On Arbitrating the Power-Performance Tradeoff in SaaS Clouds

Zhi Zhou[1] Fangming Liu*[1] Hai Jin[1] Bo Li[2] Baochun Li[3] Hongbo Jiang[4]
[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.
[4]Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China.
[2]The Hong Kong University of Science and Technology, Hong Kong. [3]University of Toronto, Canada.

***Abstract*—In this paper, we present an analytical framework for characterizing and optimizing the power-performance tradeoff in Software-as-a-Service (SaaS) cloud platforms. Our objectives are two-fold: (1) We maximize the operating profit when serving heterogeneous SaaS applications with unpredictable user requests, and (2) we minimize the power consumption when processing user requests. To achieve these objectives, we take advantage of Lyapunov Optimization techniques to design and analyze an optimal *control framework* to make online decisions on request admission control, routing, and virtual machine (VMs) scheduling. In particular, our control framework can be flexibly extended to incorporate various design choices and practical requirements of a datacenter in the cloud, such as enforcing a certain power budget for improving the performance (dollar) per watt. Our mathematical analyses and simulations have demonstrated both the optimality (in terms of a cost-effective power-performance tradeoff) and system stability (in terms of robustness and adaptivity to time-varying and bursty user requests) achieved by our proposed control framework.**

I. Introduction and Related Work

Software-as-a-Service (SaaS) cloud platforms, such as Google Apps [1] and Salesforce.com [2], have quickly ascended to the spotlight in the realm of cloud computing platforms, surpassing Infrastructure-as-a-Service (IaaS). With SaaS cloud services, enterprise applications — as critical as customer relationship management (CRM) and as simple as online slide presentations — can be hosted in the cloud, with large-scale datacenters serving a wide range of applications.

With SaaS, users are typically charged for each of its transaction or request. For example, a popular cloud-based email marketing application, called Campaign Monitor [3], charges users according to their number of processed campaigns and delivered recipients. By following the law of *diminishing marginal utility* in economics*, heavy users with more subscribers (*e.g.,* recipients of email newsletters) will receive a more significant discount on the per-request charge as the number of recipients increases. From the perspective of the SaaS cloud, however, as requests from users arrive in an unpredictable and even bursty fashion, such a per-request charging model may lead to fluctuating revenues over time.

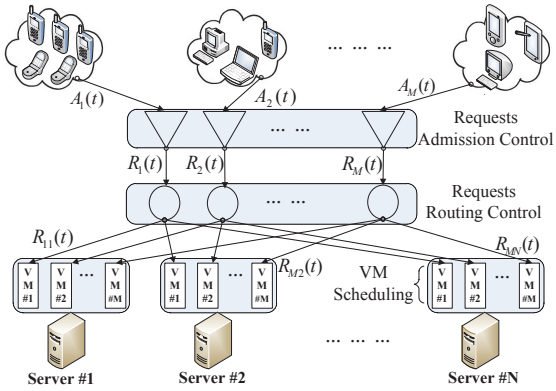

Fig. 1: A basic virtualized datacenter with three control decisions: (1) admission control that accepts (or denies) arrived requests from different applications, (2) routing control that dispatches admitted requests across VMs hosted on different servers in a datacenter, and (3) scheduling of VMs by switching between running and idle states.

Let us consider a typical SaaS cloud platform in Fig. 1. Due to risks of going beyond the processing capacity, we need a front-end proxy to admit requests. We also require a load dispatcher to redirect admitted requests across a large pool of servers with constrained power budgets [4], each of which is virtualized to multiple virtual machines (VMs) to process requests from different applications.

With the presence of *unpredictable* and *bursty* application demands, the objectives of such an SaaS cloud platform with a per-request charging model are two-fold: (1) to maximize its profit by accepting and processing as many application requests as possible (system throughput), and (2) to minimize the penalty from system congestion due to excessive requests and the resulted power consumption of servers. To balance such a *power-performance tradeoff*, three important control decisions need to be made in the SaaS cloud: (1) how many requests from diverse applications are to be admitted at any

*The Corresponding Author is Fangming Liu (fmliu@hust.edu.cn). The research was support in part by a grant from The National Natural Science Foundation of China (NSFC) under grant No.61103176 and No.61133006, by a grant from the Research Fund of Young Scholars for the Doctoral Program of Higher Education, Ministry of Education, China, under grant No.20110142120079, by the CHUTIAN Scholar Project of Hubei Province. Dr. Bo Li's work was supported by a grant from NSFC/RGC under the contract N_HKUST610/11, by grants from HKUST under the contract RPC11EG29, SRFI11EG17-C and SBI09/10.EG01-C, by a grant from Guangdong Bureau of Science and Technology under the contract GDST11EG06, by a grant from ChinaCache Int. Corp. under the contract CCNT12EG01. Dr. Hongbo Jiang's work was supported by grants from NSFC under No.61271226 and No.61073147.

*Note that even for long-term SaaS commitments with an upfront and flat payment, the law of diminishing marginal utility is also applicable in general, in the form of a discount.

given time; (2) how to distribute the admitted requests from different applications across a large number of servers hosting the corresponding VMs; and (3) how to schedule each VM by switching between a *running* state for processing requests and an *idle* state for conserving server power.

To address these challenges, this paper takes advantage of *Lyapunov optimization* [5] techniques to rigorously design and analyze a new *optimal online control framework*, designed to independently and concurrently make all three decisions in the SaaS cloud. Our framework may be used to design three simple yet effective strategies, corresponding to the three decisions that need to be made. (1) A threshold-based admission control strategy to improve system throughput while avoiding system congestion; (2) a "Join the Shortest Queue" request routing strategy for balancing the server load, reducing service delays of admitted requests; and (3) a decentralized greedy strategy to optimally schedule VMs, *i.e.,* which VMs are to process incoming requests, and which are to be kept idle for power conservation. The upshot of our new framework is that it can be extended to explore various design choices and practical requirements of a datacenter. As a case study, we demonstrate how to extend it to improve the performance (dollar) per watt [4] at a datacenter, by enforcing a certain power budget. With extensive simulations, we demonstrate that our new control framework can approach a time-averaged profit that is arbitrarily close to the optimum, while still maintaining strong stability and low congestion. Further, it is able to quickly adapt to bursty and time-varying arrivals of application requests, without incurring overwhelming power consumption costs that may outweigh the benefit of aggressively admitting a large number of requests.

While recognizing the significance of many existing works (*e.g.,* [6]–[8]) on managing the two potentially conflicting objectives related to cloud application performance and datacenter power consumption, our study is different from and complementary to existing works. *First*, a number of existing works heavily relied on prediction-based or statistical offline approaches. For example, Chen *et al.* [9] applied a multiplicative seasonal autoregressive moving average method to predict server workloads in each time interval, and then made power control decisions based on steady-state queueing analysis and feedback control theory to satisfy such predicted demands. Govindan *et al.* [10] characterized statistical properties of the power needs of hosted workloads through a measurement-driven profiling and prediction framework. In the context of application requests in an SaaS cloud, the common problem with such approaches lies in the dubious feasibility of making accurate predictions of future request patterns, due to the fact that they are, in general, bursty and nonstationary.

Second, though there exist alternative online control solutions [11]–[13] in the literature for dynamic resource allocation and power management in datacenters, our work differs substantially in at least two important aspects. (1) We take an economic viewpoint to price application throughput in a nonlinear utility function based on the law of diminishing marginal utility (Sec. II-B3), rather than a simple linear utility function [11] that does not reflect reality in general. (2) Our framework can be extended to focus on improving the performance (dollar) per watt [4] at datacenters, by achieving a desired system throughput level with a certain power budget.

