On Arbitrating the Power-Performance Tradeoff in SaaS Clouds

Fangming Liu, *Member, IEEE*, Zhi Zhou, Hai Jin, *Senior Member, IEEE*, Bo Li, *Fellow, IEEE*, Baochun Li, *Senior Member, IEEE*, and Hongbo Jiang, *Member, IEEE*

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-folded: 1) We maximize the operating revenue when serving heterogeneous SaaS applications with unpredictable user requests. 2) We minimize the power consumption when processing the user requests. To achieve these objectives, we construct a unified profit-maximizing objective to jointly consider revenue and cost in an economic view. An offline solution to maximize the supreme bound of the objective is first developed, to 1) justify the validity of our theoretical model, and 2) establish a benchmark to examine the effectiveness of other control solutions. As a highlight of our contributions, we take advantage of the Lyapunov optimization techniques to design and analyze an optimal yet practical control framework, which makes online decisions on request admission control, routing, and virtual machine (VMs) scheduling. Our control framework can accommodate a variety of design choices and operational requirements in a datacenter. Specifically, buffering facilities can be introduced to alleviate the bursty admitted requests and to improve the robustness of the system, and a power budget can be enforced to improve the datacenter performance (dollar) per watt. Our mathematical analyses and simulations have demonstrated both the optimality (in terms of the cost-effective power-performance tradeoff) and stability (in terms of robustness and adaptivity to time-varying and bursty user requests) achieved by our proposed control framework.

Index Terms—SaaS cloud, datacenter, power-performance tradeoff, Lyapunov optimization, online control

1 Introduction

Software-as-a-service (SaaS) cloud platforms, such as the 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 transaction or request. For example, a popular cloud-based email marketing application, called the Campaign Monitor [3], charges users according to the number of processed campaigns and delivered recipients. By following the law of *diminishing marginal utility* [4] 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.[1] 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.

Let us consider a typical SaaS cloud platform in Fig. 1. Due to risks of going beyond the processing capacity, a front-end proxy is needed to admit requests. We also need a load dispatcher to redirect admitted requests across a large pool of servers with constrained power budgets [5], each of which is virtualized as 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-folded: 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 given time; 2) how to distribute the admitted requests from different applications across a

- *F. Liu, Z. Zhou, and H. Jin are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {fmliu, zhiz, hjin}@mail.hust.edu.cn.*
- *B. Li is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon 999077, Hong Kong. E-mail: bli@cse.ust.hk.*
- *B. Li is with the Department of Electrical and Computer Engineering, University of Toronto, 10 Kings College Road, Toronto, ON M5S 3G4, Canada. E-mail: bli@eecg.toronto.edu.*
- *H. Jiang is with the Department of Electronics and Information Engineering, Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China. E-mail: hongbojiang2004@gmail.com.*

Manuscript received 23 Apr. 2013; revised 6 Aug. 2013; accepted 7 Aug. 2013. Date of publication 15 Aug. 2013; date of current version 17 Sept. 2014. Recommended for acceptance by R. Kwok.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.208

1. Even for long-term SaaS commitments with an upfront and flat payment, the law of diminishing marginal utility is still applicable in general, in the form of a discount.

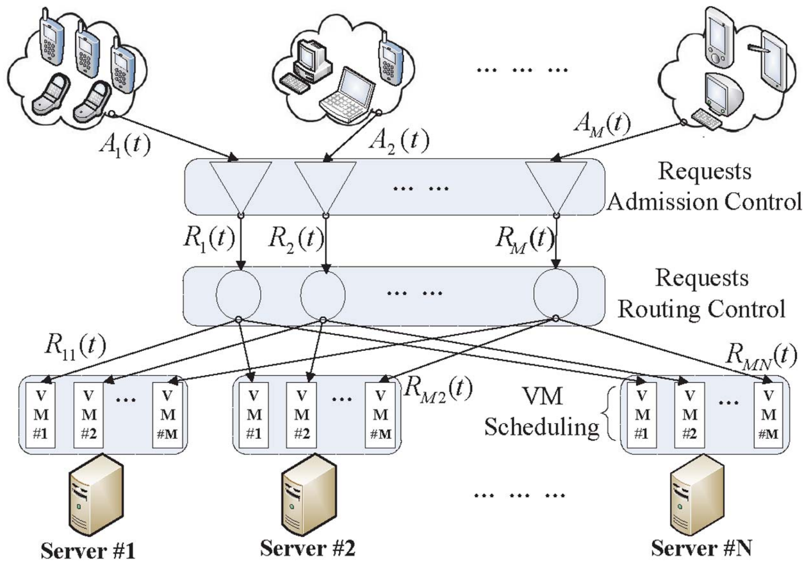


Fig. 1. Basic virtualized datacenter with three control decisions: 1) admission control that accepts (or denies) arrived requests from different applications, 2) routing control that dispatches admitted requests to VMs hosted on different servers in a datacenter, and 3) scheduling of VMs by switching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

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 the server power.

To address these challenges, this paper takes advantage of the *Lyapunov optimization* techniques [6], [7] to 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 flexibly extended to explore various design choices and operational requirements of a realistic datacenter. From a design perspective, to improve the robustness of the datacenter system, some designers may introduce buffering facilities to mitigate bursty admitted requests. Through redefining the Lyapunov function to incorporate the buffering facilities, we show that our framework can be enhanced to accommodate such a design choice. From an operational perspective, as most real-world datacenters are operated within a certain power budget to improve the performance (dollar) per watt [5], we demonstrate how our model and framework can be extended to accomplish this goal, by achieving a desired performance level with a particular 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.

