Double Eleven Shopping Festival in 2012, Alibaba, the biggest e-commerce provider in China, claimed that its website encountered a spike of over 13,000 request orders per second [1], which is much larger compared to the workloads in normal days. Such a large scale of page visits and bill transactions will definitely bring considerable profit of advertising and sale. For instance, on November 11th 2013, Taobao, the largest e-commerce website in China, witnessed a record that total sale reached amazing 5.7 billion USD in 24 hours [2]. Moreover, the sale of Double Eleven promotion contributes more than half of the annual revenue (USD 7.5 billion) of Alibaba. In short, it is critical for an e-commerce website to withstand flash crowds, in which there could be a much larger number of customers visiting the website and dealing with merchants simultaneously in a short period of time.

Under the severe flash crowds, a private cloud, which is supported by dedicated on-premise infrastructure, has to maintain a large number of TCP connections and retrieve various types of static files for web page presentation. At the same time, it also needs to process a tremendous amount of queries for dynamic data. As a result, it is hard for the website to handle the workloads on time with the limited local resources in the private cloud. Yet a considerable amount of the requests are related to confidential information of users, e.g., the account number of credit card, the password and so on; as a common practice, these should be processed by the private cloud to ensure data security. On the other hand, connected by a dedicated tunnel, for instance, AWS Direct Connect [3], a private cloud can outsource the excessive workloads to an IaaS public cloud. A public cloud, for instance, Amazon EC2, which provisions elastic and instant computing resources, can adjust its capacity based on users' computing requirements. As shown in Fig. 1, requests come from thousands of users who pose waves of workloads to the e-commerce website. The requests will be transmitted to the private cloud; however, the private cloud is unable to handle such severe flash crowds. As a result, the excessive requests will be redirected to the public cloud, e.g., Amazon EC2, via a dedicated tunnel, e.g., the AWS Direct Connect.

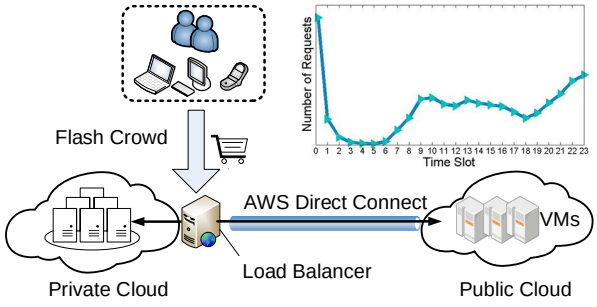

Fig. 1. Overview of e-commerce website deployed in hybrid cloud

A hybrid cloud, which contains a private cloud and a public cloud, offers a potentially ideal solution to deal with flash crowds, particularly for e-commerce websites. Fully exploring the concept of "owning the base and renting the peak" [4] however remains challenging in practice. First, due to the bursty and fluctuation of flash crowds, it is difficult for an e-commerce website to predict the exact workloads of next second; that is, the e-commerce website is blind about the workload distribution. Such unpredictability makes it challenging to leverage a hybrid cloud in a cost-effective way. Second, for an e-commerce website, performance must come in the first place. As a result, how to distribute workloads to enable the e-commerce website to provision the best services is critical.

*The Corresponding Author is Fangming Liu (fmliu@hust.edu.cn). The research was support in part by a grant from National Basic Research Program (973 program) under grant No.2014CB347800.

On the other hand, how to distribute workloads to indicate the public cloud to scale up or down to a proper capacity also faces great challenges. Third, when provisioning high quality services, the outsourcing cost spent on deploying the public cloud and the tunnel must be carefully controlled. In conclusion, it is a challenging problem for an e-commerce website to distribute workloads between the private and the public clouds so as to provision cost-effective services in the hybrid cloud.

In this paper, we systematically examine the challenges when using a hybrid cloud to accommodate flash crowds. We present a comprehensive analytical framework that captures the essentials of the interplay among the different modules in a hybrid cloud system for e-commerce. By taking advantage of Lyapunov optimization techniques, we design a cost-effective online algorithm (**CEOA**) for request distribution. The proposed online algorithm makes real time decision on workloads distributing and virtual machine (VM) scaling without *a priori* knowledge of future workload arrival. It achieves the average response time arbitrarily close to the theoretically optimum and controls the outsourcing cost based on a given budget. Our trace-driven simulation results demonstrate that, under the hybrid cloud scenario, our solution can reduce the outsourcing cost as well as provision high quality services when encountering flash crowds.

II. System Model

A. E-commerce Web Application

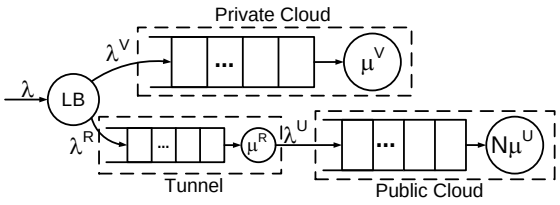

Fig. 2. The model of e-commerce website in hybrid cloud based on the queueing theory

Modern e-commerce websites mostly involve three tiers, namely, a front-end tier, a business-logic tier and a back-end tier. The front-end tier is deployed by web servers (e.g., Apache) that are responsible for receiving HTTP requests and retrieving static files. The business-logic tier implements key functionalities of the website. The back-end tier processes queries and transactions of the database [5][6]. It has been shown that such a multi-tier architecture can be abstracted as a single tier for performance analysis [6]; such an abstraction captures the essential functions of e-commerce websites, and facilitates our study on distributing flash crowds between the private cloud and the public cloud.

Like in previous studies, in the single-tier architecture website, we assume that requests arrive at the website as a Poisson process. The time interval between two requests that are served is assumed to be exponentially distributed [7], so that we can take the M/M/1 queue [8] to model how each part of the hybrid cloud functions [6]. As a result, we can estimate the average delay of serving requests, i.e., the average response time, including service time and waiting delay.

