Dynamic Pricing and Profit Maximization for the Cloud with Geo-distributed Data Centers

Jian Zhao*, Hongxing Li*, Chuan Wu*, Zongpeng Li†, Zhizhong Zhang*, Francis C.M. Lau*
* The University of Hong Kong, {jzhao,hxli,cwu,zzzhang,fcmlau}@cs.hku.hk
† University of Calgary, zongpeng@ucalgary.ca

***Abstract*—Cloud providers often choose to operate datacenters over a large geographic span, in order that users may be served by resources in their proximity. Due to time and spatial diversities in utility prices and operational costs, different datacenters typically have disparate charges for the same services. Cloud users are free to choose the datacenters to run their jobs, based on a joint consideration of monetary charges and quality of service. A fundamental problem with significant economic implications is how the cloud should price its datacenter resources at different locations, such that its overall profit is maximized. The challenge escalates when dynamic resource pricing is allowed and long-term profit maximization is pursued. We design an efficient online algorithm for dynamic pricing of VM resources across datacenters in a geo-distributed cloud, together with job scheduling and server provisioning in each datacenter, to maximize the profit of the cloud provider over a long run. Theoretical analysis shows that our algorithm can schedule jobs within their respective deadlines, while achieving a time-average overall profit closely approaching the offline maximum, which is computed by assuming that perfect information on future job arrivals are freely available. Empirical studies further verify the efficacy of our online profit maximizing algorithm.**

I. Introduction

Recent years have witnessed the proliferation of cloud computing platforms, services and applications [1] [2] [3]. To better serve the computing demands from users in different geographical regions, it is common for a cloud provider to host multiple datacenters in a number of selected locations. Given the different operational costs across service regions, resources (*e.g.*, virtual machines) are naturally priced differently across data centres [2]. Users of the cloud system can strategically decide the datacenters to run their jobs in, based on the resource prices and the desired quality of service (*e.g.*, communication delays between the user's location and the datacenters).

How the cloud provider should price its resources in datacenters distributed across different locations such that the overall profit is maximized is a problem of fundamental importance. As compared to fixed prices (*e.g.*, Amazon on-demand instances), dynamic pricing that reflects the realtime supply-demand relationship (*e.g.*, Amazon spot instances) represents a more promising charge strategy that can better exploit user payment potentials and thus larger profit gains at the cloud provider. Under the objective to maximize the overall profit in the cloud, it is however non-trivial to decide a dynamic price for VMs in each datacenter at a given time, which is intimately connected to decisions on server right-sizing (turning servers on/off) and job scheduling among different datacenters.

The challenge escalates when we want to pursue time-averaged profit maximization over a long run of the system, with dynamically arriving user jobs with heterogeneous execution times, and based on online decision making. A number of intriguing questions are involved: What is the strategy for each user to select the cloud datacenter for its job execution, in order to maximize its own utility? Given the user strategy, how should the cloud dynamically price its VMs and decide the number of active servers in each datacenter at any time such that the jobs are maximally served and its profit is maximized over time?

In this work, we answer these questions by jointly modelling job scheduling, VM pricing and server provisioning decisions as an integrated stochastic optimization framework based on Lyapunov optimization theory [4]. An efficient online algorithm is designed to guide the operational decisions of the cloud provider to pursue maximal time-averaged profit over the long run. Based on rigorous theoretical analysis, we demonstrate that the algorithm has the following desired properties: (1) The algorithm guarantees no job dropping under two mild conditions as presented in Sec. IV-B, while all the accepted jobs can be completed within their respective completion deadlines; (2) the algorithm achieves a time-averaged overall profit for the cloud provider, which can approach the offline maximum arbitrarily closely. Note that the latter is computed under the strong assumption that complete information of all job arrivals, including those in the future, are magically available.

To our knowledge, this work is among the first to design efficient strategies for joint dynamic pricing, job scheduling, and resource provisioning in the cloud computing literature, and among the first to handle jobs with variable lengths under the Lyapunov optimization framework. In particular, we consider a cloud with various VM configurations, whose operational costs vary in both the temporal and spatial domains. We address dynamic arrivals of jobs into the cloud, with various requirements on types and lengths of occupation of different VMs, as well as different job completion deadlines. A salient contribution in our Lyapunov optimization approach is that, we allow the execution time of each job to be longer than the interval of online decision making, such that decisions

The research was supported in part by grants from Hong Kong RGC under the contracts HKU 717812 and HKU 718513, and in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).
978-1-4799-3360-0/14/$31.00 ©2014 IEEE.

in consecutive decision intervals are strongly correlated, beyond what the standard Lyapunov optimization framework can handle. Employing a new design of the dynamic algorithm in two time scales, we can still ensure its close-to-optimal performance, based on rigorous theoretical analysis. It is noteworthy that our algorithm has fundamental difference from the ingenious work [5] with two-time-scale scheduling, in that we need no expectation into the future to be extracted from historical data. Instead, our framework makes dynamic decisions just based on the current status of the system.

The remainder of the paper is organized as follows. Sec. II presents the system model. The online algorithm is designed in Sec. III. The performance of the algorithm is analyzed in Sec. IV. Sec. V presents the simulation results. We review related literature in Sec. VI and conclude the paper in Sec. VII.

II. Model & Notation

A. The Cloud System Model

Consider a cloud provider with a set $\mathcal{D}$ (with size $D = |\mathcal{D}|$) of geo-distributed datacenters, indexed by d where $1 \leq d \leq D$. Each datacenter d has N^d homogeneous servers. The system operates in a time-slotted fashion, for $t = 0, 1, \ldots, T$. A set $\mathcal{H}$ (with size $H = |\mathcal{H}|$) of distinct types of virtual machine (VM) instances are provided in the cloud, each with a specific set of configurations of CPU, memory, and storage, characterizing heterogenous VM instance provisioning in the real world such as in Amazon EC2 [2]. Each server in a datacenter hosts VMs of the same type in a time slot, which can change across different time slots [6]. Let n_h^d denote the maximum number of type-h VMs that a server of datacenter d can simultaneously host.