II. Basic Power-Performance Tradeoff Model

In this section, we first formulate the basic datacenter model in Fig. 1, consisting of N homogeneous servers $\mathcal{S} = \{1, 2, ..., N\}$, each of which is virtualized to M virtual machines (VMs) to serve M types of heterogeneous applications $\mathcal{A} = \{1, 2, ..., M\}$ with diverse request arrival rates and computing demand. Specifically, the i-th VM hosted on server j serves the requests from the i-th type of application instance. Inspired by the latest modeling work on datacenters [6], we consider a discrete time-slotted system where the time slot length can range from hundreds of milliseconds to minutes [12]. In every time slot t $(= 0, 1, 2, ..., \tau, ...)$, a number of requests $A_i(t)$ generated by the i-th type of application arrive at the datacenter, and the time averaged rate of such an arrival process can be denoted as $\lambda_i = \mathbb{E}\{A_i(t)\}$.

We assume that each random variable $A_i(t), \forall i \in \mathcal{A}$, is independent and identically distributed (i.i.d.) over time slots, and they are independent of the current amount of unfinished workload in the system. We also assume that there exist certain peak levels of application requests $A_i^{\max}, \forall i \in \mathcal{A}$, such that $\{A_i(t) \leq A_i^{\max}, \forall i \in \mathcal{A}, \forall t\}$. However, since the workloads in a cloud computing environment is highly dynamic and usually unpredictable (*e.g.,* demands can spike abruptly [14] and potentially exceed the current available processing capacity of a datacenter), *our model does not assume any a priori knowledge of the statistics* of $A_i(t), \forall i \in \mathcal{A}, \forall t$.

A. Control Decisions

Under the datacenter workload model above, we focus on *three* important control decisions to be made, introduced in Fig. 1 and Sec. I, along with key notations in Table I.

1) Admission control of application requests: In each time slot t, the first control decision of a datacenter is to determine a subset of requests of each application $R_i(t), \forall i \in \mathcal{A}$ (out of the potentially substantial amount of newly arrived requests of each application $A_i(t), \forall i \in \mathcal{A}$), that can be admitted into the system[†]: $\{0 \leq R_i(t) \leq A_i(t), \forall i \in \mathcal{A}, \forall t\}$.

2) Routing control of application requests: As soon as a portion of requests of each application $R_i(t), \forall i \in \mathcal{A}$ are admitted into the datacenter, the next control decision is to route (dispatch) $R_i(t), \forall i \in \mathcal{A}$ to the corresponding *queue* for each application on each server, where the requests will wait to be processed. Let $R_{ij}(t)$ denote a subset of requests to be routed to the queue maintained by the i-th VM on the j-th server in time slot t. Then, the routing control decisions should satisfy an obvious constraint: $R_i(t) = \sum_{j=1}^{N} R_{ij}(t), \forall i \in \mathcal{A}, \forall j \in \mathcal{S}, \forall t$. Such a routing control can be implemented as the *load balancer(s)* [14] of realistic datacenters.

[†]Though such an admission control of application requests can be achieved in either a centralized frond-end component or in a distributed manner across backend servers, our model focuses on the general underlying control decision without being restricted to any specific design choice and implementation.

3) Scheduling of VMs: While the amount of routed requests $R_{ij}(t)$ are waiting in the corresponding queue maintained by the i-th ($\forall i \in \mathcal{A}$) VM on the j-th ($\forall j \in \mathcal{S}$) server, another important control decision is to schedule each VM in time slot t, by switching between the *running* state (to process the routed application requests that are waiting in this VM's queue) and the *idle* state[‡] to keep the routed requests waiting in this VM's queue, without processing them in the current time slot. Such VM scheduling decisions are denoted by the following indicator variables for $\forall i \in \mathcal{A}$, $\forall j \in \mathcal{S}, \forall t$:

$$a_{ij}(t)= \begin{cases} 1, & \text{if the } i\text{-th VM on server } j \text{ is running,} \\ 0, & \text{if the } i\text{-th VM on server } j \text{ is idle.} \end{cases}$$

For each VM, its queue backlog, arrival rate and service rate of application requests can be derived as follows. *First*, we assume that the limited processing capacity (*e.g.,* CPU, memory, disk or network bandwidth) of any server $\forall j \in \mathcal{S}$ is *fairly* allocated among its hosted M VMs, *i.e.*, the processing capacity of each VM is $1/M$ of the total processing capacity of its hosting server. *Second*, while requests from different applications require different amounts of processing capacity, we assume that each request from the same application requires the same amount of processing capacity. Then, the number of time slots d_i for a VM to process each request of a specific application $\forall i \in \mathcal{A}$ is identical. Henceforth, d_i is referred to as the *size* of each request of this particular application $\forall i \in \mathcal{A}$. *Finally*, we define the *queue backlog* $Q_{ij}(t)$ of the i-th VM on the j-th server as the total sizes of all the requests that are waiting in the queue at the beginning of time slot t (Initially, $Q_{ij}(0) = 0, \forall i \in \mathcal{A}, j \in \mathcal{S}$). The corresponding *service rate* and *arrival rate* of a queue in time slot t can be quantified as $a_{ij}(t)$ and $d_i \cdot R_{ij}(t)$, respectively. By doing so, we can capture the following *queueing dynamics* over time for each VM hosted on each server in a datacenter:

$$Q_{ij}(t+1) = \max[Q_{ij}(t) - a_{ij}(t), 0] + d_i R_{ij}(t). \quad (1)$$

With the VM scheduling above, it is intuitive that the more VMs remain in the running state, the better processing capacity and performance. Yet, the tradeoff is a larger amount of power consumed by the datacenter, as we shall characterize in the following subsection.

B. Characterizing and Optimizing the Power-Performance Tradeoff

1) System Throughput: For large-scale SaaS cloud platforms, one of the most important performance metrics is the overall application throughput in terms of the total number of requests (of all provided applications) that can be admitted and processed. Specifically, for each application $\forall i \in \mathcal{A}$, we define the *time averaged throughput* r_i of a datacenter as:

$$r_i = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{R_i(\tau)\}, \quad (2)$$

[‡]We focus on the decision for whether a VM will process application request(s) or remain idle in current time slot, rather than frequently turning on/off VMs (or servers) per time slot, which would incur considerable performance and energy overhead [15].

TABLE I: Key Parameters in the Basic Datacenter Model.

Notations	Definitions
M	The number of heterogeneous applications $\mathcal{A}$ served by a datacenter
N	The number of homogeneous servers $\mathcal{S}$ in a datacenter
$a_{ij}(t)$	The running/idle state of the i-th VM on the j-th server in time slot t
$A_i(t)$	The number of arrived requests of each application $\forall i \in \mathcal{A}$ in time slot t
$R_i(t)$	The number of admitted requests of each application $\forall i \in \mathcal{A}$ in time slot t
$R_{ij}(t)$	The portion of requests of each application $\forall i \in \mathcal{A}$ that are routed to the i-th VM on the j-th server in time slot t
$Q_{ij}(t)$	The queue backlog of each application $\forall i \in \mathcal{A}$ on the j-th server in time slot t
$P_j(t)$	The power consumption of the j-th server in time slot t
d_i	The respective size of a request of each application $\forall i \in \mathcal{A}$
λ_i, r_i, p_j	The time average of $A_i(t), R_i(t)$ and $P_j(t)$
V	Lyapunov control parameter

then, the metric $\sum_{i=1}^{M} r_i$ is the overall datacenter throughput that is expected to be maximized, subject to the following two constraints: (1) $r_i \leq \lambda_i$, as the time averaged throughput r_i cannot exceed the time averaged arrival rate λ_i for any application $\forall i \in \mathcal{A}$, and (2) $r_i \leq \frac{N}{d_i}$, as the time averaged throughput r_i cannot exceed the overall processing capacity allocated for the corresponding application $i \in \mathcal{A}$.