The rest of this paper is organized as follows. In Section 2, we survey the related works. In Section 3, we present the basic power-performance tradeoff model [8]. Different from and complementary to our preliminary work [8], Section 4 maximizes the supreme bound of the time averaged profit via an offline solution. This not only justifies the reliability of the power-performance tradeoff model, but also provides a profit benchmark for the subsequent proposed online control framework. We further construct a new optimal online control framework to approach a time averaged profit which is arbitrarily close to optimum in Section 5. We demonstrate that our framework can be extended to incorporate various design choices in the datacenter power-performance tradeoff, such as additional request buffering facilities in Section 6 to improve the system robustness as compared with our preliminary work [8], and an enforced power budget in Section 7. We evaluate our proposed framework in Section 8 with more comprehensive simulations than our preliminary work [8], and conclude the paper in Section 9.

2 Related Work

While recognizing the significance of many existing works (e.g., [9], [10], [11]) 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.* [12] 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.* [13] 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 [14], [15], [16], [17] 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 with a nonlinear utility function based on the law of diminishing marginal utility (Section 3.2.3), rather than a simple linear utility function [14] that does not reflect reality in general. 2) Our framework can be extended to focus on improving the performance (dollar) per watt [5] at datacenters, by achieving a desired system throughput level with a certain power budget.

3 Power-Performance Tradeoff Model

In this section, we first formulate the basic datacenter model in Fig. 1, consisting of N homogeneous servers $\mathcal{S} = \{1, 2, \ldots, N\}$, each of which is virtualized to M virtual machines (VMs) to serve M types of heterogeneous applications $\mathcal{A} = \{1, 2, \ldots, M\}$ 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 [9], we consider a discrete time-slotted system where the time slot length can range from hundreds of milliseconds to minutes [11]. In every time slot t $(=, 1, 2, \ldots, \tau, \ldots)$, 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.) [6] 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 [18] and potentially exceed the current available processing capacity of a datacenter), *our model does not assume any a priori knowledge of the statistics* of $A_i(t), \forall\, i \in \mathcal{A}, \forall\, t$.

3.1 Control Decisions

Under the datacenter workload model above, we focus on *three* important control decisions to be made, introduced in Fig. 1 and Section 1, along with the key notations used in Table 1 which is available in Appendix A of the online supplementary file which is available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.208.

3.1.1 Admission Control of Application Requests

In each time slot t, given the number of newly arrived requests of each application $A_i(t), \forall\, i \in \mathcal{A}$, the forefront 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:[2] $\{0 \leq R_i(t) \leq A_i(t), \forall i \in \mathcal{A}, \forall t\}$. On the other hand, for those denied a portion of requests $[A_i(t) - R_i(t)], \forall\, i \in \mathcal{A}$, the system can return negative response signals to the corresponding users, who may choose to resend their requests to the datacenter [14].

3.1.2 Routing 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 core control decision is to route (dispatch) the admitted requests from each application $R_i(t), \forall\, i \in \mathcal{A}$ 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: $R_i(t) = \sum_{j=1}^{N} R_{ij}(t), \forall\, i \in \mathcal{A}, \forall\, j \in \mathcal{S}, \forall\, t$. Essentially, such a routing control logic can be implemented as the *load balancer(s)* [18] of realistic datacenters, though our general model is not restricted to any specific design choice and implementation.

3.1.3 Scheduling of VMs

While the amount of routed requests $R_{ij}(t)$ are waiting in the aforementioned 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[3] 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

2. 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. 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 [19].

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:

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

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

3.2 Characterizing the Power-Performance Tradeoff

3.2.1 System Throughput

For the 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 throughputr_i* of a datacenter as

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

As with r_i, we define the time averaged value a_{ij} as

$$a_{ij} = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\big\{a_{ij}(\tau)\big\},$$

according to the definition above, a_{ij} is the frequency to make the i-th VM on the j-th server running, while can also be viewed as the capacity consumed by that VM during the t time slots.

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) $r_i \le \lambda_i$, 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) $d_i r_i \le \sum_{j=1}^{N} a_{ij}$, as the normalized time averaged throughput $d_i r_i$ cannot exceed the overall processing capacity allocated for the corresponding application $i \in \mathcal{A}$.

3.2.2 Power Consumption

Here we focus on a basic power consumption model of servers where the CPU processing capacity is the main bottleneck, in the case of serving computation-intensive applications. Yet, our model can be extended to incorporate other components (e.g., memory, disk and network I/O) via informed ways [14], such as representing multi-dimensional resources of a server as a vector in our formulation, and amending the aforementioned request sizes and service rates accordingly.

Specifically, it has been shown by recent studies [16], [20] 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

$$P(s) = \alpha s^v + (1 - \alpha), \tag{3}$$

where the exponent parameter v is empirically determined as $v \ge 1$ in practice (e.g., a typical value is $v = 2$ [20]). With another parameter $\alpha \in [0, 1]$, the term $(1 - \alpha)$ represents the normalized power consumption of an idle server. Practical measurements [21] 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 M VMs with the fair capacity allocation policy described in Section 3.1, its normalized CPU load and corresponding power consumption in time slot t are given as

$$\begin{aligned} s_j(t) &= \frac{\sum_{i=1}^{M} a_{ij}(t)}{M}, \\ P_j(t) &= \alpha \left(\frac{\sum_{i=1}^{M} a_{ij}(t)}{M} \right)^v + (1 - \alpha). \end{aligned}$$

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

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

then, the metric $\sum_{j=1}^{N} p_j$ is the overall power consumption of all servers[4], hopefully being minimized.