TABLE I. Key Parameters

Notation	Definition
$\lambda(t)$	Average arrival rate of requests during the tth time slot
$\mu(t)$	Average service rate during the tth time slot
P_n	Probability that the queueing system contains n requests
$\eta(t)$	Proportion of total requests assigned to the public cloud ($\frac{\lambda^R(t)}{\lambda(t)}$)
α_u	Upper threshold of scaling up
α_d	Lower threshold of scaling down
$\alpha(t)$	Average CPU utilization during the tth time slot
$S(t)$	Number of launched or removed EC2 instances in the scaling action
$N(t)$	Number of running EC2 instances in service during the tth time slot
$D(t)$	Average delay of serving requests in a hybrid cloud
$F(t)$	Total cost of deploying hybrid cloud service
$X^{U,V,R}$	Parameter X of the public cloud(U), the private cloud(V), and the tunnel(R)

*1) **Modeling the Public Cloud**:* In public cloud market, there are many public cloud providers, whose computing resources vary from each other. For instance, Amazon EC2 allows users to rent VMs on which to run their own applications. Amazon EC2 allows scalable deployment of applications by providing a web service through which a user can boot an Amazon Machine Image to create a VM called "EC2 instance", containing any software desired. Rackspace [9] public cloud provisions a scalable, reliable, high-performing cloud environment, too. Specifically, all the softwares of Rackspace are developed by an open source project named "OpenStack", which is a very popular framework. Moreover, there are also some other cloud providers, such as Windows Azure [10], VMWare vCloud Hybrid Service [11] and so on.

In this paper, we take Amazon EC2 as a representative of public clouds to provision elastic and instant computing services of the hybrid cloud. Amazon EC2 is characterized by its auxiliary service named AWS Auto Scaling [12] that can adjust the number of running EC2 instances automatically. AWS Auto Scaling can monitor various types of performance metrics of running instances, for example, CPU cycle utilization. Based on the value of the monitored metric, AWS Auto Scaling decides whether to adjust the number of the running EC2 instances. Before applying AWS Auto Scaling, users need to set a series of conditions and a scaling policy in advance.

First, Amazon EC2 claims that the scaling action mustn't happen too frequently, so the least time interval between two consecutive scaling actions can be configured. Given the time interval between two consecutive scaling actions is constant, we consider a discrete time model where a time slot length matches the timescale which equals the least time interval of adjusting the cloud capacity. Since the time spent on starting up EC2 instances varies from their different OS types, capacity levels, we assume that there exists a time interval of interest $t \in \{0, 1, ..., T-1\}$, which equals the least duration between two consecutive scaling actions and is larger than the average startup delay.

Second, a threshold of scaling must be set based on the customized type of the performance metric. For instance, Amazon EC2 scales up, i.e., increases the number of running instances, when the chosen metric, e.g., CPU cycle utilization, exceeds the set threshold. Here we select the average CPU cycle utilization as the monitored metric, which can be denoted

as $\alpha(t)$. Correspondingly, the threshold of scaling up or down can be α_u or α_d, respectively. So the action of scaling can be formulated as follows:

$$S(t) = \begin{cases} m, & \alpha(t) \geq \alpha_u \\ 0, & \alpha_d \leq \alpha(t) \leq \alpha_u \\ -n, & \alpha(t) < \alpha_d \end{cases}$$

where the number of EC2 instances in once scaling up or down can be set, which can be denoted as m and n respectively. When $\alpha(t)$ exceeds α_u, AWS Auto Scale will start up m EC2 instances. On the contrary, when $\alpha(t)$ drops below α_d, AWS Auto Scale will remove n running EC2 instances. When $\alpha(t) \in [\alpha_d, \alpha_u]$, the number of EC2 instances remains unchanged.

Finally, users need to set the minimum size and the maximum size of an auto scaling group, in which the number of running EC2 instances must be adjusted within the range from the minimum size to the maximum size. However, when outsourcing workloads are redirected to the public cloud, it is possible that the workloads are too heavy to handle for the finite auto scaling group. So users need to set a large number as the maximum size of the auto scaling group. The number of the running EC2 instances during the tth time slot can be denoted as follows:

$$N(t) = N(t-1) + S(t-1). \tag{1}$$

where $N(t)$ must satisfies the following constraint:

$$N_{min} \leq N(t) \leq N_{max}. \tag{2}$$

Moreover, N_{min} and N_{max} are the minimum size and the maximum size of the auto scaling group, respectively.

After modeling the real IaaS public cloud, we take a look at the model of the e-commerce website deployed in the public cloud. As mentioned above, we regard the part in the public cloud as a single unit which can complete all the functions offloaded from the private cloud. Hence, an M/M/1 queue [13] is used to evaluate the average response time in the public cloud, which can be denoted as

$$D^U(t) = \frac{1}{N(t)\mu^U - \lambda^U(t)}, \tag{3}$$

where μ^U is the constant service rate of each EC2 instance and $\lambda^U(t)$ is the average arrival rate of requests redirected to the public cloud during the tth time slot. In order to provision high quality services, Eq. (3) must satisfy the following constraint

$$\lambda^U(t) < N(t)\mu^U. \tag{4}$$

For simplicity, we assume that the CPU utilization reaches the maximum value when an EC2 instance is busy and drops to the minimum value when it is idle. As a result, the average CPU cycle utilization during the tth time slot can be denoted as

$$\alpha(t) = P_0(t) \cdot \beta_{min} + (1 - P_0(t))\beta_{max}. \tag{5}$$

where β_{min} and β_{max} are the minimum utilization and the maximum utilization of CPU respectively.

*2) **Modeling the Private Cloud**:* The private cloud is modeled as an M/M/1 queue here, too. In the queueing system, as mentioned in the public cloud modeling, we also enforce the following constraint:

$$\lambda^V(t) < \mu^V, \tag{6}$$

where μ^V is the average service rate of the private cloud. The value of μ^V is constant, which means that the capacity of the private cloud is finite.