2) Power Consumption: Here we focus on a basic power consumption model of servers where the CPU processing capacity is the main bottleneck, such as for serving computation-intensive applications. Yet, our model can be extended to incorporate other components (*e.g.,* memory, disk and network I/O) [11], such as representing multi-dimensional resources of a server as a vector in our formulation, by amending the aforementioned request sizes and service rates accordingly.

Specifically, it has been widely shown by recent studies [12], [16] that, the amount of power consumed by a server (CPU processor) is primarily associated with its current CPU running speed s, as formally characterized by the following Eq. (3). Without loss of generality, we consider a normalized $s \in [0, 1]$ (alternatively viewed as the CPU utilization ratio) and its corresponding normalized power consumption $P(s) \in [0, 1]$, where $s = 0$ represents the idle state of a server while $s = 1$ represents its maximum CPU speed in the running state:

$$P(s) = \alpha s^v + (1 - \alpha), \quad (3)$$

where the exponent parameter v is empirically determined as $v \geq 1$ in practice (*e.g.,* a typical value is $v = 2$ [16]). With another parameter $\alpha \in [0, 1]$, the term $(1 - \alpha)$ represents the normalized power consumption of an idle server. Practical measurements [17] have shown that $(1-\alpha)$ is around 0.6 (and barely lower than 0.5), which implies that an idle server still consumes a non-trivial amount of power.

Based on the power model above, for a server $j \in \mathcal{S}$ that hosts a number of M VMs with the fair capacity allocation policy described in Sec. II-A, its normalized CPU load and corresponding power consumption in time slot t are given as: $s_j(t) = \frac{\sum_{i=1}^{M} a_{ij}(t)}{M}$ and $P_j(t) = \alpha(\frac{\sum_{i=1}^{M} a_{ij}(t)}{M})^v + (1 - \alpha)$, respectively. Accordingly, the *time average of normalized power consumption* p_j of each server $\forall j \in \mathcal{S}$ in a datacenter

can be defined as:

$$p_j = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{P_j(\tau)\}, \tag{4}$$

then, the metric $\sum_{j=1}^{N} p_j$ is the overall power consumption of all servers[§], hopefully being minimized.

3) A Unified Objective from an Economic Perspective: So far, we have derived both the datacenter performance metric $r_i, \forall i \in \mathcal{A}$ in Eq. (2) (time averaged throughput) and power consumption metric $p_j, \forall j \in \mathcal{S}$ in Eq. (4) (time averaged power consumption). However, the fundamental challenge is *how to optimize the tradeoff between the two potentially conflicting objectives in a balanced and cost-effective manner?* To this end, we first construct a unified profit objective to couple both sides in an economic way as follows.

First, quantifying the power cost. The power cost of a datacenter can be measured as $(\text{Price} \cdot \text{PUE} \cdot \sum_{j=1}^{N} p_j)$, where Price is the electricity market price of each unit of the normalized power consumption, and PUE is the power usage efficiency metric provided by Green Grid [19]. It represents the ratio of the total amount of power used by the entire datacenter facility to the power delivered to the computing equipment. Reportedly, inefficient datacenter facilities can have a PUE $\in [2.0, 3.0]$, while leading industry datacenter facilities are announced to approach a PUE of around 1.2 [14].

Second, pricing the system throughput. Different from the power which is demanded rigidly, there exists "elastic demand" for the cloud applications served by a datacenter. Hence, we choose to price the throughput of each application $i \in \mathcal{A}$ served by the datacenter as a log function $g(r_i) = \log(1 + d_i r_i)$, according to the law of diminishing marginal utility in economics. Such a rule has also been adopted by real-world SaaS cloud services (*e.g.,* [3]). In addition, such a nonlinear function is different from a closely related work [11], which used a simple linear utility function.

Given the *revenue* brought by the system throughput and the *cost* incurred by the power consumption, we can maximize the time averaged profit of a datacenter as follows:

$$\begin{aligned} \max \quad & \sum_{i\in\mathcal{A}} g(r_i) - \beta \sum_{j\in\mathcal{S}} p_j \\ \text{s.t.} \quad & 0 \le r_i \le \lambda_i, r_i \le N/d_i, \quad \forall i \in \mathcal{A}, \end{aligned} \tag{5}$$

where the factor $\beta = \text{Price} \cdot \text{PUE}$.

Let r_i^* and $p_j^*, \forall i \in \mathcal{A}, \forall j \in \mathcal{S}$ denote the optimal solution to the **Problem** (5). For realistic datacenters, there are two challenges when solving this problem: (1) The datacenter workload is time-varying and unpredictable, which makes it infeasible to precisely capture key parameters (such as λ_i), and impractical to calculate optimal solution in an offline manner. (2) The large number of servers and their hosted applications exacerbate the computational complexity of centralized solution. In response, we seek to design an *online* and *distributed* control algorithm in Sec. III, which is able to efficiently make decisions on all three important control decisions.

[§]Given recent reports [18] that the power consumption of other non-IT equipments (*e.g.,* cooling) is roughly proportional to that by servers, our basic model can also be extended to capture the overall power consumption of a datacenter by scaling up $\sum_{j=1}^{N} p_j$ with a constant factor [12].

III. Constructing an Online Control Framework

In response to the challenges of **Problem** (5), we take advantage of Lyapunov optimization techniques [5] to design an online control framework, which is able to concurrently make all three important control decisions in Fig. 1, including *request admission control*, *routing*, and *VM scheduling*. In particular, our control algorithms can be proved to approach a time averaged profit that is arbitrarily close to optimum, while still maintaining system stability.

A. Problem Transformation Using Lyapunov Optimization

As the function $g(r_i)$ is nonlinear, we are inspired by recent techniques [20] to transform **Problem** (5) to the framework of Lyapunov optimization **Problem** (6), which introduces *auxiliary variables* γ_i for each admitted stream of application requests $R_i(t), \forall i \in \mathcal{A}$, in the system described in Fig. 1:

$$\max \quad \sum_{i\in\mathcal{A}} g(\gamma_i) - \beta \sum_{j\in\mathcal{S}} p_j \tag{6}$$

$$\text{s.t.} \quad \gamma_i \le r_i, \quad \forall i \in \mathcal{A} \tag{7}$$

$$0 \le r_i \le \lambda_i, \quad \forall i \in \mathcal{A} \tag{8}$$

$$r_i \le N/d_i, \quad \forall i \in \mathcal{A}. \tag{9}$$

It is easy to check that the optimal solution of the problem above is the same as that of the original **Problem** (5), as the function $g(*) = \log(1 + *d_i)$ is non-decreasing.