3.2.3 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.

4. Given recent reports [22] 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.

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 the Green Grid [23]. 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 [18].

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 [4]. 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 [14], 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:

$$\begin{aligned} \max \quad & \sum_{i=1}^{M} g(r_i) - \beta \sum_{j=1}^{N} p_j \\ \text{s.t.} \quad & 0 \le r_i \le \lambda_i, \quad \forall\, i \in \mathcal{A}, \\ & d_i r_i \le \sum_{j=1}^{N} a_{ij}, \quad \forall\, i \in \mathcal{A}, \end{aligned} \tag{5}$$

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

Intuitively, one may attempt to solve the problem above directly. Before attempting this, we should rewrite p_j in the form of a_{ij}, and thus to constrain the power consumption to the admission rate. As $P_j(t)$ is nonlinear to $a_{ij}(t)$, it is impossible to establish the equation between p_j and a_{ij}. Fortunately, since $P_j(t)$ is convex to $a_{ij}(t)$, according to *Jesen's inequality*, we have

$$p_j \ge \alpha \left(\frac{\sum_{i=1}^{M} a_{ij}}{M} \right)^2 + (1 - \alpha). \tag{6}$$

As we cannot obtain the explicit form of p_j, it is impractical to maximize the time averaged profit directly. Instead, we propose an offline control strategy to maximize the supreme bound of the time averaged profit in the next section.

4 Maximizing the Supreme Bound

In this section, we resort to the theoretical analysis to design an offline control strategy, which makes stationary decisions on admission control and VM scheduling, and thus to maximize the supreme bound of the time averaged profit. Such an offline control solution helps to justify the reliability of our basic power-performance tradeoff model. Furthermore, the supreme bound of the time averaged profit can be used as a profit benchmark, and thus to examine the effectiveness of alternative control solutions.

Based on the inequality (6) in Section 3.2.3, we maximize the supreme bound of the time averaged profit

$$\begin{aligned} \max_{r_i, a_{ij}} \quad & \sum_{i=1}^{M} \log(1 + d_i r_i) - \frac{\alpha\beta}{M^2} \sum_{j=1}^{N} \left(\sum_{i=1}^{M} a_{ij} \right)^2 + N(1 - \alpha) \\ \text{s.t.} \quad & 0 \le r_i \le \lambda_i \\ & d_i r_i \le \sum_{j=1}^{N} a_{ij} \\ & 0 \le a_{ij} \le 1. \end{aligned} \tag{7}$$

First, note that the objective function is non-decreasing to $a_{ij}, \forall\, i \in \mathcal{A}, \forall\, j \in \mathcal{S}$. Hence, to maximize the objective function, we should choose

$$\sum_{j=1}^{N} a_{ij} = d_i r_i. \tag{8}$$

Since $\sum_{j=1}^{N} a_{ij}$ is the processing capacity allocated to the i-th application by the corresponding VMs, the equation above indicates that to maximize the profit, the processing capacity allocated to each application should be completely consumed to process the admitted requests, no power is consumed without processing any request and thus the power cost is minimized.

Then, to simplify the objective function, we first take *Cauchy's inequality* $(\sum_{k=1}^{n} a_k^2 \sum_{k=1}^{n} b_k^2 \ge (\sum_{k=1}^{n} a_k b_k)^2)$ to bound the complicated term $\sum_{j=1}^{N} (\sum_{i=1}^{M} a_{ij})^2$,yielding

$$\sum_{j=1}^{N} \left(\sum_{i=1}^{M} a_{ij} \right)^2 \ge \frac{\left(\sum_{j=1}^{N} \sum_{i=1}^{M} a_{ij} \right)^2}{\sum_{j=1}^{N} 1^2} = \frac{\left(\sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij} \right)^2}{N},$$

according to Cauchy's inequality, the equation above is established if and only if

$$\sum_{i=1}^{M} a_{i1} = \sum_{i=1}^{M} a_{i2} = \cdots = \sum_{i=1}^{M} a_{iN}. \tag{9}$$

Sine $\sum_{i=1}^{M} a_{ij}$ is the processing capacity consumed by the j-th server, the equation above implies that each server consumes the same amount of processing capacity. Furthermore, note that the term $N(1 - \alpha)$ is constant, then the **Problem** (7) is now transformed to the following simplified problem:

$$\begin{aligned} \max_{r_i} \quad & f = \sum_{i=1}^{M} \log(1 + d_i r_i) - \frac{\alpha\beta}{M^2 N} \left(\sum_{i=1}^{M} d_i r_i \right)^2 \\ \text{s.t.} \quad & 0 \le r_i \le \min\left[\lambda_i, \frac{N}{d_i} \right]. \end{aligned} \tag{10}$$

To make it easier to derive the peak value of the objective function f, let $x_i = d_i r_i$, differentiating f with respect to each $x_k, \forall\, k \in \mathcal{A}$, yielding

$$\frac{\partial f}{\partial x_k} = \frac{1}{1 + x_k} - \frac{2\alpha\beta}{M^2 N} \sum_{i=1}^{M} x_i, \ \ for\ k = 1, 2, \ldots, M.$$

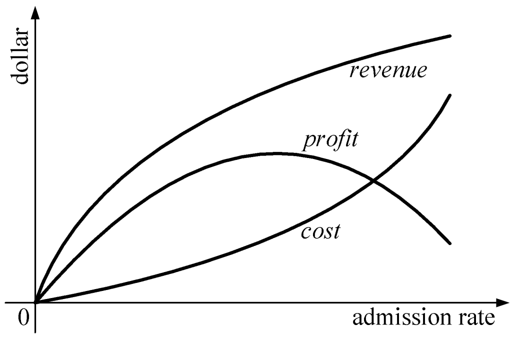


Fig. 2. Illustration of the optimal solution r_i, which reflects the shape of revenue, cost and profit.