According to Little's Law, the average response time, i.e., the average time spent on serving requests in the private cloud, can be denoted as

$$D^V(t) = \frac{1}{\mu^V - \lambda^V(t)}. \tag{7}$$

*3) **Modeling the Tunnel**:* AWS Direct Connect [3] is a dedicated network connection from a user's premises to Amazon EC2. Using the AWS Direct Connect, the e-commerce website can establish a private connection between Amazon EC2 and the on-premise infrastructure, which in many cases can reduce the network costs, increase the bandwidth throughput, and provide a more consistent network experience than Internet-based connections. The AWS Direct Connect can be used to redirect requests, transmit files and even database queries. As such, when a flash crowd hits, excessive requests will be redirected to the public cloud, and the private cloud will not establish TCP connections between users and the private cloud. In other words, as long as requests are assigned to the public cloud, the public cloud will take full charge of the request serving process.

TABLE II. AVAILABLE LEVELS OF TUNNEL

level	1	2	3	4	5	6	7
bandwidth(Mbps)	50	100	200	300	400	500	10,000
pricing(USD/hour)	0.03	0.06	0.12	0.18	0.24	0.30	2.25

The tunnel is priced based on different levels of bandwidth $l \in \{0, 1, ..., L\}$, as shown in Table II. Correspondingly, the capability of l-level tunnel can be denoted as μ_l^R and $\mu_l^R \in \{\mu_0^R, \mu_1^R, ..., \mu_L^R\}$. Meanwhile, fee of renting l-level tunnel can be denoted as k_l and $k_l \in \{k_0, k_1, ..., k_L\}$. As a result, fee of renting the tunnel during the tth time slot can be denoted as $K(t) = \sum_{l=0}^{L} k_l x_l(t)$ and the capability of the tunnel during the tth time slot can be denoted as $\mu^R(t) = \sum_{l=0}^{L} \mu_l^R x_l(t)$, where $x_l \in \{0, 1\}$. The tunnel is also modeled as an M/M/1 queue; hence, the queueing delay, i.e., the average delay of transmitting requests, can be denoted as

$$D^R(t) = \frac{1}{\mu^R(t) - \lambda^R(t)}, \tag{8}$$

where $\lambda^R(t)$ is the average arrival rate of requests that are redirected to the public cloud via AWS Direct Connect and $\mu^R(t)$ is the finite transmitting capability of the tunnel. With the surge of workloads, tunnel may be incapable of transmitting requests timely, which may limit the number of EC2 instances. When such a problem happens, it means that performance of the hybrid cloud is constrained and a larger capacity of the tunnel

is needed. Given that the highest-level tunnel has a capacity of $10Gbps$, we assume that the capability of the tunnel is large enough, which leads to the following constraint:

$$\lambda^R(t) < \mu^R(t). \tag{9}$$

As illustrated in Fig. 2, the request leaving rate of the tunnel $\mu^R(t)$ is the arrival rate of the public cloud $\lambda^U(t)$. When the queueing system does not serve any request, the leaving rate of requests is zero. Meanwhile, the service rate of the tunnel, i.e., $\mu^R(t)$, can be considered as the request arrival rate of the public cloud, when the tunnel is busy. Therefore, the request arrival rate of the public cloud can be denoted as

$$\lambda^U(t) = 0 \cdot P_0(t) + \mu^R(t)(1 - P_0(t)) = \lambda^R(t), \tag{10}$$

where $P_0(t)$ is the probability that the website doesn't contain any request. And $P_0(t) = 1 - \frac{\lambda^R(t)}{\mu^R}$.

B. Load Balancing for Provisioning Cost-Effective Services

As mentioned in Sec. I, the main task of the load balancer is to decide how many requests are to be redirected to the public cloud, so that the whole website can provision cost-effective services. Furthermore, the outsourcing cost should be spent as less as possible. The requests arrived at the website can be divided into two parts. One is assigned to the private cloud, the other to the public cloud, which can be formulated as follows:

$$\lambda(t) = \lambda^V(t) + \lambda^R(t), \tag{11}$$

where $\lambda(t)$ is the average request arrival rate during the tth time slot. As a result, the average response time of requests during the tth time slot can be denoted as

$$D(t) = \frac{\lambda^V(t)}{\lambda(t)} D^V(t) + \frac{\lambda^R(t)}{\lambda(t)} (D^R(t) + D^U(t)). \tag{12}$$

When a flash crowd hits, the e-commerce website deploys more EC2 instances to handle it. The cost of renting EC2 instances during the tth time slot can be denoted as:

$$I(t) = AN(t), \tag{13}$$

where A is the price of renting one running EC2 instance.

Meanwhile, when deploying a tunnel, the e-commerce website also needs to rent a proper level of bandwidth and maintain the minimum delay of transmitting requests. The cost of renting the tunnel during the tth time slot can then be denoted as

$$K(t) = \sum_{l=0}^{L} k_l x_l(t). \tag{14}$$

In order to provision cost-effective services, the website needs to spend outsourcing cost as less as possible. Commonly, the website will set a time-averaged budget M. The constraint on the time-averaged outsourcing cost of deploying the hybrid cloud can be enforced as follows:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} (AN(t) + AS(t) + \sum_{l=0}^{L} k_l x_l(t)) \leq M. \tag{15}$$

From the perspective of performance, we take $D(t)$, i.e., average response time, as the performance metric of the website. On the other hand, the time-averaged outsourcing cost should stay below the budget. So the optimization problem can be formulated as follows:

$$\begin{aligned} \min \quad & \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} D(t) \\ \text{s.t.} \quad & \text{constraints (2) (4) (6) (9) (15)} \end{aligned} \tag{16}$$

For the real hybrid cloud scenario, there exists a potential conflict when handling this optimization problem: due to the bursty and fluctuation of flash crowds, it is difficult to predict the workloads of next time slot. How to distribute workloads between the private cloud and the public cloud to make the website provision cost-effective services in a long period of time remains a challenging problem.