To solve the problem above, we first transform the inequality constraint (7) into a queue stability problem [5]. Specifically, we introduce *virtual queues* $H_i(t)$ for each $R_i(t)$. Initially, we define $H_i(0) = 0, \forall i \in \mathcal{A}$, and then update the queues per each time slot as follows:

$$H_i(t+1) = \max[H_i(t) - R_i(t), 0] + \gamma_i(t), \tag{10}$$

where $\gamma_i(t)$ denotes a process of non-negative auxiliary variables that the admission control of the datacenter system (Sec. II-A) will determine in every time slot, which satisfy the constraint (11). The time average of each $\gamma_i(t)$ is defined as $\gamma_i = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{\gamma_i(\tau)\}$.

$$0 \le \gamma_i(t) \le A_i^{\max}. \tag{11}$$

Insight: Intuitively, the auxiliary variables $\gamma_i(t)$ can be viewed as the "arrivals" of virtual queues $H_i(t)$, while $R_i(t)$ can be viewed as the service rate of such virtual queues. The constraints (7) are enforced on the condition that the virtual queues $H_i(t)$ are stable, *i.e.,* $\lim_{t\to\infty} \mathbb{E}\{H_i(t)\}/t = 0$. Specifically, from (10) it is clear that: $H_i(t+1) \ge [H_i(t) - R_i(t) + \gamma_i(t)]$. By summing this inequality over time slots $\tau \in \{0, 1, ..., t-1\}$ and then dividing the result by t, we have: $\frac{H_i(t) - H_i(0)}{t} + \frac{1}{t}\sum_{\tau=0}^{t-1} R_i(\tau) \ge \frac{1}{t}\sum_{\tau=0}^{t-1} \gamma_i(\tau)$. With $H_i(0) = 0$, taking expectations of both sides yields:

$\lim_{t\to\infty}\frac{\mathbb{E}\{H_i(t)\}}{t}+r_i\geq\gamma_i$. If the virtual queues $H_i(t)$ are stable, then $\lim_{t\to\infty}\mathbb{E}\{H_i(t)\}/t=0$ (Note that we will prove the strong stability of the virtual queues $H_i(t)$ in Theorem 1 later), so that the constraint (7) can be satisfied.

1) Characterizing the Stability-Profit Tradeoff: Let $\mathbf{Q}(t)=(Q_{ij}(t))$ and $\mathbf{H}(t)=(H_i(t))$ denote the matrix of the actual and virtual queues maintained by VMs (Sec. II-A). Then, we use $\mathbf{\Theta}(t)=[\mathbf{Q}(t);\mathbf{H}(t)]$ to denote the combined matrix of all the actual queues and virtual queues. However, since $\mathbf{Q}(t)$ and $\mathbf{H}(t)$ have different scales (the former corresponds to the request size d_i according to Eq. (1), while the latter corresponds to the number of requests according to Eq. (10)), we define a Lyapunov function $L(\mathbf{\Theta}(t))$ which assigns different weights d_i and 1 to $H_i(t)$ and $Q_{ij}(t)$, respectively:

$$L(\mathbf{\Theta}(t))=\frac{1}{2}\Big[\sum_{i\in\mathcal{A}}d_i^2H_i^2(t)+\sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{S}}Q_{ij}^2(t)\Big]. \tag{12}$$

This represents a scalar metric of *queue congestion* [5] in the datacenter system. For example, a small value of $L(\mathbf{\Theta}(t))$ implies that both actual queue backlogs and virtual queue backlogs are small. The implication is that the corresponding datacenter system has *strong stability*.

To keep the system stable by persistently pushing the Lyapunov function towards a lower congestion state, we introduce $\Delta(\mathbf{\Theta}(t))$ as the *one-step conditional Lyapunov drift* [5]: $\Delta(\mathbf{\Theta}(t))=\mathbb{E}\{L(\mathbf{\Theta}(t+1))-L(\mathbf{\Theta}(t))|\mathbf{\Theta}(t)\}$. In the sense of Lyapunov optimization, the underlying objective of our optimal control decisions on request admission control, routing and VM scheduling is to minimize an infimum bound on the following *drift-minus-profit* expression in each time slot:

$$\Delta(\mathbf{\Theta}(t))-V\mathbb{E}\Big\{\sum_{i\in\mathcal{A}}g(\gamma_i(t))-\beta\sum_{j\in\mathcal{S}}P_j(t)|\mathbf{\Theta}(t)\Big\}. \tag{13}$$

Insight: The control parameter $V(\geq 0)$ represents *a design knob of the stability-profit tradeoff*, *i.e.,* how much we shall emphasize the profit maximization (**Problem** (6)) compared to system stability. It empowers system operators to make flexible design choices among various tradeoff points between system stability and profit. For example, one may prefer to achieve as much expected profit $\mathbb{E}\{\sum_{i=1}^{M}g(\gamma_i(t))-\beta\sum_{j=1}^{N}P_j(t)|\mathbf{\Theta}(t)\}$ as possible, while having to keep $\Delta(\mathbf{\Theta}(t))$ small to avoid higher system congestion.

2) Bounding Drift-Minus-Profit: The analysis above instructs a system designer to derive an infimum bound of the *drift-minus-profit* expression given in Eq. (13), which requires the following Lemma 1.

Lemma 1: In each time slot t, for any value of $\mathbf{\Theta}(t)$, the Lyapunov drift $\Delta(\mathbf{\Theta}(t))$ of a datacenter system under any control strategy satisfies the following, where $B_1\triangleq\frac{MN+3\sum_{i=1}^{M}(d_iA_i^{max})^2}{2}$ is a finite constant parameter.

$$\Delta(\mathbf{\Theta}(t))\leq B_1-\sum_{i\in\mathcal{A}}d_i^2H_i(t)\mathbb{E}\{R_i(t)-\gamma_i(t)|\mathbf{\Theta}(t)\} \tag{14}$$
$$-\sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{S}}Q_{ij}(t)\mathbb{E}\{a_{ij}(t)-d_iR_{ij}(t)|\mathbf{\Theta}(t)\}.$$

Interested readers are referred to our detailed technical report [21] for a complete proof of Lemma 1.

Based on Lemma 1, subtracting the expression $V\mathbb{E}\{\sum_{i=1}^{M}g(\gamma_i(t))-\beta\sum_{j=1}^{N}P_j(t)|\mathbf{\Theta}(t)\}$ from both sides of Eq. (14) yields an infimum bound of *drift-minus-profit* expression of the datacenter system:

$$\Delta(\mathbf{\Theta}(t))-V\mathbb{E}\Big\{\sum_{i\in\mathcal{A}}g(\gamma_i(t))-\beta\sum_{j\in\mathcal{S}}P_j(t)|\mathbf{\Theta}(t)\Big\}\leq B_1$$
$$-\sum_{i\in\mathcal{A}}\mathbb{E}\{Vg(\gamma_i(t))-d_i^2H_i(t)\gamma_i(t)|\mathbf{\Theta}(t)\} \tag{15}$$
$$-\sum_{i\in\mathcal{A}}\mathbb{E}\Big\{d_i^2H_i(t)R_i(t)-\sum_{j\in\mathcal{S}}d_iR_{ij}(t)Q_{ij}(t)|\mathbf{\Theta}(t)\Big\} \tag{16}$$
$$-\sum_{j\in\mathcal{S}}\mathbb{E}\Big\{\sum_{i\in\mathcal{A}}Q_{ij}(t)a_{ij}(t)-V\beta P_j(t)|\mathbf{\Theta}(t)\Big\}. \tag{17}$$

B. An Optimal Online Control Algorithm

Instead of directly minimizing the *drift-minus-profit* expression in Eq. (13) that involves implicit $\max[*]$ terms in both Eq. (1) and Eq. (10), we seek to design an optimal **Online Control Algorithm (OCA)** to minimize its infimum bound given above (*i.e.,* equivalent to maximizing the terms (15)(16)(17) on the right-hand-side), without undermining the optimality and performance of the algorithm according to [5]. Interestingly, we will show that the maximization of the terms (15)(16)(17) can be decoupled to a series of independent subproblems, which can be computed concurrently in a decentralized fashion.