For each application $k \in \mathcal{A}$, let $\frac{\partial f}{\partial x_k} = 0$. Note that the term $\sum_{i=1}^{M} x_i$ is identical to each application k, hence, we have the following equation:

$$x_1 = x_2 = \cdots = x_M,$$

then, $\sum_{i=1}^{M} x_i = Mx_1$, plugging it into $\frac{\partial f}{\partial x_1} = 0$, yielding

$$\frac{\partial f}{\partial x_1} = \frac{1}{1+x_1} - \frac{2\alpha\beta}{MN} x_1 = 0.$$

Solving it yields the following solution:

$$x_1 = x_2 = \cdots = x_M = \frac{-\alpha\beta + \sqrt{\alpha^2\beta^2 + 2\alpha\beta MN}}{2\alpha\beta},$$

where the other negative root is discarded. Recall that $x_i = d_i r_i$, so we have

$$r_i = \frac{-\alpha\beta + \sqrt{\alpha^2\beta^2 + 2\alpha\beta MN}}{2\alpha\beta d_i}.$$

Before reaching the peak point, the objective function is non-decreasing, while after reaching the peak point, the objective function is decreasing. Hence, taking the additional constraint $0 \le r_i \le \min[\lambda_i, \frac{N}{d_i}]$ into consideration, yielding the following optimal solution $r_i, \forall i \in \mathcal{A}$, which can be viewed as the decision of admission control:

$$r_i = \min\left[\lambda_i, \frac{N}{d_i}, \frac{-\alpha\beta + \sqrt{\alpha^2\beta^2 + 2\alpha\beta MN}}{2\alpha\beta d_i}\right].$$

Insight: Fig. 2 illustrates the tendency of the revenue, cost and profit as the admission rate increases. To support the increasing admission rate, the cost that is incurred by the power consumption increases more and more rapidly, while the corresponding revenue brought by processing the increasing requests grows more and more slowly. Therefore, the profit increases initially and then decreases as shown in Fig. 2, and there exists a certain admission rate that maximizes the profit.

Meanwhile, to maximize the objective function, $a_{ij}, \forall i \in \mathcal{A}, \forall j \in \mathcal{S}$ should satisfy Eq. (8). Besides, based on Eqs. (8) and (9) which can be viewed as the decision of VM scheduling, we further have:

$$\sum_{i=1}^{M} a_{i1} = \sum_{i=1}^{M} a_{i2} = \cdots = \sum_{i=1}^{M} a_{iN} = \frac{\sum_{i=1}^{M} d_i r_i}{N}. \tag{11}$$

The equation above suggests that the admitted requests are fairly allocated to each server, which can be viewed as load balancing. However, note that according to Eqs. (8) and (11), the optimal solution a_{ij} is not unique, i.e., there exist a variety of optimal VM scheduling decisions.

The optimal offline control solution above is reasonable and reliable. However, for realistic datacenters, there is a challenge when applying this solution: 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. In response, we seek to design an *online* control algorithm in Section 5, which is able to efficiently make decisions on all three important control decisions.

5 Constructing an Online Control Framework

In response to the challenges of **Problem** (5), we take advantage of the Lyapunov optimization techniques [6] 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 the optimum of **Problem** (5), while keeping the system stable.

5.1 Problem Transformation Using Lyapunov Optimization

Given that the function $g(r_i)$ is nonlinear, we are inspired by a recent technique [24] to transform *Problem*(5) to the following framework of Lyapunov optimization **Problem** (12), 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:

$$\max \quad \sum_{i=1}^{M} g(\gamma_i) - \beta \sum_{j=1}^{N} p_j \tag{12}$$

$$\text{s.t.} \quad \gamma_i \le r_i, \quad \forall i \in \mathcal{A} \tag{13}$$

$$d_i r_i \le \sum_{j=1}^{N} a_{ij}, \quad \forall i \in \mathcal{A}. \tag{15}$$

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 (13) into a queue stability problem [6]. 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:

$$H_i(t+1) = \max[H_i(t) - R_i(t), 0] + \gamma_i(t), \tag{16}$$

where $\gamma_i(t)$ denotes a process of non-negative auxiliary variables that the admission control of the datacenter system (Section 3.1) will determine in every time slot, which satisfies the constraint (17). The time average of each $\gamma_i(t)$ is defined as $\gamma_i = \lim_{t\to\infty}(\frac{1}{t})\sum_{\tau=0}^{t-1}\mathbb{E}\{\gamma_i(\tau)\}$

$$0 \le \gamma_i(t) \le A_i^{\max}. \tag{17}$$

Insight: Intuitively, the auxiliary variables $\gamma_i(t)$ can be viewed as the "arrivals" of the virtual queues $H_i(t)$, while $R_i(t)$ can be viewed as the service rate of such virtual queues. The constraints (13) 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 (16) 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 $\tau \in \{0, 1, \ldots, t-1\}$ 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

$$\lim_{t\to\infty}\frac{\mathbb{E}\{H_i(t)\}}{t} + r_i \ge \gamma_i. \tag{18}$$

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 virtual queues $H_i(t)$ in Theorem 1 later). Plugging this into (18) yields $r_i \ge \lambda_i$, so that the constraint (13) can be satisfied.