III. Cost-Effective Requests Distributing Online Algorithm

Flash crowds are featured by their bursty and unpredictability; hence, the website is generally unaware of how workloads in next second will be. Furthermore, the e-commerce website provisions instant services, which means that it has to react promptly to the upcoming workloads. As a result, an online algorithm is needed for the website. As mentioned in Sec. II-B, the objective is to minimize the average response time. The constraint (15) however implies that the outsourcing cost must not exceed the budget. During the tth time slot, the outsourcing cost increases or decreases a little, which seems like a queue with someone coming or leaving. As a result, a closer examination on the constraint suggests that the inequality (15) can be modeled as a queue. When the increasing part approximately equals the decreasing part, the outsourcing cost, i.e., the length of the queue, will be stable. Therefore, by optimizing the upper bound of the outsourcing cost, we can control the outsourcing cost below the budget. This leads to Lyapunov optimization techniques [14] for solving the problem (16).

A. Problem Transformation Using Lyapunov Optimization

Based on the analysis above, to capture the states of the outsourcing cost and the budget, we introduce a virtual queue $Q(t)$, which is used to accumulate the part of the outsourcing cost that exceeds the budget. Initially, we define $Q(0) = 0$ and then update the queue as follows:

$$\begin{aligned} Q(t+1) = \max\{&Q(t) + AS(t) \\ &+ AN(t) + \sum_{l=0}^{L} k_l x_l(t) - M, 0\}. \end{aligned}$$

where $S(t)$,$\sum_{l=0}^{L} k_l x_l(t)$, M and $N(t)$ are defined in Sec. II. And $Q(t)$ can be obtained from the backlog of the virtual queue. In fact, the constraint (15) enforces that the virtual queue is stable, i.e., $\lim_{T\to\infty} \frac{Q(t)}{T} = 0$.

Proof: From equation (17), we have

$$Q(t+1) \geq Q(t) + AS(t) + \sum_{l=0}^{L} k_l x_l(t) + AN(t) - M \tag{17}$$

Summing up both sides of the inequality (17) over time slots $t \in \{0, T-1\}$, and then dividing T, we have

$$\frac{Q(T-1)-Q(0)}{T} \geq \frac{1}{T}\sum_{t=0}^{T-1} AS(t) + \frac{1}{T}\sum_{t=0}^{T-1}\sum_{l=0}^{L} k_l x_l(t) + \frac{1}{T}\sum_{t=0}^{T-1} AN(t) - M.$$

Finally, making $T \to \infty$, and applying $Q(0) = 0$, we have

$$\lim_{T\to\infty} \frac{Q(T-1)}{T} \geq \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1}(AS(t) + \sum_{l=0}^{L} k_l x_l(t) + AN(t)) - M. \quad (18)$$

Applying the constraint (2) and (15) to inequality (18), we can conclude that $\lim_{T\to\infty} \frac{Q(t)}{T} = 0$ means the virtual queue $Q(t)$ is stable. When the length of the virtual queue is large, it means that the current outsourcing cost has far exceeded the budget. As a result, there is little probability for the e-commerce website provisioning cost-effective services. ■

1) Bounding 1-slot Lyapunov Drift: Taking advantage of Lyapunov optimization techniques, we define a Lyapunov function as a scalar measure of congestion in the system. The larger value of the Lyapunov function is, the more the outsourcing cost is. As a result, the constraint (15) is hard to be enforced, i.e., the outsourcing cost will exceed the budget. Specifically, we define the following Lyapunov function:

$$L(Q(t)) = \frac{1}{2}Q^2(t). \quad (19)$$

To keep the virtual queue stable, i.e., keep the outsourcing cost staying below the budget, we need to push the Lyapunov function towards a lower congestion state, which means that the website seeks to prevent the virtual queue increasing too much in next time slot. In this way the website can keep the virtual queue in a bounded state; in other words, control the outsourcing cost. We define the conditional 1-slot Lyapunov drift as follows:

$$\Delta(Q(t)) = \mathbb{E}\{L(Q(t+1)) - L((Q(t))|Q(t)\}. \quad (20)$$

Insight: By defining the Lyapunov function and the conditional 1-slot Lyapunov drift, we can obtain a metric of the queue's stability, i.e., $L(Q(t))$, and a tool of keeping the queue staying in a stable state, i.e., $\Delta(Q(t))$. We now can use them to enforce the constraint (15) during the process of optimization.

We first calculate the upper bound of $\Delta(Q(t))$, and try to make the bound closer to $\Delta(Q(t))$.

Lemma 1: For any $t \in \{0, T-1\}$, given any possible control decision, the Lyapunov drift $\Delta(Q(t))$ can be deterministically bounded as follows:

$$\Delta(Q(t)) \leq B + Q(t)\mathbb{E}\{AS(t) + \sum_{l=0}^{L} k_l x_l(t) + AN(t) - M|Q(t)\}.$$

Proof: Apply Eq. (19) to Eq. (20), we have

$$\Delta(Q(t)) = \mathbb{E}\{L(Q(t+1)) - L((Q(t))|Q(t)\} = \frac{1}{2}\mathbb{E}\{Q^2(t+1) - Q^2(t)|Q(t)\}.$$

Noting that $\max^2\{a, 0\} \leq a^2$, we have

$$Q^2(t+1) \leq Q^2(t) + (AS(t) + \varphi(t))^2 + 2Q(t)(AS(t) + \varphi(t)),$$

where $\varphi(t) = AN(t) + \sum_{l=0}^{L} k_l x_l(t) - M$. Hence, we have

$$\Delta(Q(t)) = \frac{1}{2}\mathbb{E}\{Q^2(t+1) - Q^2(t)|Q(t)\} \leq \mathbb{E}\{Q(t)(AS(t) + \varphi(t)) + \frac{1}{2}(AS(t) + \varphi(t))^2|Q(t)\}.$$