Specifically, in each time slot t, based on *online* observation of the queue backlogs $\mathbf{Q}(t)$ and $\mathbf{H}(t)$, **OCA** performs the following four phases of control operations, including: (1) auxiliary variable selection, (2) request admission control and routing control, (3) VM scheduling, and (4) queue update.

1) Auxiliary Variable Selection: For each application $i\in\mathcal{A}$ served by the datacenter, we determine $\gamma_i(t)$ by maximizing the term (15) in Sec. III-A2. Fortunately, as the decision variables $\gamma_i(t)$ are independent among applications, such centralized maximization can be decoupled to be computed concurrently as follows:

$$\max_{\gamma_i(t)}\quad V\log(1+d_i\gamma_i(t))-d_i^2H_i(t)\gamma_i(t) \tag{18}$$
$$\text{s.t.}\quad 0\leq\gamma_i(t)\leq A_i^{\max},\forall i\in\mathcal{A}.$$

Differentiating the objective function above with respect to $\gamma_i(t)$ can yield the peak value of the objective function when $\gamma_i(t)=\frac{V}{d_i^2H_i(t)}-\frac{1}{d_i}$. By taking the constraint (11) into consideration, we obtain the optimal solution to problem (18):

$$\gamma_i(t)=\begin{cases}0, & H_i(t)>\dfrac{V}{d_i}\\ \dfrac{V}{d_i^2H_i(t)}-\dfrac{1}{d_i}, & \dfrac{V}{d_i^2A_i^{\max}+d_i}\leq H_i(t)\leq\dfrac{V}{d_i}\\ A_i^{\max}, & H_i(t)<\dfrac{V}{d_i^2A_i^{\max}+d_i}\end{cases} \tag{19}$$

Insight: The stepped solution above is directly related to the value of $H_i(t)$. Recall from Sec. III-A that, if the value of

$H_i(t)$ is small, then it implies that the time average of $\gamma_i(t)$ is close to that of $R_i(t)$, which improves the system stability as we expected. In this case, a larger value of $\gamma_i(t)$ can be chosen with respect to $H_i(t)$. On the other hand, if the value of $H_i(t)$ is large, then it implies that the time average of $\gamma_i(t)$ is far away from that of $R_i(t)$. To fill this gap, it would be better to choose a lower value of $\gamma_i(t)$. In addition, as the selection of auxiliary variables can be separately performed for each application $\forall i \in \mathcal{A}$, this can facilitate a distributed implementation of the decision phase above.

2) Request Admission Control and Routing: For each application $i \in \mathcal{A}$ served by the datacenter, the request admission decisions $R_i(t)$ and routing decisions $(R_{i1}(t), R_{i2}(t), ..., R_{iN}(t))$ as illustrated in Fig. 1 can be decided by maximizing the term (16) in Sec. III-A2. Again, since the admission decisions $R_i(t)$ and routing decisions $R_{ij}(t)$ of different applications are independent from each other, this centralized maximization can be decoupled to be computed concurrently as follows:

$$\begin{aligned} \max_{R_i(t), R_{ij}(t)} \quad & d_i^2 H_i(t) R_i(t) - d_i \sum_{j \in \mathcal{S}} R_{ij}(t) Q_{ij}(t) \quad (20) \\ \text{s.t.} \quad & 0 \le R_i(t) \le A_i(t), \forall i \in \mathcal{A}, \\ & R_i(t) = \sum_{j \in \mathcal{S}} R_{ij}(t). \end{aligned}$$

Different from the previous problem (18), both the control decisions of $R_i(t)$ and $R_{ij}(t)$ in problem (20) need to be determined. We first start from a simple case: if the value of $R_i(t)$ is known in advance, then problem (20) is exactly equivalent to the following problem for making routing decisions:

$$\begin{aligned} \min_{R_{ij}(t)} \quad & d_i \textstyle\sum_{j \in \mathcal{S}} R_{ij}(t) Q_{ij}(t) \quad (21) \\ \text{s.t.} \quad & \textstyle\sum_{j \in \mathcal{S}} R_{ij}(t) = R_i(t), \forall i \in \mathcal{A}. \end{aligned}$$

Insight: The problem (21) is a generalized *min-weight problem*, where the amount of requests routed to server $j \in \mathcal{S}$ for application $i \in \mathcal{A}$ is weighted by the current queue backlog $Q_{ij}(t)$. Hence, for each application $i \in \mathcal{A}$ served by the datacenter, the optimal routing strategy tends to *dispatch as many admitted requests as possible to the VM with least backlogged queue*:

$$R_{ij}(t) = \begin{cases} R_i(t), & j = j_i^*, \\ 0, & \text{else}, \end{cases} \quad (22)$$

where $j_i^* = \arg\min_{j \in \{1,2,...,N\}} Q_{ij}(t)$, *i.e.*, the queue of the i-th VM on server j_i^* is the shortest queue among all the N queues for the i-th type of application. Such a routing policy is an intuitive "*Join the Shortest Queue*" policy for the purpose of load balancing, which is consistent with a recent work on scheduling of cloud computing clusters [22]. Further, it can effectively reduce the response delay of newly admitted requests, as they are preferentially routed to the shortest queues. However, it requires to obtain all the queue backlog information of those VMs serving the i-th type of applications. To mitigate this complexity, we can adopt a recently developed "Power-of-Two-Choices" [22] routing policy, which randomly samples two VMs and routes the application requests to the VM with a smaller queue backlog.

Recall that the optimal value of $R_i(t)$ is still undecided so far. Based on the routing strategy in Eq. (22), the second term of Eq. (20) (*i.e.,* $d_i \sum_{j \in \mathcal{S}} R_{ij}(t) Q_{ij}(t)$) can be rewritten as $d_i R_i(t) Q_{ij_i^*}(t)$. Then, the request admission control decision can be solved as:

$$\begin{aligned} \max_{R_i(t)} \quad & d_i^2 H_i(t) R_i(t) - d_i R_i(t) Q_{ij_i^*}(t) \quad (23) \\ \text{s.t.} \quad & 0 \le R_i(t) \le A_i(t), \forall i \in \mathcal{A}. \end{aligned}$$

The problem (23) is a simple linear programming problem in which the optimal value of $R_i(t)$ is:

$$R_i(t) = \begin{cases} A_i(t), & d_i H_i(t) > Q_{ij_i^*}(t) \\ 0, & \text{else} \end{cases} \quad (24)$$

Insight: This is a simple *threshold-based* admission control strategy. When the backlog of the shortest queue $Q_{ij_i^*}(t)$ is smaller than a threshold $d_i H_i(t)$, then all the newly arrived requests are admitted into the datacenter. Essentially, this not only reduces the value of $H_i(t)$ so as to push γ_i to become closer to r_i, but also increases the datacenter throughput r_i so as to improve the profit. On the other hand, when the backlog of the shortest queue $Q_{ij_i^*}(t)$ is larger than the threshold $d_i H_i(t)$, then all the requests will be denied to ensure the stability of the datacenter.