5.1.1 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 (Section 3.1). 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. (16)), 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

$$L(\mathbf{\Theta}(t)) = \frac{1}{2}\left[\sum_{i=1}^{M} d_i^2 H_i^2(t) + \sum_{i=1}^{M}\sum_{j=1}^{N} Q_{ij}^2(t)\right]. \tag{19}$$

This represents a scalar metric of *queue congestion* [6] 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* [6]

$$\Delta(\mathbf{\Theta}(t)) = \mathbb{E}\{L(\mathbf{\Theta}(t+1)) - L(\mathbf{\Theta}(t)) | \mathbf{\Theta}(t)\}.$$

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:

$$\Delta(\mathbf{\Theta}(t)) - V\mathbb{E}\left\{\sum_{i=1}^{M} g(\gamma_i(t)) - \beta\sum_{j=1}^{N} P_j(t)|\mathbf{\Theta}(t)\right\}. \tag{20}$$

Insight: The control parameter $V(\ge 0)$ represents *a design knob of the stability-profit tradeoff*, i.e., how much we shall emphasize the profit maximization (**Problem** (12)) 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.

5.1.2 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. (20), which requires the following Lemma.

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* $B_1 \triangleq \frac{MN+3\sum_{i=1}^{M}(d_i A_i^{\max})^2}{2}$ *is a finite constant parameter.*

$$\Delta(\mathbf{\Theta}(t)) \le B_1 - \sum_{i=1}^{M} d_i^2 H_i(t)\mathbb{E}\{R_i(t) - \gamma_i(t)|\mathbf{\Theta}(t)\}$$
$$- \sum_{i=1}^{M}\sum_{j=1}^{N} Q_{ij}(t)\mathbb{E}\big\{a_{ij}(t) - d_i R_{ij}(t)|\mathbf{\Theta}(t)\big\}. \tag{21}$$

Interested readers are referred to Appendix B for a complete proof of Lemma 1, available in the online supplemental material.

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. (21) yields an infimum bound of the *drift-minus-profit* expression of the datacenter system

$$\Delta(\mathbf{\Theta}(t)) - V\mathbb{E}\left\{\sum_{i=1}^{M} g(\gamma_i(t)) - \beta\sum_{j=1}^{N} P_j(t)|\mathbf{\Theta}(t)\right\} \le B_1$$
$$- \sum_{i=1}^{M}\mathbb{E}\big\{Vg(\gamma_i(t)) - d_i^2 H_i(t)\gamma_i(t)|\mathbf{\Theta}(t)\big\} \tag{22}$$
$$- \sum_{i=1}^{M}\mathbb{E}\left\{d_i^2 H_i(t)R_i(t) - \sum_{j=1}^{N} d_i R_{ij}(t)Q_{ij}(t)|\mathbf{\Theta}(t)\right\} \tag{23}$$
$$- \sum_{j=1}^{N}\mathbb{E}\left\{\sum_{i=1}^{M} Q_{ij}(t)a_{ij}(t) - V\beta P_j(t)|\mathbf{\Theta}(t)\right\}. \tag{24}$$

5.2 An Optimal Online Control Algorithm

Instead of directly minimizing the *drift-minus-profit* expression in Eq. (20) that involves implicit $\max[*]$ terms in both Eqs. (1) and (16), we seek to design an optimal **Online Control Algorithm (OCA)** to minimize its infimum bound given above (i.e., equivalent to maximizing the terms (22), (23), (24) on the right-hand-side), without undermining the optimality and performance of the algorithm according to [6]. Interestingly, we will show that the maximization of the terms (22), (23), (24) 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* observations 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,
4. queue update.

Recall that the phases (2) and (3) make corresponding decisions for the three fundamental control decisions that we described in Section 3.1 and Fig. 1.

5.2.1 Auxiliary Variable Selection

For each application $i \in \mathcal{A}$ served by the datacenter, we determine $\gamma_i(t)$ by maximizing the term (22) in Section 5.1.2. Fortunately, as the decision variables $\gamma_i(t)$ are independent among applications, such centralized maximization can be decoupled to be computed concurrently as follows

$$\begin{aligned} \max_{\gamma_i(t)} \quad & V\log(1+d_i\gamma_i(t)) - d_i^2 H_i(t)\gamma_i(t) \\ \text{s.t.} \quad & 0 \le \gamma_i(t) \le A_i^{\max}, \forall\, i \in \mathcal{A}. \end{aligned} \tag{25}$$

Differentiating the objective function above with respect to $\gamma_i(t)$ can yield the peak value of the objective function when $\gamma_i(t) = \frac{V}{d_i^2 H_i(t)} - \frac{1}{d_i}$. By taking the constraint (17) into consideration, we obtain the optimal solution to problem (25):

$$\gamma_i(t) = \begin{cases} 0, & H_i(t) > \frac{V}{d_i} \\ \frac{V}{d_i^2 H_i(t)} - \frac{1}{d_i}, & \frac{V}{d_i^2 A_i^{\max} + d_i} \le H_i(t) \le \frac{V}{d_i} \\ A_i^{\max}, & H_i(t) < \frac{V}{d_i^2 A_i^{\max} + d_i}. \end{cases} \tag{26}$$

Insight: The stepped solution above is directly related to the value of $H_i(t)$. Recall from Eq. (18) 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.