Datacenters receive VM requests from customers in the form of *jobs*. Each job $r \in \mathcal{R}$ is a pair (h_r, w_r), where $h_r \in \mathcal{H}$ is the type of VM requested; $w_r \in [w^{min}, w^{max}]$ is the number of time slots requested and is referred to the *workload* of the job. The set of possible job types is $\mathcal{R}$ with $R = |\mathcal{R}|$. As a Service Level Agreement (SLA), the cloud provider guarantees that the maximum job scheduling latency is bounded by l, *i.e.*, the delay from the time the job is submitted to a datacenter to the time it is allocated a VM, will not exceed l.

Cloud customers reside in a set of geo-distributed zones $\mathcal{J}$ with size $J = |\mathcal{J}|$. The utility obtained by customers in zone j when $a_r^{j,d}$ type-r jobs are served by datacenter d is $U_r^{j,d}(a_r^{j,d}), j \in \mathcal{J}, d \in \mathcal{D}, r \in \mathcal{R}$, which is a differentiable, concave utility function.

B. The Cloud Provider's Solution Space

We aim to design dynamic, optimal algorithms for the cloud provider to strategically make the following operational decisions in each datacenter at each time slot: (i) *Front-end job pricing*: What prices should be charged to each type of jobs with a specific workload? (ii) *Job scheduling*: How many jobs of each type should be scheduled for execution in each datacenter? (iii) *Server/VM provisioning*: How many servers should be turned on, and what type of VMs should each active server provision? The goal is overall profit optimization from all datacenters over the long run.

Job Pricing. Let $p_r^d(t) \in [0, p_r^{d,max}]$ be the price charged to a type-r job at datacenter d at time t, upper-bounded by $p_r^{d,max}$, which will be related to customers' maximum value for a type-r job at datacenter d.

Given the job price $p_r^d(t)$, customers in zone j will request $a_r^{j,d}(t)$ type-r jobs from datacenter d, for maximizing their surplus (total utility minus total charges) under the charging prices $p_r^d(t)$:

$$a_r^{j,d}(t) = \text{argmax}_{a_r^{j,d}} \sum_{d \in \mathcal{D}} [U_r^{j,d}(a_r^{j,d}(t)) - p_r^d(t) \cdot a_r^{j,d}(t)]. \quad (1)$$

The total number of type-r jobs datacenter d receives is $\sum_{j \in \mathcal{J}} a_r^{j,d}(t)$, with total workload $w_r \sum_{j \in \mathcal{J}} a_r^{j,d}(t)$. The job arrival rate $a_r^{j,d}(t)$ is upper-bounded by a_r^{max}.

Job Scheduling. Each datacenter maintains R queues of unscheduled workload, Q_r^d, each corresponding to a distinct job type $r, \forall r \in \mathcal{R}$. Upon arrival of a type-r job at datacenter d, w_r units of workload are appended to Q_r^d; when a job is scheduled for execution, a unit workload departs from Q_r^d in that time slot. Let $\mu_r^d(t)$ denote the number of type-r jobs scheduled to run in datacenter d in time slot t. Once scheduled, the job will occupy a VM of type h_r for w_r *consecutive* time slot(s). Let $\mu_r^d(t^-)$ denote the number of type-r jobs scheduled before t, which are still running on datacenter d in t. We model the potential dropping of a job when its SLA requirement l (max scheduling delay) cannot be met. Let $G_r^d(t)$ denote the number of unscheduled jobs of type r in datacenter d, which are dropped in t, $0 \leq G_r^d(t) \leq G_r^{max}$, where G_r^{max} is the maximum number of jobs allowed to drop in one time slot, In practice, a cloud may never drop a user's job. The "drop" in our model can be understood as follows: The cloud maintains a set of regular resources ($\sum_{d \in \mathcal{D}} N^d$ VMs) while keeping a set of backup resources, whose provisioning can be expensive. When a job is "dropped" due to not being scheduled using the regular resources when its response delay is due, the cloud uses its expensive backup resources to serve the job, subject to a cost η_r ("the job drop penalty") to serve one type-r job. The SLA requirement can be formulated as follows:

Each type-r job is either scheduled or dropped (subject to a penalty) before its maximum scheduling delay l. (2)

Let $Q_r^d(t)$ be the total unprocessed workload of type-r jobs in datacenter d at t. It is updated over time as follows:

$$Q_r^d(t+1) = \max\{Q_r^d(t) - \mu_r^d(t) - \mu_r^d(t^-) - w_r G_r^d(t), 0\} + w_r \sum_{j \in \mathcal{J}} a_r^{j,d}(t). \quad (3)$$

Here, $\mu_r^d(t)$ is the total number of type-r jobs newly scheduled to run on VMs of type h_r at the beginning of time slot t; for each of these jobs, one unit of workload is reduced from $Q_r^d(t)$ after the job has been running for the time slot. $\mu_r^d(t^-)$ is the total number of left-over type-r jobs still running in datacenter d; for each of these jobs, one unit of workload is also reduced from $Q_r^d(t)$ after it has been running for that time slot. The