3) VM Scheduling: In each time slot t, the running or idle state of each VM $(a_{1j}(t), a_{2j}(t), ..., a_{Mj}(t))$ (Sec. II-A) on server $j \in \mathcal{S}$ can be determined by maximizing the term (17) in Sec. III-A2. Observing that the indicator variables $a_{ij}(t)$ are independent among different servers, the centralized maximization can be implemented by each server in a fully *distributed* manner:

$$\begin{aligned} \max_{a_{ij}(t)} \quad & \sum_{i \in \mathcal{A}} Q_{ij}(t) a_{ij}(t) - V \beta P_j(t) \quad (25) \\ \text{s.t.} \quad & a_{ij}(t) \in \{0, 1\}, \forall i \in \mathcal{A}, \forall j \in \mathcal{S}, \end{aligned}$$

where $Q_{ij}(t)$ can be viewed as the weight of the decision variable $a_{ij}(t)$. As the growth of power consumption caused by running each VM is the same under our model (see the definition of $P_j(t)$ in Sec. II-B2), the optimal solution for Eq. (25) would prefer to schedule the VMs with most backlogged queues (*i.e.,* larger $Q_{ij}(t)$ as weights) to the running state. Following this intuition, each server adopts a simple yet effective *greedy strategy* that ranks hosted VMs according to their queue backlogs. Then, it searches from VMs with the most backlogged queues to VMs with the least backlogged queues: (1) If the growth in the sum of backlogs exceeds the growth of power consumption (weighted by $V\beta$) caused by running a certain VM, then the VM is preferentially scheduled to the running state. (2) Such a search process can continue until the growth in the sum of backlogs falls below the growth of power consumption (weighted by $V\beta$) for a certain VM, which is scheduled to the idle state. Then, the other remaining VMs are scheduled to the idle state.

4) Queue Update: Finally, the virtual queues $\mathbf{H}(t)$ can be updated according to Eq. (10), by using the optimal values of $\gamma_i(t)$ and $R_i(t)$ determined by the phases above. Likewise, the actual queues $\mathbf{Q}(t)$ maintained by VMs in the datacenter can be updated according to Eq. (1), based on the optimal values of $R_{ij}(t)$ and $a_{ij}(t)$ derived above.

C. Optimality Analysis

We are now ready to analyze the optimality of **OCA** algorithm, in terms of a well-balanced tradeoff between the profit maximization and strong stability of the datacenter.

Theorem 1: For arbitrary arrival rates of application requests $(\lambda_1(t), \lambda_2(t), ..., \lambda_M(t))$ (possibly exceeding the processing capacity of a datacenter), a datacenter using the **OCA** algorithm with any $V \geq 0$ (the stability-profit tradeoff parameter defined in Sec. III-A1) can guarantee that all the actual and virtual queues are strongly stable over time slots:

$$H_i(t) \leq \frac{V}{d_i} + A_i^{\max}, \forall i \in \mathcal{A}, \tag{26}$$

$$Q_{ij}(t) \leq V + 2d_i A_i^{\max}, \forall i \in \mathcal{A}, \forall j \in \mathcal{S}. \tag{27}$$

Meanwhile, the gap between its achieved time averaged profit and the optimal profit ξ^* is within B_1/V:

$$\liminf_{t\to\infty} \Big\{ \sum_{i\in\mathcal{A}} g(r_i) - \beta \sum_{j\in\mathcal{S}} p_j \Big\} \geq \xi^* - \frac{B_1}{V}, \tag{28}$$

where $\xi^* = \sum_{i=1}^{M} g(r_i^*) - \beta \sum_{j=1}^{N} p_j^*$, r_i^* and p_j^* are the optimal solution to **Problem** (5), and B_1 is a finite constant parameter defined in Lemma 1.

Insight: Theorem 1 provides a strong deterministic guarantee of the upper bounds on backlogs of all the actual and virtual queues in any time slot. Meanwhile, Eq. (28) indicates that the gap between the time averaged profit achieved by **OCA** algorithm and the optimal profit is within $O(1/V)$. Interestingly, as the value of parameter V increases to sufficiently large, the time averaged profit under **OCA** can be pushed arbitrarily close to optimum. However, according to Eq. (26) and Eq. (27), overly aggressive increases of profit can also increase the bounds of queue backlogs. By Little's law, the bounds of response delays for application requests would also increase. Interested readers are referred to our detailed technical report [21] for a complete proof of Theorem 1.

IV. Extended Model: Enforcing a Power Budget

Our optimization framework can be *flexibly* extended to incorporate various design choices and practical requirements of datacenter power-performance tradeoff. For instance, as most real-world datacenters are operated within a certain *power budget* [4], it is important for datacenter operators to improve the *performance (dollar) per watt* [4] by achieving a desired performance level with an enforced power budget.

Specifically, we enforce an additional power budget constraint $p_j \leq p_j^{av}, \forall j \in \mathcal{S}$ (in terms of the time average of normalized server power consumption) within the basic **Problem** (5), while still maintaining a demanded performance requirement on time averaged throughput of applications $r_i \geq r_i^{av}, \forall i \in \mathcal{A}$. This yields an extended optimization model:

$$\begin{aligned} \max \quad & \sum_{i\in\mathcal{A}} g(r_i) - \beta \sum_{j\in\mathcal{S}} p_j \\ \text{s.t.} \quad & r_i^{av} \leq r_i \leq \lambda_i, \quad \forall i \in \mathcal{A}, \\ & r_i \leq N/d_i, \quad \forall i \in \mathcal{A}, \\ & p_j \leq p_j^{av}, \quad \forall j \in \mathcal{S}. \end{aligned} \tag{29}$$

To accommodate the newly introduced constraints, we further define virtual queues $Z_i(t), X_j(t)$ for each application $i \in \mathcal{A}$ and each server $j \in \mathcal{S}$, respectively: $Z_i(t+1) = \max[Z_i(t) - R_i(t), 0] + r_i^{av}$ and $X_j(t+1) = \max[X_j(t) - p_j^{av}, 0] + P_j(t)$. Then, with our control framework based on Lyapunov optimization presented in previous sections, we define the Lyapunov function as: $L(\mathbf{\Theta}(t)) = \frac{1}{2}\Big[\sum_{i=1}^{M} d_i^2 H_i^2(t) + \sum_{i=1}^{M} d_i^2 Z_i^2(t) + \sum_{j=1}^{N} X_j^2(t) + \sum_{i=1}^{M}\sum_{j=1}^{N} Q_{ij}^2(t)\Big]$. According to the definition of the Lyapunov drift $\Delta(\mathbf{\Theta}(t))$ in Sec. III-A1, the corresponding drift-minus-profit expression under the extended model can be rewritten as follows, where $B_2 = \frac{1}{2}[MN + 5\sum_{i=1}^{M}(d_i A_i^{\max})^2 + \sum_{i=1}^{M}(r_i^{av})^2 + \sum_{j=1}^{N}(p_j^{av})^2]$:

$$\begin{aligned} \Delta(\mathbf{\Theta}(t)) - V\mathbb{E}\{\xi(t)|\mathbf{\Theta}(t)\} \leq & \sum_{i\in\mathcal{A}} r_i^{av} Z_i(t) - \sum_{j\in\mathcal{S}} p_j^{av} X_j(t) \\ & + B_2 - \sum_{i\in\mathcal{A}} \mathbb{E}\{V g(\gamma_i(t)) - d_i^2 H_i(t)\gamma_i(t)|\mathbf{\Theta}(t)\} - \\ \sum_{i\in\mathcal{A}} \mathbb{E}\Big\{ d_i^2 R_i(t)(H_i(t) + Z_i(t)) & - \sum_{j\in\mathcal{S}} d_i R_{ij}(t) Q_{ij}(t)|\mathbf{\Theta}(t)\Big\} \\ - \sum_{j\in\mathcal{S}} \mathbb{E}\Big\{ \sum_{i\in\mathcal{A}} Q_{ij}(t) a_{ij}(t) & - P_j(t)(V\beta + X_j(t))|\mathbf{\Theta}(t)\Big\}. \end{aligned} \tag{30}$$