5.2.2 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 $(R_{i1}(t), R_{i2}(t), \dots, R_{iN}(t))$ as illustrated in Fig. 1 can be decided by maximizing the term (23) in Section 5.1.2. 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:

$$\begin{aligned} \max_{R_i(t), R_{ij}(t)} \quad & d_i^2 H_i(t) R_i(t) - d_i \sum_{j=1}^{N} R_{ij}(t) Q_{ij}(t) \\ \text{s.t.} \quad & 0 \le R_i(t) \le A_i(t), \forall\, i \in \mathcal{A}, \\ & R_i(t) = \sum_{j=1}^{N} R_{ij}(t). \end{aligned} \tag{27}$$

Different from the previous problem (25), both the control decisions of $R_i(t)$ and $R_{ij}(t)$ in problem (27) need to be determined. We first start from a simple case: if the value of $R_i(t)$ is known in advance, then problem (27) is exactly equivalent to the following problem for making routing decisions:

$$\begin{aligned} \min_{R_{ij}(t)} \quad & d_i \sum_{j=1}^{N} R_{ij}(t) Q_{ij}(t) \\ \text{s.t.} \quad & \sum_{j=1}^{N} R_{ij}(t) = R_i(t), \forall\, i \in \mathcal{A}. \end{aligned} \tag{28}$$

Insight: The problem (28) 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 the least backlogged queue*

$$R_{ij}(t) = \begin{cases} R_i(t), & j = j_i^*, \\ 0, & \text{else}, \end{cases} \tag{29}$$

where $j_i^* = \arg\min_{j \in \{1,2,\dots,N\}} Q_{ij}(t)$, 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 [25]. 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" [25] 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. (29), the second term of Eq. (27) (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

$$\begin{aligned} \max_{R_i(t)} \quad & d_i^2 H_i(t) R_i(t) - d_i R_i(t) Q_{ij_i^*}(t) \\ \text{s.t.} \quad & 0 \le R_i(t) \le A_i(t), \forall\, i \in \mathcal{A}. \end{aligned} \tag{30}$$

The problem (30) is a simple linear programming problem in which the optimal value of $R_i(t)$ is

$$R_i(t) = \begin{cases} A_i(t), & d_i H_i(t) > Q_{ij_i^*}(t) \\ 0, & \text{else}. \end{cases} \tag{31}$$

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 system. The intuition is to *prevent the system with considerable backlogged requests from being overloaded by newly arrived requests*.

5.2.3 VM Scheduling

In each time slot t, the running or idle state of each VM $(a_{1j}(t), a_{2j}(t), \ldots, a_{Mj}(t))$ (Section 3.1) on server $j \in \mathcal{S}$ can be determined by maximizing the term (24) in Section 5.1.2. 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

$$\begin{aligned} \max_{a_{ij}(t)} \quad & \sum_{i=1}^{M} Q_{ij}(t) a_{ij}(t) - V\beta P_j(t) \\ \text{s.t.} \quad & a_{ij}(t) \in \{0,1\}, \forall\, i \in \mathcal{A}, \forall\, j \in \mathcal{S}, \end{aligned} \tag{32}$$

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 Section 3.2.2), the optimal solution for Eq. (32) would prefer to schedule those VMs with the 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.

5.2.4 Queue Update

Finally, the virtual queues $\mathbf{H}(t)$ can be updated according to Eq. (16), 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.

5.3 Optimality Analysis

We further analyze the optimality of the proposed online control algorithm, in terms of a well-balanced tradeoff between the profit maximization and strong stability of the datacenter system. (i.e., the stability-profit tradeoff characterized in Section 5.1.1).

Theorem 1. *For arbitrary arrival rates of application requests $(\lambda_1(t), \lambda_2(t), \ldots, \lambda_M(t))$ (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 Section 5.1.1) can guarantee that all the actual and virtual queues are strongly stable over time slots*

$$H_i(t) \leq \frac{V}{d_i} + A_i^{\max}, \forall\, i \in \mathcal{A}, \tag{33}$$

$$Q_{ij}(t) \leq V + 2d_i A_i^{\max}, \forall\, i \in \mathcal{A}, \forall j \in \mathcal{S}. \tag{34}$$

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

$$\liminf_{t\to\infty} \left\{ \sum_{i=1}^{M} g(r_i) - \beta \sum_{j=1}^{N} p_j \right\} \geq \xi^* - \frac{B_1}{V}, \tag{35}$$

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. (35) 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 the optimum. However, according to Eqs. (33) and (34), overly aggressive increases of profit can increase the bounds of queue backlogs. By Little's law [6], the bounds of response delays for application requests would also increase. The proof of Theorem 1 can be found in Appendix C of the online supplemental material.