TABLE I
IMPORTANT NOTATIONS

$\mathcal{D}$	set of datacenters	$\mathcal{H}$	set of VM types
$\mathcal{R}$	set of all job types	N^d	# of servers at $d \in \mathcal{D}$
h_r	VM type for type-r jobs	w_r	size of a type-r job
l	Max job scheduling delay	$\mathcal{J}$	set of customer zones
n_h^d	Max # of type h VMs a server in datacenter d can host		
$p_r^d(t)$	price charged to a type-r job at t in d		
$a_r^{j,d}(t)$	# of admitted type r jobs from zone j at t in d		
$\mu_r^d(t)$	# of new type r jobs executed at datacenter d at time slot t		
$\mu_r^d(t^-)$	# of type r jobs at datacenter d left over from earlier		
$G_r^d(t)$	# of dropped jobs in Q_r^d at time slot t		
$N_h^d(t)$	# of active servers for type-h VMs in d at time t		
$c^d(t)$	unit power cost for running one server at datacenter d in t		
η_r	penalty for dropping a type-r job		
$P(t)$	cloud's profit at time t		
Q_r^d	queue of unscheduled type r workload at datacenter d		
Z_r^d	virtual queue for bounding queueing delay in Q_r^d		
ϵ_r	preset constant for controlling queueing delay in Q_r^d		
$p_r^{d,max}$	max price for type-r jobs in datacenter d		
a_r^{max}	max # of type-r jobs arriving in one slot		
$\mu_r^{d,max}$	max # of type-r jobs allowed to be scheduled in one slot		
G_r^{max}	max # of type-r jobs allowed to be dropped in one slot		

drop of $G_r^d(t)$ unscheduled jobs brings a reduction of $w_r G_r^d(t)$ units of workload from $Q_r^d(t)$.

$\mu_r^d(t^-), \forall r \in \mathcal{R}, \forall d \in \mathcal{D}$, are known in t, based on hitherto scheduling decisions and the workload size of each scheduled job. $\mu_r^d(t)$, $G_r^d(t)$, and $p_r^d(t)$ (which decides $a_r^{j,d}(t)$), $\forall r \in \mathcal{R}, \forall d \in \mathcal{D}$, are decision variables our algorithm judiciously computes in each time slot, not only to maximize the profit, but also to guarantee that the scheduling delay of each job of type r is within its deadline l. In particular, if the maximum queueing delay of each unit of workload in Q_r^d can be bounded by l, then the maximum scheduling delay for each incoming type-r job is also bounded within l.

Server/VM Provisioning. Let $N_h^d(t)$ denote the number of active servers in datacenter d configured to provision VMs of type h in time slot t. These servers can be used to serve jobs of type-r, where $h_r = h$. We have

$$N_h^d(t) \geq \sum_{r:h_r=h} (\mu_r^d(t) + \mu_r^d(t^-))/n_h^d.$$

We are interested in the minimum number of servers required to meet the VM demands, assuming an efficient intra-datacenter VM migration algorithm [7]) that helps move running VMs from one server to another, for reducing the number of active servers.

C. The Profit Maximization Problem

The cloud provider's net profit is the difference between the revenue and the costs. The total revenue by taking in jobs of different types in t is $\sum_{d\in\mathcal{D}} \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{J}} a_r^{j,d}(t) p_r^d(t)$. We consider power consumption in operating servers as the major component of operational costs in a datacenter [8]. Let $c^d(t)$ be the unit cost of operating one server in datacenter d in time slot t, which is naturally time varying and location dependent. The total cost in the cloud in t is $\sum_{d\in\mathcal{D}} c^d(t) \sum_{h\in\mathcal{H}} N_h^d(t)$. A penalty of η_r is enforced for dropping a job of type-r, with $\eta_r \geq p_r^{d,max}, \forall r \in \mathcal{R}, \forall d \in \mathcal{D}$. Hence, expenditure on penalty occurs in time slot t if there are dropped jobs, with the total amount of $\sum_{d\in\mathcal{D}} \sum_{r\in\mathcal{R}} \eta_r G_r^d(t)$.

The net profit of the cloud provider in time slot t is:

$$P(t) = \sum_{d\in\mathcal{D}} \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{J}} a_r^{j,d}(t) p_r^d(t) - \sum_{d\in\mathcal{D}} c^d(t) \sum_{h\in\mathcal{H}} N_h^d(t) - \sum_{d\in\mathcal{D}} \sum_{r\in\mathcal{R}} \eta_r G_r^d(t).$$

The time-averaged expected profit of the cloud is:

$$\overline{P(t)} \triangleq \lim_{T\to\infty} \sup \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left[P(t)\right].$$

The profit maximization pursued by the cloud is therefore:

$$\max: \quad \overline{P(t)} \tag{4}$$

$$\text{s.t.}: \quad 0 \leq p_r^d(t) \leq p_r^{d,max}, \forall r \in \mathcal{R}, d \in \mathcal{D}, t \in [1,T]; \tag{5}$$

$$0 \leq G_r^d(t) \leq G_r^{max}, \forall r \in \mathcal{R}, d \in \mathcal{D}, t \in [1,T]; \tag{6}$$

$$\sum_{h\in\mathcal{H}} N_h^d(t) \leq N^d, \forall d \in \mathcal{D}, t \in [1,T]; \tag{7}$$

$$\sum_{r:h_r=h} (\mu_r^d(t) + \mu_r^d(t^-))/n_h^d \leq N_h^d(t), \quad \forall h \in \mathcal{H}, \forall d \in \mathcal{D}, t \in [1,T]; \tag{8}$$

$$\mu_r^d(t) \geq 0, \forall r \in \mathcal{R}, \forall d \in \mathcal{D}, t \in [1,T]; \tag{9}$$

$$N_h^d(t) \in \mathbb{Z}^+ \cup 0, \forall h \in \mathcal{H}, \forall d \in \mathcal{D}, t \in [1,T]; \tag{10}$$

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{w_r \sum_{j\in\mathcal{J}} a_r^{j,d}[p_r^d(t)]\} < \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\mu_r^d(t) + \mu_r^d(t^-) + w_r G_r^d(t)\}, \quad \forall r \in \mathcal{R}, d \in \mathcal{D}; \tag{11}$$

Constraint (2).