Applying the constraint (2), $S(t) \leq m$ and $\sum_{l=0}^{L} k_l x_l(t) \leq k_{max}$, we have

$$\Delta(Q(t)) \leq B + Q(t)\mathbb{E}\{AS(t) + \sum_{l=0}^{L} k_l x_l(t) + AN(t) - M|Q(t)\}, \quad (21)$$

where $B = \frac{1}{2}(Am + AN_{max} + k_{max} - M)^2$. ■

2) Bounding Drift-Plus-Performance: After bounding the 1-slot Lyapunov drift, we can minimize this upper bound to make the queue stable, so as to enforce the constraint (15). Following the Lyapunov optimization framework, we add a penalty term to both sides of inequality (21). The underlying objective of our optimal control decisions is to minimize the bound on the following drift-plus-performance expression in each time slot:

$$VD(t) + \Delta(Q(t)) \leq B + VD(t) + Q(t)\mathbb{E}\{AS(t) + \sum_{l=0}^{L} k_l x_l(t) + AN(t) - M|Q(t)\}. \quad (22)$$

Insight: Now we can minimize the average response time and enforce the constraint (15). At the same time, we have a parameter V; by tuning it, we can choose which objective we need to emphasize. In order to make the website provision cost-effective services, we just need to apply a proper value of V and minimize the upper bound of drift-plus-performance. Hence, the problem (16) can be transformed as follow:

$$\begin{aligned} \min\ & VD(t) + Q(t)(AS(t) + \sum_{l=0}^{L} k_l x_l(t) + AN(t) - M) \quad (23) \\ \text{s.t.}\quad & \text{constraints (2) (4) (6) (9)} \end{aligned}$$

Insight: Till now, we have transformed the long-term optimization problem to the problem above in each time slot. During each time slot $t \in \{0, 1, ..., T-1\}$, we minimize the sum of the outsourcing cost and the penalty of the average response time. And V is a control knob to adjust our emphasis on the outsourcing cost compared to the average response time, which finally enables the website to provision cost-effective services during a long period of time.

B. Optimal Analysis

We now analyze the optimality of the proposed one time slot optimization problem above, in terms of a tradeoff between the average response time and the outsourcing cost. For an algorithm with any fixed parameter V such that $V > 0$, independent of the current queue backlogs, it yields the following steady state values during any time slot $t \in \{0, 1, ..., T-1\}$:

$$\frac{1}{T}\sum_{t=0}^{T-1} D(t) = P^*, \tag{24}$$

where P^* is the optimal response time in theory. Based on the constraint (15), we have

$$\mathbb{E}\{AN(t) + AS(t) + \sum_{l=0}^{L} k_l x_l(t)\} \leq M. \tag{25}$$

Therefore, there must exist an ϵ, which can transform the inequality (25) to the following form:

$$\mathbb{E}\{AN(t) + AS(t) + \sum_{l=0}^{L} k_l x_l(t)\} \leq M - \epsilon, \tag{26}$$

where $\epsilon > 0$.

The problem (23) implies that we need to make a decision to minimize right side of inequality (22). By applying P^* to inequality (22), and summing up the inequality (22) over time slots $t \in \{0, 1, ..., T-1\}$, and then dividing both sides by T, we have

$$\frac{V}{T}\sum_{t=0}^{T-1} D(t) + \frac{L(Q(T)) - L(Q(0))}{T} \leq B + VP^* - \frac{\epsilon}{T}\sum_{t=0}^{T-1} Q(t).$$

Making $T \to \infty$, as a result, $\frac{L(Q(T))-L(Q(0))}{T} = 0$. Note that $\frac{V}{T}\sum_{t=0}^{T-1} D(t) > 0$, we have:

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} Q(t) \leq \frac{B + VP^*}{\epsilon}. \tag{27}$$

Also, noting that $-\frac{\epsilon}{T}\sum_{t=0}^{T-1} Q(t) < 0$, we have

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} D(t) \leq \frac{B}{V} + P^*. \tag{28}$$

Insight: The inequalities (27) and (28) demonstrate an $[O(\frac{1}{V}), O(V)]$ performance-cost tradeoff. The inequality (28) shows that by choosing a larger V, the average response time under the online algorithm can be pushed closer to the optimum value P^*. However, when V is too large, the virtual queue will be unstable, which means that the outsourcing cost has exceeded the budget and it is hard to enforce the constraint (15). Within the online algorithm above, tuning the parameter V such that $V > 0$ for all $t \in \{0, 1, ..., T-1\}$, we minimize both the outsourcing cost and the average response time.

C. Cost-Effective Online Algorithm

According to the problem (23) transformed by Lyapunov optimization techniques, the current optimal problem is much simpler. We find that there exists a variable $Q(t)$, which is the length of the virtual queue. In each round of optimization, based on the backlog of $Q(t)$ and the request arrival rate, we make the algorithm being "adaptive". Furthermore, during each time slot $t \in \{0, 1, ..., T-1\}$, we choose the optimal $\eta(t) = \frac{\lambda^R(t)}{\lambda(t)}$, which makes our algorithm "greedy". First, switching among the different levels of the tunnel does not have any constraint. Also, we can switch from the current level to any other level freely. Second, the average response time $D(t)$ is a continuous function in the available space of $\eta(t)$; it is easy to calculate the minimum value by using the bisection method, which only has $O(\log n)$ complexity.