6 Extension: Buffering the Requests

In a realistic datacenter, to satisfy some practical requirements, the designer of the datacenter may have additional design choices based on the typical cloud platform shown in Fig. 1. For example, to alleviate the impact of bursty arrivals (e.g., the flash crowd phenomenon in Internet content distribution and video streaming [26], [27]) and thus to improve the robustness of the datacenter system, the designer may prefer to store the excessive admitted requests for each application in a buffering facility before part of them are routed to the corresponding VMs. In this section, we first show that the Lyapunov-based optimization framework can be enhanced to accommodate this additional design choice in realistic datacenters. Then, we present the corresponding buffer-enhanced optimal control algorithm which is denoted as **BOCA**. Finally, we distinguish **BOCA** from **OCA**. Please refer to Sec. D of the online supplementary material for detailed discussions.

7 Extended Model: Enforcing a Power Budget

In additional to accommodating diverse design choices such as buffering facilities in a realistic datacenter, our model and optimization framework can also be extended to incorporate various practical requirements of the datacenter power-performance tradeoff. For instance, as most real-world datacenters are operated within a certain *power budget* [5], it is important for datacenter operators to improve the *performance (dollar) per watt* [5] by achieving a desired performance level with an enforced power budget. Please refer to Sec. E of the online supplementary material for detailed discussions.

8 Performance Evaluation

We conduct comprehensive simulations to evaluate our online control algorithm **OCA** and its extensions from a variety of perspectives, including:

1. the verification of algorithm optimality,
2. the examination of system stability,
3. the adaptivity to bursty request arrivals,
4. the effectiveness of admission control,
5. the effectiveness of power budget enforcement,
6. the sensitivity of key parameters,
7. the exactness of the maximized supreme bound.

Please refer to Sec. G of the online supplementary material for detailed discussions on the experimental configurations and evaluation results available online.

Based on the evaluation under representative datacenter scenarios, we show that: 1) our online control framework can approach a time averaged profit that is arbitrarily close to the optimum, which implies a cost-effective power-performance tradeoff; 2) it can maintain strong system stability, in term of the robustness and adaptivity to time-varying and bursty application requests; 3) it can reflect the potential benefits of practical factors, such as the impact of power efficient hardware and cooling control.

9 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. To show the superiority of our online control framework, an offline control solution is derived in advance to provide a profit benchmark, and also to justify the reliability of the basic model.

In particular, our online control framework is shown to be extensible to explore the various design choices and operational constraints. Specifically, additional buffering facilities are introduced for better system scalability in term of design choices, and a certain power budget is enforced towards better performance (dollar) per watt in term of operational constraints. 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 the optimum, while still maintaining strong system stability, in term of the robustness and adaptivity to time-varying and bursty application requests. As future work, we will investigate more effective cost reduction mechanisms, by dynamically adjusting the number of activated servers in the datacenter.

Acknowledgment

The research was supported in part by a grant from National Basic Research Program (973 program) under Grant 2014CB347800, by a grant from National Natural Science Foundation of China under grant No. 61133006.

References

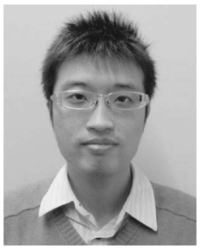
[1] Google Apps. [Online]. Available: http://www.googleapps.com
[2] Salesforce. [Online]. Available: http://www.salesforce.com
[3] Campaign Monitor. [Online]. Available: http://www.campaignmonitor.com/pricing
[4] N.G. Mankiw, *Principles of Economics* 6th ed., Cincinnati, OH, USA: South-Western, 2011.
[5] X. Fan, W. Weberand, and L. Barroso, ''Power Provisioning for a Warehouse-Sized Computer,'' *ACM SIGARCH Comput. Architect. News*, vol. 35, no. 2, pp. 13-23, May 2007.
[6] M.J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. San Rafael, CA, USA: Morgan & Claypool, 2010.
[7] Z. Zhou, F. Liu, Y. Xu, R. Zou, H. Xu, C. Lui, and H. Jin, ''Carbon-Aware Load Balancing for Geo-Distributed Cloud Services,'' in *Proc. IEEE MASCOTS*, 2013, pp. 232-241.
[8] Z. Zhou, F. Liu, H. Jin, B. Li, B.C. Li, and H. Jiang, ''On Arbitrating the Power-Performance Tradeoff in SaaS Clouds,'' in *Proc. IEEE INFOCOM*, 2013, pp. 872-880.
[9] M. Lin, A. Wierman, L. Andrew, and E. Thereska, ''Dynamic Right-Sizing for Power-Proportional Data Centers,'' in *Proc. IEEE INFOCOM*, 2011, pp. 1098-1106.
[10] 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. IEEE INFOCOM*, 2010, pp. 1-9.
[11] Z. Liu, M. Lin, A. Wierman, S.H. Low, and L.L.H. Andrew, ''Greening Geographic Load Balancing,'' in *Proc. ACM SIGMETRICS*, 2011, pp. 233-244.
[12] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, ''Managing Server Energy and Operational Costs in Hosting Centers,'' in *Proc. ACM SIGMETRICS*, 2005, pp. 303-314.
[13] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, ''Statistical Profiling-Based Techniques for Effective Power Provisioning in Data Centers,'' in *Proc. EuroSys*, 2009, pp. 317-330.
[14] R. Urgaonkar, U. Kozat, K. Igarashi, and M. Neely, ''Dynamic Resource Allocation and Power Management in Virtualized Data Centers,'' in *Proc. IEEE NOMS*, 2010, pp. 479-486.
[15] D. Xu and X. Liu, ''Geographic Trough Filling for Internet Datacenters,'' in *Proc. IEEE INFOCOM*, 2012, pp. 2881-2885.
[16] 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. IEEE INFOCOM*, 2012, pp. 1431-1439.
[17] W. Deng, F. Liu, H. Jin, and C. Wu, ''SmartDPSS: Cost-Minimizing Multi-Source Power Supply for Datacenters with Arbitrary Demand,'' in *Proc. IEEE ICDCS*, 2013, pp. 420-429.
[18] A. Greenberg, J. Hamilton, D. Malta, and P. Patel, ''The Cost of a Cloud: Research Problems in Data Center Networks,'' *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68-73, Jan. 2009.
[19] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, ''Power and Performance Management of Virtualized Computing Environments Via Lookahead Control,'' *Cluster Comput.*, vol. 12, no. 1, pp. 1-15, Mar. 2009.
[20] A. Wierman, L.L.H Andrew, and A. Tang, ''Power-Aware Speed Scaling in Processor Sharing Systems,'' in *Proc. IEEE INFOCOM*, 2009, pp. 2007-2015.
[21] 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. NSDI*, 2008, pp. 337-350.
[22] N. Rasmussen, *Electrical Efficiency Modeling for Data Centers*, White Paper No. 113.
[23] The Green Grid. [Online]. Available: http://www.thegreengrid.org