This optimization problem is for the cloud provider to choose an appropriate price for each type of jobs at each datacenter ($p_r^d(t)$), the best number of servers to provision each type of VMs in each datacenter ($N_h^d(t)$), the optimal numbers of jobs of each type to schedule and to drop ($\mu_r^d(t)$ and $G_r^d(t)$), in each t at each datacenter, to maximize its time-averaged profit. Constraint (7) ensures that the total number of active servers in each datacenter is bounded by the number of on-premise servers. Constraint (8) specifies that the total number of newly scheduled and left-over jobs in a datacenter, each requiring a type-h VM, does not exceed the number of type-h VMs provisioned. Constraint (11) guarantees the stability of job queue Q_r^d, by ensuring that the average arrival rate is no higher than the average departure rate [4].

Table I summarizes the notations for ease of reference.

III. THE DYNAMIC PROFIT MAXIMIZATION ALGORITHM

We now design an online algorithm to solve the profit maximization problem in (4).

A. Addressing SLA Requirements

To guarantee that the worst-case queueing delay in each workload queue $Q_r^d, \forall r \in \mathcal{R}, d \in \mathcal{D}$, is bounded by l, we associate each workload queue Q_r^d with a virtual queue $Z_r^d(t)$, based on the ϵ-persistent service queue technique for delay

bounding [9]. When the queue backlogs of Q_r^d and $Z_r^d, \forall r \in \mathcal{R}, d \in \mathcal{D}$, are bounded, the jobs' queueing delays are bounded.

The backlog of the virtual queue is initially $Z_r^d(0) = 0$, and then updated as follows:

$$Z_r^d(t+1) = \max[Z_r^d(t) + 1_{Q_r^d(t)>0}(\epsilon_r - \mu_r^d(t) - \mu_r^d(t^-)) - w_r G_r^d(t) - 1_{Q_r^d(t)=0}\mu_r^{d,max}, 0]. \quad (12)$$

Here the indicator function $1_{Q_r^d(t)>0}$ is 1 when $Q_r^d(t) > 0$, and 0 otherwise. Similarly, $1_{Q_r^d(t)=0}$ is 1 when $Q_r^d(t) = 0$, and 0 otherwise. ϵ_r is a pre-defined constant that is no larger than $w_r a_r^{max}$ and can be gauged to control the queueing delay bound. $\mu_r^{d,max}$ is the maximum number of type-r jobs that can run simultaneously in datacenter d, with $\mu_r^d(t) + \mu_r^d(t^-) \leq \mu_r^{d,max}$.

By designing a dynamic algorithm that guarantees the lengths of queues Z_r^d and Q_r^d are bounded over time, we are able to guarantee the queueing delay of workload queue Q_r^d is bounded by l. The rationale can be intuitively explained as follows: Let $Z_r^{d,max}$, $Q_r^{d,max}$ be the bound of queues Z_r^d, Q_r^d, respectively. Consider workloads arriving at any time slot t. In the subsequent l time slots after t, if Q_r^d decreases to 0, the workloads are served within l time slots; otherwise, Z_r^d has a constant arrival rate ϵ_r, and the same departure rate $\mu_r^d(t) + \mu_r^d(t^-) + w_r G_r^d(t)$ as that in the workload queue Q_r^d. For the interval of l time slots following t, the total arrivals into queue Z_r^d minus the total departures is smaller than or equal to the queue length bound $Z_r^{d,max}$, *i.e.*, $\epsilon_r l - \sum_{\tau=t+1}^{t+l}[\mu_r^d(t) + \mu_r^d(t) + w_r G_r^d(t)] \leq Z_r^{d,max}$. At any time slot t, as the positions of workloads arriving at time slot t in queue Q_r^d would not exceed the bound $Q_r^{d,max}$, when the total departure number during the l time slots following t is at least $Q_r^{d,max}$, *i.e.*, $\sum_{t+1}^{t+l}[\mu_r^d(t)+\mu_r^d(t^-)+w_r G_r^d(t)] \geq Q_r^{d,max}$, jobs arriving at t will be served within these l time slots. Hence when $\epsilon_r \cdot l - Z_r^{d,max} \geq Q_r^{d,max}$ (*i.e.*, $l = \lceil (Z_r^{d,max} + Q_r^{d,max})/\epsilon_r \rceil$), which guarantees $\sum_{t+1}^{t+l}[\mu_r^d(t) + \mu_r^d(t^-) + w_r G_r^d(t)] \geq \epsilon_r \cdot l - Z_r^{d,max} \geq Q_r^{d,max}$, all jobs are scheduled with delays of at most l time slots.