Algorithm 1 Cost-Effective Online Algorithm (CEOA)

```
for each l ∈ [0, L] do
  calculate available space S of η(t)
  find η_a(t) ∈ S, which satisfies d(D(t))/d(η(t)) |_{η(t)=η_a(t)} = 0, and
  denote all the satisfied η_a(t) as η_Z(t)
  filter all the η_z(t) ∈ η_Z(t) and get η_m(t), which makes
  D(t) minimum
  get minimum objective O_min
  get η_min(t) that make objective minimum
  if O_opt > O_min then
    η_opt(t) = η_min(t)
    O_opt = O_min
  end if
end for
```

IV. Performance Evaluation

In this section, we conduct trace-driven simulations to evaluate the performance of our Cost-Effective Online Algorithm, i.e., **CEOA**. Our trace imitates the real flash crowd during the Double Eleven promotion season of Taobao, which is the largest e-commerce website with 6 billion subscribers in China.

A. Simulation Setup

We simulate a private cloud with constant capacity, and a public cloud with changeable capacity. In a promotion season, an e-commerce website knows flash crowds will come, and the initial capacity of the public cloud should be set large enough to handle the first wave of flash crowds. Then the public cloud will adjust the capacity to accommodate unknown upcoming workloads.

1) Hybrid Cloud Web Service Framework: In order to resolve the general and essential problem in a website within a hybrid cloud, we propose a simple framework for adjusting the capacity of the public cloud. Our online framework is built as a high-level abstraction that hides the details of web application tiers, the operating system used, the types of HTTP servers and cloud providers. Our online framework aims to give a basic understanding of flash crowds in e-commerce web services within a hybrid cloud. It contains three main modules: a private cloud, a public cloud and a load balancer. In order to imitate the real web service and request serving process, we also add

a request generator out of clouds. Moreover, we also add a tunnel with limited bandwidth which represents the connection between the private cloud and the public cloud, resembling the function of the AWS Direct Connection. During each time slot, the request generator produces a series of requests; then each cloud which contains a queue will receive requests dispatched by the load balancer.

2) Flash Crowd Arrival Pattern: In order to simulate the large scale of flash crowds, we apply four types of request arrival patterns to our simulations. First, we imitate the real trace of a flash crowd from Taobao. The trace of the flash crowd comes from the real trace of page view (PV) during Double Eleven promotion season of Taobao. Second, apart from the real trace, we also design three other request arrival patterns calculated by mathematic formulations. They are: 1) *constant arrival pattern*; 2) *random arrival pattern*; 3) *realistic arrival pattern* which has been characterized based on empirical measurement studies and fluid modeling analysis function. All the three mathematic request arrival patterns can provide simulation results as comparison and different scales of flash crowds by tuning key parameters.

3) Two Request Distributing Strategies for Comparison: In order to explore the performance of **CEOA**, we set two simple strategies to be compared with **CEOA**. One is the "public cloud first" strategy ("cloud first" for short), which tends to redirect requests to the public cloud as many as possible. When the public cloud is incapable to handle the excessive requests, the requests will be redirected to the private cloud. The other strategy is the "local first" strategy. The "local first" strategy allows the e-commerce website to use the private cloud as much as possible, which is a totally contrary strategy to "cloud first". By comparing **CEOA** with such two extreme strategies, we can analyze the performance of **CEOA** more clearly.

B. Performance Evaluation

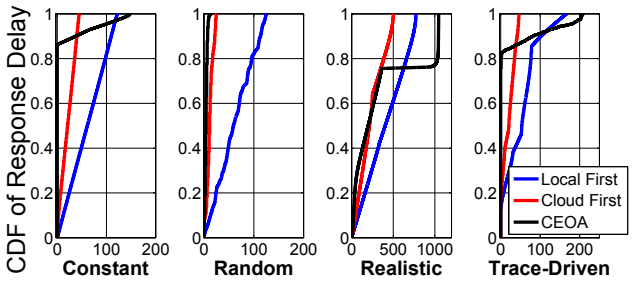

Fig. 3. CDF of response time of different strategies under different request arrival patterns

CDF of response time. We apply three different optimizing strategies to four flash crowd patterns. And the results are displayed in Fig. 3. The four flash crowd patterns are introduced in Sec. IV-A2. In Fig. 3, we can observe that the "cloud first" strategy can enable the website to provision best services under any type of flash crowds. Because the capacity of the public cloud is theoretically infinite, performance of "cloud first" strategy is the best. On the contrary, the "local first" strategy provisions the worst services. In Fig. 3, about 80 percent of response time results produced by **CEOA** are close to those of the "cloud first" strategy. Nearly 20 percent of response time results are close to those of "local first" strategy. This interesting phenomenon implies that **CEOA** is searching the balancing point of service quality between those two extreme solutions.

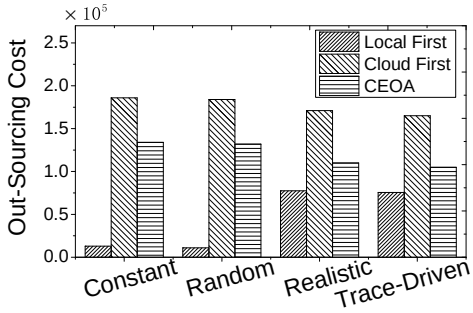

Fig. 4. Outsourcing cost of different strategies under different flash crowd arrival patterns

Outsourcing cost. As illustrated in Fig. 4, the outsourcing cost of the "local first" strategy is the minimum while the outsourcing cost of the "cloud first" strategy is the maximum among these three strategies. Based on Fig. 3 and 4, although it seems that **CEOA** is not outstanding in either performance or cost, we can discover that **CEOA** is trying to make the website provision better services by spending less outsourcing cost, i.e., cost-effective services. In order to justify whether **CEOA** can help the e-commerce website provision cost-effective services, we then measure the performance-cost ratio of all the strategies under different request arrival patterns.