Insight: Based on Eq. (30), we can also design an online control algorithm for the extended **Problem** (29), named **EOCA**, which follows the skeleton of the four phases of **OCA** control operations in Sec. III-B. Specifically, the auxiliary variable selection of **EOCA** is the same as that of **OCA**, which implies that the newly introduced power budget and performance constraints do not affect this phase. Additionally, the routing policy of **EOCA** still follows the "Join the Shortest Queue" policy as in **OCA**. However, **EOCA** differs from **OCA** in the following aspects:

First, for the request admission control and routing phase of **EOCA**, the weights of $R_i(t), \forall i \in \mathcal{A}$ would be increased to $d_i^2(H_i(t) + Z_i(t)), \forall i \in \mathcal{A}$, after introducing the virtual queues $Z_i(t), \forall i \in \mathcal{A}$. This changes the admission control policy to become: if $d_i(H_i(t) + Z_i(t)) > Q_{ij_i^*}(t)$, $R_i(t) = A_i(t)$; otherwise, $R_i(t) = 0$. *Second*, for the VM scheduling phase of **EOCA**, the term $\sum_{i\in\mathcal{A}} Q_{ij}(t)a_{ij}(t) - V\beta P_j(t)$ to be maximized is rewritten as $\sum_{i\in\mathcal{A}} Q_{ij}(t)a_{ij}(t) - V\beta(P_j(t) + X_j(t))$. This adjusts the greedy strategy in **OCA** to become: only when the growth in $\sum_{i\in\mathcal{A}} Q_{ij}(t)a_{ij}(t)$ exceeds the growth of $V\beta(P_j(t) + X_j(t))$ caused by running a certain VM, the VM will be preferentially scheduled to the running state. *Third*, the queue backlogs of $Z_i(t)$, $X_j(t)$ and $Q_{ij}(t)$ are not guaranteed

to be deterministically bounded under **EOCA**, though it can ensure the stability of these queues (*i.e.,* $p_j \leq p_j^{av}, \forall j \in \mathcal{S}$ and $r_i \geq r_i^{av}, \forall i \in \mathcal{A}$) and a bounded gap between its achieved time averaged profit and the optimal profit for the extended **Problem** (29), as given by the following Theorem 2.

***Theorem** 2:* For arbitrary arrival rates of application requests $(\lambda_1(t), \lambda_2(t), ..., \lambda_M(t))$ (possibly exceeding the processing capacity of a datacenter), a datacenter can use **EOCA** algorithm with any $V \geq 0$ to guarantee the stability of queues $Q_{ij}(t), Z_i(t), X_j(t)$ over time slots. Meanwhile, the gap between its achieved time averaged profit and the optimal profit $\xi^\star$ for the extended **Problem** (29) is within B_2/V:

$$\liminf_{t\to\infty} \left\{ \sum_{i\in\mathcal{A}} g(r_i) - \beta \sum_{j\in\mathcal{S}} p_j \right\} \geq \xi^\star - \frac{B_2}{V}, \tag{31}$$

where $\xi^\star = \sum_{i=1}^{M} g(r_i^\star) - \beta \sum_{j=1}^{N} p_j^\star$, $r_i^\star$ and $p_j^\star$ are the optimal solution to the extended **Problem** (29), and B_2 is a finite constant parameter defined in Eq. (30).

Interested readers are referred to our detailed technical report [21] for a complete proof of Theorem 2.

V. Performance Evaluation

We conduct simulations to evaluate our online control algorithm **OCA** and its extensions under an illustrative datacenter scenario. This consists of 100 homogeneous servers, each of which hosts 10 VMs to serve 10 heterogeneous applications, respectively. Specifically, the requests from each application i arrive according to a random process of mean rate λ_i, and different applications have different mean arrival rates and request sizes d_i (Sec. II-A) in Table II. For each application i, we set its peak request arrival rate as $A_i^{\max} = 2\lambda_i$, and the number of newly arrived requests in each time slot is assumed to be uniformly and randomly distributed within $[0, A_i^{\max}]$.

With such a setup, each application is being served by a total of 100 VMs across servers, with an allocated total processing capacity of 100 (in terms of the total queue service rates of these VMs as modeled in Sec. II-A), according to the fair allocation policy of server capacity in Sec. II-A. Observing the capacity demand of each application in terms of $d_i\lambda_i$ in Table II, we find that the capacity demands of requests for applications $1-6$ are within the respective allocated processing capacity, while that for applications $7-10$ exceed the respective allocated processing capacity. We choose a typical setting of exponent parameter $v = 2$ and $\alpha = 0.5$ [16] for the normalized power function in Eq. (3), and an empirical value of parameter $\beta = 0.4$. The following simulations are carried out for $100,000$ time slots.

TABLE II: Request Arrival Rates and Sizes of Different Applications.

App i	1	2	3	4	5	6	7	8	9	10
$\lambda_i(\times 10^3)$	2.5	2	3.5	2	3	2	2.75	2.4	2.6	2.8
$d_i(\times 10^{-2})$	2	3	2	4	3	5	4	5	5	5
$d_i\lambda_i(\times 10)$	5	6	7	8	9	10	11	12	13	14

First, verification of algorithm optimality. Fig. 2 plots the time averaged profit for different values of the control parameter V under our **OCA** algorithm. We observe that:

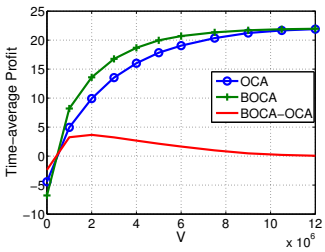

Fig. 2: Time averaged profit versus different values of the control parameter V under OCA and BOCA algorithms.

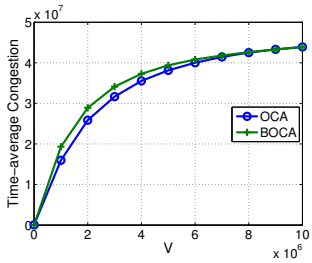

Fig. 3: Time averaged system congestion versus different values of the control parameter V under OCA and BOCA algorithms.

(1) as the value of V increases, the time averaged profit achieved by **OCA** improves significantly and converges to the maximum level for larger values of V. This quantitatively corroborates Theorem 1 in that **OCA** can approach the optimal profit with a diminishing gap $(1/V)$ (captured by Eq. (28)), which also implies a cost-effective tradeoff between power and performance unified by the profit objective (recall **Problem** (5)). However, such an improvement starts to diminish with excessive increases of V, which can adversely aggravate the congestion of queues in the system (captured by Eq. (12)). (2) Furthermore, we enhance the system facility by introducing a routing buffer component for each application (between the request admission and routing control decisions in Fig. 1) to store the admitted requests before they are routed [11]. Compared to **OCA**, the variant algorithm with an enhanced buffering facility denoted as **BOCA** (with detailed derivation in our technical report [21]) can achieve a slightly higher time averaged profit for smaller values of V, as more admitted requests can be buffered (rather than being directly denied) when the current workload of VMs is heavy. However, such a profit gap (marked as BOCA–OCA) will diminish as V grows.