B. Dynamic Algorithm Design

In an online algorithm, we compute instantaneous values of the decision variables, while seeking to solve the optimization in (4) that involves time-averaged variable values. To satisfy constraint (11), we need to guarantee that each workload queue Q_r^d is stable over time [10]. To maximize the time-averaged objective function based on decisions in each time slot, we resort to the drift-plus-penalty framework in Lyapunov optimization [4], a classic technique for translating a long-term time-average optimization problem into a series of similar one-shot optimization problems. In particular, let $\mathbf{\Theta}(t) = [\mathbf{Q}(t), \mathbf{Z}(t)]$ be the vector of all queues in the system, where $\mathbf{Q}(t)$ and $\mathbf{Z}(t)$ are the vectors of workload queues $Q_r^d(t)$ and virtual queues $Z_r^d(t)$, respectively, $\forall r \in \mathcal{R}, d \in \mathcal{D}$. We define a Lyapunov function as follows:

$$L(\mathbf{\Theta}(t)) = \frac{1}{2}[\sum_{r\in\mathcal{R}}\sum_{d\in\mathcal{D}}(Q_r^d(t)^2 + Z_r^d(t)^2)].$$

The one-slot conditional Lyapunov drift is

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

Following the *drift-plus-penalty* framework in Lyapunov optimization [4], we minimize an upper bound for the following expression in each time slot t, with the observation of the queue states ($[\mathbf{Q}(t), \mathbf{Z}(t)]$), the number of jobs still running in datacenters ($\mu_r^d(t^-), \forall r \in \mathcal{R}, d \in \mathcal{R}$), and costs of running servers in the datacenters ($c^d(t), \forall d \in \mathcal{D}$), such that a lower bound for $\overline{P(t)}$ is maximized (see Chapter 5 in [4]):

$$\Delta(\mathbf{\Theta}(t)) - VP(t).$$

Here, V is a non-negative parameter chosen by the cloud to control the tradeoff between the profit and the SLA guarantee. A larger V leads to a higher time-averaged profit but a higher queueing delay at the same time.

Squaring the queueing laws (3) and (12), we can derive the following inequality (detailed steps in technical report [11]):

$$\begin{aligned}
&\Delta(\mathbf{\Theta}(t)) - VP(t) \\
&\leq B + \sum_{d\in\mathcal{D}}\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{J}} a_r^{j,d}(t)[w_r Q_r^d(t) - V p_r^d(t)] \\
&+ V\sum_{d\in\mathcal{D}} c^d(t)\sum_{h\in\mathcal{H}} N_h^d(t) - \sum_{\mathcal{D},\mathcal{R}}[\mu_r^d(t) + \mu_r^d(t^-)][Q_r^d(t) + Z_r^d(t)] \\
&+ \sum_{d\in\mathcal{D}}\sum_{r\in\mathcal{R}}[V\eta_r - w_r Q_r^d(t) - w_r Z_r^d(t)]G_r^d(t) \qquad (13)
\end{aligned}$$

where

$$B = \frac{1}{2}\sum_{r\in\mathcal{R}}\sum_{d\in\mathcal{D}}[(w_r a_r^{max})^2 + 2(\mu_r^{d,max} + w_r G_r^{max})^2 + (\epsilon_r)^2]$$

is a constant.

Our algorithm seeks to minimize the RHS of inequality (13), to minimize the upper bound for $\Delta(\mathbf{\Theta}(t)) - VP(t)$, and thus to maximize the lower bound of $\overline{P(t)}$. The bound of workload queues Q_r^d's and virtual queues Z_r^d's can also be guaranteed in this process (Sec. IV), such that constraint (11) and the SLA requirements of each type of jobs are satisfied.

In particular, in each time slot t, the algorithm observes the queues $Q_r^d(t)$ and $Z_r^d(t)$, the current unit costs of running servers in datacenters $c^d(t)$, the number of active type-r jobs at datacenter d $\mu_r^d(t^-)$, and decides the optimal values of $p_r^d(t)$,$\mu_r^d(t)$, $G_r^d(t)$ and $N_h^d(t)$, by solving the following one-shot optimization problem:

$$\begin{aligned}
\text{min:} \quad & \text{RHS of (13)} \qquad (14)\\
\text{s.t.:} \quad & \text{Constraints (5)(6)(7)(8)(9)(10).}
\end{aligned}$$

A difference between this work and previous work using Lyapunov optimization is that the previous work usually assume each job can be completed in one time slot, while we model the more general scenario in which a job may take more than one time slot to finish (and they can not be prematurely terminated once scheduled to run on the required VMs). Previously scheduled jobs may still be running in datacenters and occupying VMs. This constrains the control decisions in the current time slot. We present the detailed control decisions in the following.

A careful investigation of the RHS of (13) reveals that optimization (14) can be equivalently decoupled into three types of independent optimization (excluding constant terms), dealing with (a) front-end job pricing, (b) job dropping, and (c) job scheduling and server/VM provisioning, respectively.

(a) Front-end Pricing: It decides the price charged to a type-r job in each datacenter. To minimize the RHS of (13), the part related to prices is as follows:

$$\min \sum_{d\in\mathcal{D}}\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{J}} a_r^{j,d}(t)[w_r Q_r^d(t) - V p_r^d(t)]. \tag{15}$$

Recall that the number of type-r jobs users in a zone j submit to each datacenter d, $a_r^{j,d}(t)$, is decided by the prices $p_r^d(t)$'s in different datacenters, in order to maximize their surplus, as given in (1). The *marginal surplus* is derived as $(U_r^{j,d})'(a_r^{j,d}(t)) - p_r^d(t)$. As $U_r^{j,d}(\cdot)$ is a differentiable concave function, $(U_r^{j,d})'(a_r^{j,d}(t))$ is non-increasing. When $p_r^d(t) > (U_r^{j,d})'(0)$, customers in zone j will not run their type-r jobs in datacenter d, i.e., $a_r^{j,d} = 0$. When $p_r^d(t) \le (U_r^{j,d})'(0)$, the number of type-$r$ jobs that customers in zone j will send to datacenter d is computed by setting the marginal surplus to zero, as $a_r^{j,d}(t) = (U_r^{j,d})'^{-1}(p_r^d(t))$, where $(U_r^{j,d})'^{-1}(\cdot)$ denotes the inverse function of $(U_r^{j,d})'(\cdot)$.