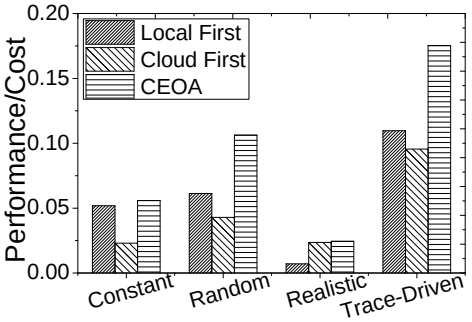

Fig. 5. The performance-cost ratio of different strategies under different request arrival patterns

Performance-Cost ratio. In this paper, we take average response time as the metric of performance, so reciprocal of the average response time can be regarded as the performance metric of the website. Hence, dividing the performance metric by total outsourcing cost, we can get the performance-cost ratio. Based on the analysis above, the "local first" strategy is the most economic while the "cloud first" strategy helps the website provision the best services. However, Fig. 5 clearly demonstrates that **CEOA** has the largest performance-cost ratio. In other words, we can conclude that **CEOA** is the best solution and enables the website to provision cost-effective services. According to the simulations above, we can intuitively conclude that there exists a tradeoff between performance and cost. As such, we try to further demonstrate the tradeoff between them.

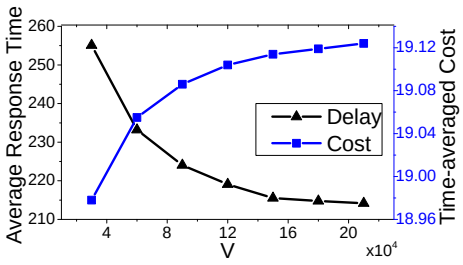

Fig. 6. The time-averaged outsourcing cost and the average response time vs. different values of V

The performance-cost tradeoff. To explore dedicated tradeoff between performance and cost, we vary the parameter V to choose the metric we want to emphasize, average response time or outsourcing cost. In Fig. 6, by tuning the value of V to a small one, we observe that the average response time is large while the outsourcing cost is small. Meanwhile, by setting a large value of V, it brings markedly increase of the outsourcing cost and decrease of the average response time. Furthermore, when the average response time drops, the outsourcing cost grows remarkably. By now, we have justified the tradeoff analyzed in Sec. III-B. And choosing a value of V represents how much we emphasize the average response time compared to the outsourcing cost and finally enables the e-commerce website to provision cost-effective services.

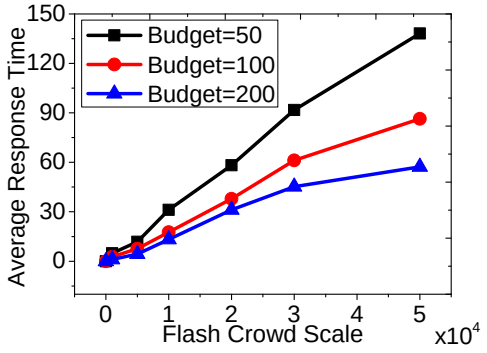

Fig. 7. The average response time under different scales of flash crowds with different values of budget

The average response time under different values of budget. Fig. 7 demonstrates the relationships among average response time, budget, and the scale of flash crowds. The time-averaged budget varies from 50 to 200, while the maximum scale of the flash crowd varies from 0 to 50000. Then we observe some interesting trends in Fig. 7: on the one hand, as the budget increases and the scale of the flash crowd stays constant, we observe that the average response time decreases much when the budget increases form 50 to 100. However, when the budget increases from 100 to 200, the average response time decreases not as much as it dose when budget increases from 50 to 100. The phenomenon demonstrates that more budget will improve performance, however, too much amount of budget will not promote the performance much. On the other hand, under a certain amount of budget, when the scale of the flash crowd increases, the average response time increases conspicuously. Yet we find that the increasing trend is approximately linear, which implies that the scale of the flash crowd may affects average response time linearly.

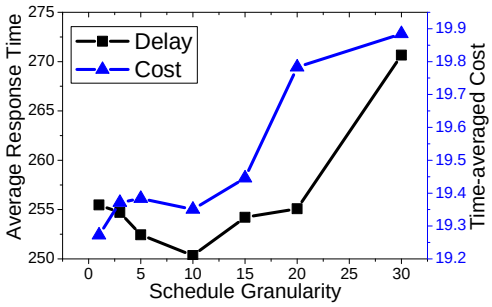

Fig. 8. The time-averaged outsourcing cost and the average response time vs. different values of granularity

Analysis on different values of granularity. In Fig. 8, we set various types of optimizing granularity, which is the least time interval of making one time request distribution. When the granularity of optimizing grows from 1 to 10, the average response time decreases correspondingly. However, when granularity of optimizing increases from 10 to 30, the average response time increases. Fig. 8 implies that when granularity of optimizing is too small, it will not help minimize the average response time. Because a flash crowd is featured by its bursty and fluctuation, the number of requests of every time slot may be quite different from each other. If granularity of optimizing is too small, the public cloud will be sensitive, and easy to adjust its capacity based on the fluctuating workloads. On the other hand, adjusting capacity frequently will lead to minimize the outsourcing cost, however, the performance can not be guaranteed. Based on Fig. 8, we can conclude that the granularity of optimizing can neither be too large nor too small.

V. Related Work

For a hybrid cloud, how to outsource workloads from a private cloud to a public cloud is a critical challenge. We recognize that existing works [15][16] that contribute to workload distribution and modeling under hybrid cloud scenarios. Our study is different from and complement to these works. More importantly, our work takes flash crowds into consideration, which makes the problem more challenging.

Compared to [15] which splits workloads into two parts in a hybrid cloud, our study differs in at least two important aspects. First, we model the e-commerce website in the hybrid cloud with the queueing theory while Zhang *et al.* [15] do not propose any mathematic model. Second, our work proposes an online framework to provision cost-effective services with rigorous optimality analysis. However, Zhang *et al.* [15] concentrate on reducing workload dynamics with neither further exploring impacts brought to the performance nor proving the effectiveness theoretically.