Second, examination of system stability. Fig. 3 plots the time average of queue congestion [5] captured by Eq. (12) for different values of V under both **OCA** and **BOCA**. With the growth of V, the time averaged system congestion with both algorithms increases. Along with Fig. 2, this reflects the tradeoff between system stability and profit maximization, as revealed in Sec. III-A1. In comparison, though **BOCA** gains a higher time averaged profit than **OCA** does, the former, on the other hand, incurs a higher level of system congestion.

Third, online control in response to bursty request arrivals. Different from our simulations above with fixed mean arrival rates of application requests, Fig. 4 plots the fluctuation of the number of running VMs in the system under **OCA**, when the mean request arrival rates of applications vary in a bursty manner. We set the mean request arrival rate of each application as half of their original rate in Table II (*i.e.,* $0.5\lambda_i$) during the first third of the simulation time. Then, the rates abruptly rise to $1.5\lambda_i$ in the next third interval before dropping to the original λ_i in the last third of simulation time. Fig. 4 shows that, even under bursty and unpredictable arrivals of requests, **OCA** is able to quickly adapt to the varying demand by increasing or decreasing the number of running VMs.

Fourth, the effectiveness of admission control. We further

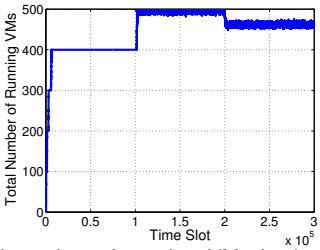

Fig. 4: The total number of running VMs in the system using OCA algorithm over all time slots.

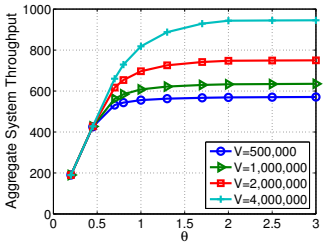

Fig. 5: Total system throughput verses the multiplier θ of request arrival rates, with the admission control of the OCA algorithm.

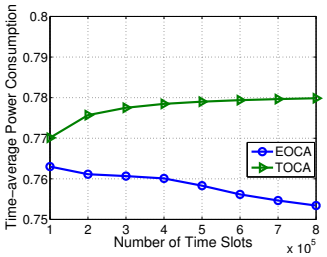

Fig. 6: Average power consumption (normalized) of servers versus the length of simulation time, with EOCA against TOCA.

examine the effects of admission control with **OCA** on the total system throughput (quantified as $\sum_{i=1}^{M} d_i r_i$) by varying the request arrival rates in Fig. 5. By tuning the mean request arrival rates of all applications by a factor of θ times the original setting in Table II, we observe that as long as θ is relatively small, the total system throughput increases linearly with the increasing request arrival rates, despite different values of V. The rationale is that under such a condition, all application requests can be admitted by **OCA** according to the threshold-based admission control policy in Sec. III-B. With even higher request arrival rates, the aggregate throughput achieved by the system under different values of V will gradually become stabilized, which shows that **OCA** can prevent the system from being overwhelmed by excessive requests.

Finally, effectiveness of power budget enforcement. Fig. 6 compares the average power consumption of servers under our extended algorithm **EOCA** with both power budget enforcement and throughput requirement discussed in Sec. IV, against a counterpart algorithm with only throughput requirement (marked as TOCA). We set the time averaged throughput requirement of both **EOCA** and TOCA as $r_i^{av} = 2,000, \forall i \in \mathcal{A}$. The normalized time averaged power budget constraint of **EOCA** for half of the servers in the system is set as 0.7, while that of the other half is set as 0.8. The other parameters are set as the same for both algorithms. As expected, we observe that, in the long run, **EOCA** indeed outperforms TOCA with respect to the reduction of server power consumption.

VI. Conclusion

In response to dynamic and unpredictable user requests from heterogeneous applications served by a SaaS cloud datacenter platform, this paper designs and analyzes an optimal online control framework to balance the tradeoff between the throughput performance and power consumption for processing requests. By applying rigorous Lyapunov optimization approaches, our online control framework can independently and simultaneously make decisions on three important control decisions of such a cloud platform, including request admission control, routing, and VM scheduling. In particular, our control framework is shown to be flexibly extensible to explore various design choices and practical constrains of a datacenter, such as enforcing a certain power budget towards better performance (dollar) per watt. Through in-depth mathematical analysis and various simulations, we demonstrate that our online control framework can approach a time averaged profit — unifying a cost-effective power-performance tradeoff — that is arbitrarily close to optimum, while still maintaining strong system stability, in term of the robustness and adaptivity to time-varying and bursty application requests.

References

[1] Google Apps. [Online]. Available: http://www.googleapps.com
[2] Salesforce. [Online]. Available: http://www.salesforce.com
[3] Campaign Monitor. [Online]. Available: http://www.campaignmonitor.com/pricing
[4] X. Fan, W. Weberand, and L. Barroso, "Power Provisioning for A Warehouse-Sized Computer," vol. 35, no. 2, pp. 13–23, 2007.
[5] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.
[6] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic Right-Sizing for Power-Proportional Data Centers," in *Proc. of IEEE INFOCOM*, 2011.
[7] L. Rao, X. Liu, L. Xie, and W. Liu, "Minimizing Electricity Cost: Optimization of Distributed Internet Data Centers in a Multi-Electricity-Market Environment," in *Proc. of IEEE INFOCOM*, 2010.
[8] Z. Liu, M. Lin, A. Wierman, and L. A. SH Low, "Greening Geographic Load Balancing," in *Proc. of ACM Sigmetrics*, 2011.
[9] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing Server Energy and Operational Costs in Hosting Centers," in *Proc. of ACM SIGMETRICS*, 2005.
[10] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical Profiling-based Techniques for Effective Power Provisioning in Data Centers," in *Proc. of EuroSys*, 2009.
[11] R. Urgaonkar, U. Kozat, K. Igarashi, and M. Neely, "Dynamic Resource Allocation and Power Management in Virtualized Data Centers," in *Proc. of IEEE NOMS*, 2010.
[12] D. Xu and X. Liu, "Geographic Trough Filling for Internet Datacenters," in *Proc. of IEEE INFOCOM*, 2012.
[13] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data Centers Power Reduction: A Two Time Scale Approach for Delay Tolerant Workloads," in *Proc. of IEEE INFOCOM*, 2012.
[14] A. Greenberg, J. Hamilton, D. Malta, and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," in *Proc. of ACM Sigcomm Computer Communication Review*, 2008.
[15] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and Performance Management of Virtualized Computing Environments via Lookahead Control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.
[16] L. A. A. Wierman and A. Tang, "Power-Aware Speed Scaling in Processor Sharing Systems," in *Proc. of IEEE INFOCOM*, 2009.
[17] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services," in *Proc. of NSENIX NSDI*, 2008.
[18] N. Rasmussen, "Electrical Efficiency Modeling for Data Centers," in *White Paper No. 113*.
[19] The Green Grid. [Online]. Available: http://www.thegreengrid.org
[20] M. J. Neely, "Delay-Based Network Utility Maximization," in *Proc. of IEEE INFOCOM*, 2010.
[21] Z. Zhou, F. Liu, H. Jin, B. Li, B. Li, and H. Jiang, "On Arbitrating the Power-Performance Tradeoff in SaaS Clouds," HUST Technical report, http://grid.hust.edu.cn/fmliu/oca.pdf, July 2012.
[22] S. T. Theja, R. Srikant, and L. Ying, "Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters," in *Proc. of IEEE INFOCOM*, 2012.