We replace $a_r^{j,d}(t)$ in (15) by $\max\{0, (U_r^{j,d})'^{-1}(p_r^d(t))\}$, and optimization (15) is now on the price variables $p_r^d(t)$'s only.

Let $\hat{p}_r^{j,d}$ denote $(U_r^{j,d})'(0)$, *i.e.*, the price value under which users in zone j will not request type-r jobs from datacenter d. In general, the J price values $\hat{p}_r^{j,d}, 1 \le j \le J$ can be sequenced from the lowest one to the highest one, $\hat{p}_r^{j_1,d} \le \hat{p}_r^{j_2,d} \le \ldots \le \hat{p}_r^{j_J,d}$. For prices among region $[\hat{p}_r^{j_m,d}, \hat{p}_r^{j_{m+1},d}], 1 \le m \le J-1$, users in zones from j_{m+1} to zone j_J will request VMs from datacenter d, and the corresponding optimization problem is as follows:

$$\begin{aligned} \min \quad & \sum_{i=m+1}^{J} (U_r^{j_i,d})'^{-1}(p_r^d(t))[w_r Q_r^d(t) - V p_r^d(t)] \\ \text{s.t.} \quad & \hat{p}_r^{j_m,d} \le p_r^d(t) \le \hat{p}_r^{j_{m+1},d}. \end{aligned} \tag{16}$$

For each region $[\hat{p}_r^{j_m,d}, \hat{p}_r^{j_{m+1},d}], 1 \le m \le J-1$, there is an optimization problem. There are in total $J-1$ optimization problems. Among different price regions, the objective function changes due to the reason that users in some zones may not use the service. The optimal pricing strategy in the resulted $J-1$ solutions is the one achieving the minimum objective function value.

(b) Job Dropping: The number of jobs dropped from queue Q_r^d in t, $G_r^d(t), \forall r \in \mathcal{R}$, is derived by solving the following minimization problem:

$$\begin{aligned} \text{min:} \quad & [V\eta_r - w_r Q_r^d(t) - w_r Z_r^d(t)]G_r^d(t) \\ \text{s.t.:} \quad & \text{Constraint (6).} \end{aligned} \tag{17}$$

The optimal solution to the above LP is:

$$G_r^d(t) = \begin{cases} G_r^{max}, & \text{if } Q_r^d(t) + Z_r^d(t) > \frac{V\eta_r}{w_r}; \\ 0, & \text{if } Q_r^d(t) + Z_r^d(t) \le \frac{V\eta_r}{w_r}. \end{cases} \tag{18}$$

The above strategy indicates that a type-r job is less likely to be dropped in t when the penalty of dropping a type-r job, η_r, is large, and jobs requiring smaller running times, w_r, are less likely to be dropped too.

In **Theorem 2** to be proved in Sec. IV, we will show that our scheduling algorithm guarantees zero job dropping, *i.e.*, all jobs admitted into the cloud are successfully processed in time, under two conditions: (1) At any datacenter, the accumulated workload of any type of jobs since the last time slot when workload from the respective queue is scheduled, can all be dispatched to run on servers the next time when the queue is being scheduled; (2) the drop penalty is high enough to make the cloud more willing to turn on servers than to drop jobs, even though the power cost reaches the maximum value.

(c) Job Scheduling and Server/VM Provisioning: Decisions on $\mu_r^d(t)$ and $N_h^d(t)$ in datacenter d are made by solving the following minimization problem:

$$\begin{aligned} \text{min:} \quad & V\sum_{h\in\mathcal{H}} c^d(t) N_h^d(t) - \sum_{r\in\mathcal{R}} [Q_r^d(t) + Z_r^d(t)]\mu_r^d(t) \\ \text{s.t.:} \quad & \text{Constraints (7)(8)(9)(10).} \end{aligned} \tag{19}$$

(19) is a joint job scheduling and server/VM provisioning problem. It can be solved by first converting to a pure server/VM provisioning problem and then deciding job scheduling based on the server/VM provisioning decisions.

Jobs of different types r scheduled to datacenter d, where $h_r = h$, compete for type-h VMs provisioned in the datacenter, as given in constraint (8). Suppose the number of servers configured to provision type-h VMs in datacenter d, $N_h^d(t)$, is known. To minimize (19), we should maximally schedule jobs of type r_h^*, whose observed value of $Q_r^d(t) + Z_r^d(t)$ is the largest among all types of jobs requiring type-h VMs, onto the provisioned type-h VMs, *i.e.*,

$$r_h^* = \text{argmax}_{r:h_r=h}[Q_r^d(t) + Z_r^d(t)], \tag{20}$$

where ties are broken randomly. The number of type-r_h^* jobs we can schedule in t is decided by constraint (8), at

$$\mu_{r_h^*}^d(t) = n_h^d N_h^d(t) - \sum_{r:h_r=h} \mu_r^d(t^-), \forall h \in \mathcal{H}. \tag{21}$$

That is, except VMs occupied by left-over jobs, all other type-h VMs should be used to serve type-r_h^* jobs, and no other types of jobs are scheduled, *i.e.*,

$$\mu_r^d(t) = 0, \forall r \ne r_h^*, \forall h \in \mathcal{H}. \tag{22}$$

Hence, the second part of (19) can be expressed using variables $N_h^d(t)$'s:

$$\sum_{r\in\mathcal{R}} [Q_r^d(t) + Z_r^d(t)]\mu_r^d(t) = \sum_{h\in\mathcal{H}} [Q_{r_h^*}^d(t) + Z_{r_h^*}^d(t)]\mu_{r_h^*}^d(t) =$$
$$\sum_{h\in\mathcal{H}} [Q_{r_h^*}^d(t) + Z_{r_h^*}^d(t)]n_h^d N_h^d(t) - \sum_{h\in\mathcal{H}} [Q_{r_h^*}^d(t) + Z_{r_h^*}^d(t)] \sum_{r:h_r=h} \mu_r^d(t^-).$$