Compared to [16] that also solves workloads distribution in a hybrid cloud, our study is different in the following aspects. First, we discuss flash crowds in the e-commerce websites, where workloads are more fluctuating. Second, our workloads are user requests that are different from the MapReduce workloads in [16]. It is necessary for user requests to be served

as quickly as possible, while MapReduce tasks are delay-tolerant. Third, we add the tunnel that connects the public cloud and the private cloud in the system model, which is more practical.

Liu *et al.* [17] focus on the effects of flash crowds and leverage population control for alleviating flash crowds in P2P live streaming system. Inspired by [17] yet different from it, our work studies flash crowds in e-commerce website scenarios deployed in hybrid clouds.

With respect to studies on managing the performance overhead of VMs, part of the work in [18] summarizes it under diverse scenarios of the IaaS cloud. Specifically, Xu *et al.* [18] discuss the performance modeling methods with a particular focus on their cost, and compare the overhead mitigation techniques by identifying their effectiveness. Correspondingly, by leveraging a mitigation technique of request distributing, our work aims at guaranteeing performance along with controlling cost, i.e., provisioning cost-effective services, in a hybrid cloud, which is different from and complementary to the IaaS cloud in [18].

On the other hand, our work studies provisioning cost-effective services in e-commerce websites, which is from the perspective of applications rather than VMs in [18]. By using a multi-tier architecture, Urgaonkar *et al.* [5] abstract all the web applications as well as evaluate the queueing delay. As a result, they approximately measure the performance of web applications. In this paper, we use an M/M/1 queue to encapsulate the whole functionalities in a website so that we can estimate the performance of e-commerce websites and concentrate on solving request distribution problem.

A majority of the existing literature leverages Lyapunov optimization techniques. For example, with respect to geo-distributed datacenters, Zhou *et al.* [19] take advantage of Lyapunov optimization techniques to design and analyze a carbon-aware control framework, which makes online decisions on geographical load balancing, capacity right-sizing, and server speed scaling; with respect to single mega-datacenter, they also apply Lyapunov optimization techniques to design and analyze an optimal control framework to make online decisions on request admission control, routing, and VM scheduling [20]. Shu *et al.* [21] present eTime, a novel energy-efficient data transmission strategy between cloud and mobile devices, based on Lyapunov optimization.

Different from the works mentioned above, we particularly adapt Lyapunov optimization approaches to request distributing and decision-making of scaling in the context of hybrid clouds. After transforming original optimization objective to a new problem, we use a greedy strategy to solve it. The function of queueing delay is continuous, so we use the bisection method to obtain its approximate minimum value. It only takes $O(\log n)$ of the algorithm complexity. Furthermore, from the real product of AWS Direct Connect, users can switch from the current level of bandwidth to any other one freely to get the optimal level. Hence, applying greedy algorithm to this problem is efficient and simple.

VI. Conclusion

In response to increasing page visits and bill transactions brought by flash crowds during promotion seasons, most e-commerce websites rent public cloud computing resources to ease workloads laid to private clouds. How to outsource the workloads to a public cloud so that the website can provision cost-effective services becomes a critical challenge. In this paper, we designed an online algorithm to help an e-commerce website provision cost-effective services. By applying Lyapunov optimization techniques, our online algorithm can make real time decision on how to outsource workloads from a private cloud. Furthermore, we proved that our online algorithm can approach a dedicated $[O(\frac{1}{V}), O(V)]$ tradeoff between outsource cost and average response time. Through simulations with empirical real e-commerce PV trace, we demonstrated the effectiveness of our solution in both minimizing the average response time and controlling the outsourcing cost.

References

[1] How Alibaba Catered To USD 3 Billion Sales In A Day. [Online]. Available: http://www.infoq.com/news/2012/12/interview-taobao-tmall

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

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

[4] A. Perkins and T. Owen, "How businesses should be incorporating hybrid cloud as part of their core it strategy," White Paper, Rackspace, Sep. 2013.

[5] 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.

[6] 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.

[7] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *Networking, IEEE/ACM Transactions on*, 1995.

[8] L. Kleinrock, *Queueing Systems, Volume 1: Theory.* John Wiley and Sons, Inc, 1975.

[9] RackSpace. [Online]. Available: http://www.rackspace.com/

[10] Windows Azure. [Online]. Available: https://azure.microsoft.com/

[11] VMWare vCloud Hybrid Service. [Online]. Available: http://vcloud.vmware.com/

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

[13] L. P. Slothouber, "A model of web server performance," in *Proc. of ACM WWW*, 1996.

[14] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan and Claypool Publishers, 2010.

[15] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena, "Intelligent workload factoring for a hybrid cloud computing model," in *Proc. of World Conference on Services - I*, 2009.

[16] X. Qiu, W. L. Yeow, C. Wu, and F. Lau, "Cost-minimizing preemptive scheduling of mapreduce workloads on hybrid clouds," in *Proc. of IEEE/ACM IWQoS*, 2013.

[17] 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 Trans. Parallel and Distributed Systems*, 2012.

[18] F. Xu, F. Liu, H. Jin, and A. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proc. of the IEEE*, vol. 102, no. 1, pp. 11–31, Jan 2014.

[19] Z. Zhou, F. Liu, Y. Xu, R. Zou, H. Xu, J. Lui, and H. Jin, "Carbon-aware load balancing for geo-distributed cloud services," in *Proc. of IEEE MASCOTS*, 2013.

[20] Z. Zhou, F. Liu, H. Jin, B. Li, B. Li, and H. Jiang, "On arbitrating the power-performance tradeoff in saas clouds," in *Proc. of IEEE INFOCOM*, 2013.

[21] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, and Y. Qu, "eTime: Energy-efficient transmission between cloud and mobile devices," in *Proc. of IEEE INFOCOM*, 2013.