[24] M.J. Neely, ''Delay-Based Network Utility Maximization,'' in *Proc. IEEE INFOCOM*, 2010, pp. 1-9.

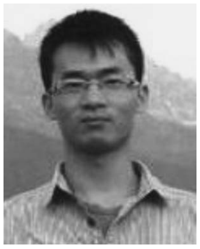
[25] S.T. Theja, R. Srikant, and L. Ying, ''Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters,'' in *Proc. IEEE INFOCOM*, 2012, pp. 702-710.

[26] 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 Distrib. Syst.*, vol. 23, no. 7, pp. 1227-1239, July 2012.

[27] F. Liu, Y. Sun, B. Li, B. Li, and X. Zhang, ''FS2You: Peer-Assisted Semi-Persistent Online Hosting at a Large Scale,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 10, pp. 1442-1457, Oct. 2010.



Fangming Liu received the BEngr degree in 2005 from Department of Computer Science and Technology, Tsinghua University, Beijing, China; and the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology, in 2011. He is an Associate Professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. He was awarded as the CHUTIAN Scholar of Hubei Province, China. He is the Youth Scientist of National 973 Basic Research Program Project on Software-defined Networking (SDN)-based Cloud Datacenter Networks, which is one of the largest SDN projects in China. Since 2012, he has also been invited as a StarTrack Visiting Young Faculty in Microsoft Research Asia (MSRA), Beijing. From 2009 to 2010, he was a visiting scholar at the Department of Electrical and Computer Engineering, University of Toronto, Canada. He was the recipient of two Best Paper Awards from IEEE GLOBECOM 2011 and IEEE CloudCom 2012, respectively. His research interests include cloud computing and datacenter networking, mobile cloud, green computing and communications, software-defined networking and virtualization technology, large-scale Internet content distribution and video streaming systems. He is a member of ACM, as well as a of the China Computer Federation (CCF) Internet Technical Committee. He has been a Guest Editor for IEEE Network Magazine, an Associate Editor for Frontiers of Computer Science, and served as TPC for IEEE INFOCOM 2013-2014 and GLOBECOM 2012-2013. He is a member of the IEEE.



Zhi Zhou is a master student in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interests include cloud computing and green datacenters.

Hai Jin received the PhD degree in a computer engineering from HUST, in 1994. He is a Cheung Kung Scholars Chair Professor of computer science and engineering at the Huazhong University of Science and Technology (HUST), China. He is now Dean of the School of Computer Science and Technology at HUST. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He worked at the University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded the Excellent Youth Award from the National Science Foundation of China in 2001. He is the chief scientist of ChinaGrid, the largest grid computing project in China, and chief scientist of the National 973 Basic Research Program Project of Virtualization Technology of Computing Systems. He is a member of the ACM. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security. He is a Senior Member of the IEEE.

Bo Li received the BEng degree in computer science from Tsinghua University, Beijing, and the PhD degree in electrical and computer engineering from University of Massachusetts at Amherst. He is a Professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. His current research interests include large-scale content distribution in the Internet, datacenter networking, cloud computing, and mobile sensor networks. He made pioneering contributions in the Internet video broadcast with a system, called Coolstreaming (the keyword had over 2,000,000 entries on Google), which was credited as the world first large-scale Peer-to-Peer live video streaming system, which spearheaded a momentum in video broadcast industry, with no fewer than a dozen successful companies adopting the same mesh-based pull streaming technique to deliver live media content to hundreds of millions of users in the world. Dr. Li has been an editor or a guest editor for a dozen of IEEE journals and magazines. He was the Co-TPC Chair for IEEE INFOCOM 2004. He is a Fellow of the IEEE.

Baochun Li received the BEng degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and the MS and PhD degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. He is a Professor with the Department of Electrical and Computer Engineering at the University of Toronto, Canada. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. Dr. Li was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM. He is a Senior Member of the IEEE.

Hongbo Jiang received the BS and MS degrees from Huazhong University of Science and Technology, China, and the PhD degree from Case Western Reserve University in 2008, where he joined the faculty of Huazhong University of Science and Technology as an Associate Professor. His research concerns computer networking, especially algorithms and architectures for high-performance networks and wireless networks. He is a member of the IEEE.