Removing the constant terms, (19) can be converted into the following equivalent server/VM provisioning problem:

$$\begin{aligned} \text{min:} \quad & V\sum_{h\in\mathcal{H}} c^d(t) N_h^d(t) - \sum_{h\in\mathcal{H}} [Q_{r_h^*}^d(t) + Z_{r_h^*}^d(t)] n_h^d N_h^d(t) \\ \text{s.t.:} \quad & N_h^d(t) \ge \sum_{r:h_r=h} \mu_r^d(t^-)/n_h^d, \forall h \in \mathcal{H}; \\ & \text{Constraints (7), (10).} \end{aligned} \tag{23}$$

The objective function of (23) is equivalent to $\sum_{h\in\mathcal{H}} N_h^d(t)\left[Vc^d(t) - [Q^d_{r_h^*}(t) + Z^d_{r_h^*}(t)]n_h^d\right]$ and is linear in $N_h^d(t)$. For an efficient solution to (23), we can compute the VM type h_d^* as

$$h_d^* = \text{argmax}_{h\in\mathcal{H}}[Q^d_{r_h^*}(t) + Z^d_{r_h^*}(t)]n_h^d, \tag{24}$$

where ties are broken randomly. There are two cases:

(i) If $Vc^d(t) \geq [Q^d_{r_h^*}(t) + Z^d_{r_h^*}(t)]n_h^d|_{h=h_d^*}$, the objective function is always non-negative, and $N_h^d(t)$'s should be as small as possible. Hence, only the minimum number of servers running left-over jobs are kept on, while the other servers should be turned down in this datacenter d, *i.e.*,

$$N_h^d(t) = \lceil \sum_{r:h_r=h} \mu_r^d(t^-)/n_h^d \rceil, \forall h \in \mathcal{H}. \tag{25}$$

(ii) If $Vc^d(t) < [Q^d_{r_h^*}(t) + Z^d_{r_h^*}(t)]n_h^d|_{h=h_d^*}$, all servers in datacenter d should be activated, and except those occupied by left-over jobs, they should provision type h_d^* VMs, *i.e.*,

$$N_h^d(t) = \lceil \sum_{r:h_r=h} \mu_r^d(t^-)/n_h^d \rceil, \forall h \in \mathcal{H}, h \neq h_d^*, \tag{26}$$

$$N_{h_d^*}^d(t) = N^d - \sum_{h\neq h_d^*} N_h^d(t). \tag{27}$$

After $N_h^d(t)$'s are decided, the job scheduling decisions can be made based on Eqn. (20)(21)(22). In particular, in case (i), no new jobs are scheduled onto datacenter d in t; in case (ii), all newly provisioned type-h_d^* VMs serve jobs of type $r^*_{h_d^*}$.

Dealing with Varying Job Workloads. In the standard Lyapunov optimization framework, minimizing the 1-slot drift-plus-penalty in each time slot can be proved to optimize a time-averaged utility over the long run, with the critical assumption that all jobs have the equal fixed length, equivalent to one time slot. Decisions made in one time slot do not influence resources to be allocated in the subsequent slots [4]. Our system model is more general: a type-r job scheduled in t will occupy a VM for w_r time slots, directly affecting job scheduling and resource provisioning choices in later times. We novelly make the following design in our non-preemptive algorithm, with algorithmic optimality proved in Sec. IV.

We group Γ time slots into a *time frame*, where Γ is larger than w^{max}. The above job scheduling and server/VM provision algorithm varies slightly depending on which time slot it is running in (front-end pricing and job dropping algorithms remain intact): in a time slot $t \in [n\Gamma, (n+1)\Gamma - w^{max}]$ in the beginning part of a time frame, the above job scheduling and server/VM provisioning algorithm remains intact; in a time slot $t \in [(n+1)\Gamma - w^{max} + 1, (n+1)\Gamma - 1]$ towards the end of a time frame, the algorithm differs in that only type-r jobs with $w_r \leq (n+1)\Gamma - t$ (*i.e.*, which can be finished in this time frame), are considered in the choice of r_h^*:

$$r_h^* = \text{argmax}_{r:h_r=h, w_r\leq(n+1)\Gamma-t}[Q_r^d(t) + Z_r^d(t)], \tag{28}$$

and h_d^* is calculated correspondingly by (24).

The complete dynamic algorithm, carried out in each time slot by the cloud, is summarized in Algorithm 1.

Algorithm 1 Dynamic algorithm in time slot t

Input: $Q_r^d(t)$, $Z_r^d(t)$, $\mu_r^d(t^-)$, $c^d(t)$, N^d, n_h^d, V, ϵ_r, p_r^{max},$\mu_r^{d,max}$, G_r^{max}, Γ ($\forall r \in \mathcal{R}$, $\forall d \in \mathcal{D}$, $\forall h \in \mathcal{H}$).
Output: $p_r^d(t)$, $N_h^d(t)$, $\mu_r^d(t)$, $G_r(t)$ ($\forall r \in \mathcal{R}$, $\forall d \in \mathcal{D}$, $\forall h \in \mathcal{H}$)

for Each datacenter $d \in \mathcal{D}$ **do**
  Choose the price by solving the $J-1$ optimization problems in (16).
  if $(t \bmod \Gamma) \in [0, \Gamma - w^{max}]$ **then**
    for Each VM type $h \in \mathcal{H}$ **do**
      Determine the type of jobs type-h VMs should serve, r_h^*, using equation (20).
    end for
  else if $(t \bmod \Gamma) \in [\Gamma - w^{max}, \Gamma - 1]$ **then**
    for Each VM type $h \in \mathcal{H}$ **do**
      Determine the type of jobs type-h VMs should serve, r_h^*, using equation (28).
    end for
  end if
  Determine type of VMs new configured servers should run, h_d^*, using equation (24)
  if $[Q^d_{r_h^*}(t) + Z^d_{r_h^*}(t)]n_h^d|_{h=h_d^*} \leq Vc^d(t)$ **then**
    Keep servers running leftover jobs on, close all other servers
  else
    Keep servers running leftover jobs on, configure all other servers to run type-h_d^* VMs, use these type-h_d^* VMs to serve type-$r_h^*|_{h=h_d^*}$ jobs.
  end if
  Choose the job drop number according to Eqn. (18)
  Update the queues $Q_r^d(t)$, $Z_r^d(t)$ according to queue dynamic equations (3) (12).
end for

IV. Performance Analysis

We next analyze the performance of Algorithm 1 in terms of queueing delay bound, conditions for avoiding job dropping, and profit optimality. Detailed proofs can be found in the related technical report [11].

Theorem 1. (**Queueing Delay Bound**) *The length of workload queue Q_r^d is bounded by $Q_r^{d,max} = Vp_r^{d,max}/w_r + w_r a_r^{max}$, and the SLA of jobs can be guaranteed by $l = \lceil \frac{Q_r^{d,max} + Z_r^{d,max}}{\epsilon_r} \rceil$, where $Z_r^{d,max} = V \cdot \eta_r/w_r + \epsilon_r$ is the upper bound of the length of virtual queue Z_r^d.*

The queue length bound can be proved through induction. For Q_r^d, once its queue length exceeds $Vp_r^{d,max}/w_r$, we have $w_r Q_r^d - Vp_r^d(t) > 0$; to minimize (16), our algorithm takes $p_r^d(t) > p_r^{d,max}$ and no new jobs are admitted in the next time slot. Q_r^d will start to decrease. As the maximum increase in one time slot is $w_r a_r^{max}$, the queue length can not exceed $Vp_r^{d,max}/w_r + w_r a_r^{max}$. Similarly, for Z_r^d, once its queue length exeeds $V\eta_r/w_r$, unserved jobs are dropped, Z_r^d will start to decrease. As the maximum increase in one time slot is ϵ_r, the queue length can not exceed $V\eta_r/w_r + \epsilon_r$.

The following theorem states the conditions under which Algorithm 1 guarantees zero job dropping.

Theorem 2. (**No Job Drop Conditions**) *If the following two conditions are satisfied,*

$$N^d n_{h_r}^d w_r \geq (\sum_{r'\in\mathcal{R}} w_{r'})(w^{max} a^{max} + \epsilon^{max}), \forall r \in \mathcal{R}, \forall d \in \mathcal{D}, \tag{29}$$

$$V \cdot \frac{\eta_r}{w_r} \geq V \cdot \frac{c^{max}}{n^{min}} + (\sum_{r' \in \mathcal{R}} w_{r'})(w^{max}a^{max} + \epsilon^{max}), \quad \forall r \in \mathcal{R}, \forall d \in \mathcal{D}. \quad (30)$$

there is no job dropping in any datacenter at any time. Here, c^{max} is the maximum cost for running a server for one time slot at any datacenter. n^{min} is the minimum number of VMs a server in any datacenter can host. $\epsilon^{max} = \max\{\epsilon_r, \forall r \in \mathcal{R}\}$, $a^{max} = \max\{a_r^{max}, \forall r \in \mathcal{R}\}$.

Condition (29) means that the maximum workload reduction by scheduling type-r jobs once should be no smaller than the overall workload accumulated in the corresponding queue since last time when workload in the queue was scheduled. With condition (29), we can prove that the sum of queue lengths is upper-bounded by $V \cdot \frac{c^{max}}{n^{min}} + (\sum_{r' \in \mathcal{R}} w_{r'})(w^{max}a^{max}+\epsilon^{max})$, *i.e.*, $Q_r^d(t)+Z_r^d(t) \leq V \cdot \frac{c^{max}}{n^{min}} + (\sum_{r' \in \mathcal{R}} w_{r'})(w^{max}a^{max} + \epsilon^{max})$.

Under the bound of the sum of queue lengths, condition (30) guarantees $Q_r^d(t) + Z_r^d(t) \leq V\eta_r/w_r$. According to job dropping decisions in (18), no job dropping would happen.

Hence, to prove the theorem, it is sufficient by showing that the bound of aggregated queue length $Q_r^d(t) + Z_r^d(t) \leq V \cdot \frac{c^{max}}{n^{min}} + (\sum_{r' \in \mathcal{R}} w_{r'})(w^{max}a^{max} + \epsilon^{max})$ under condition (29). We prove it by contradiction. Assume $Q_r^d(t) + Z_r^d(t) > V \cdot \frac{c^{max}}{n^{min}} + (\sum_{r' \in \mathcal{R}} w_{r'})(w^{max}a^{max} + \epsilon^{max})$ under condition (29). As the maximum increase of the sum of queue length in one time slot is $w^{max}a^{max} + \epsilon^{max}$, to achieve a queue length exceeding $V \cdot \frac{c^{max}}{n^{min}} + (\sum_{r' \in \mathcal{R}} w_{r'})(w^{max}a^{max}+\epsilon^{max})$, $\sum_{r' \in \mathcal{R}} w_{r'}$ consecutive time slots are needed when type-r jobs are not scheduled after the sum just becomes larger than $V \cdot \frac{c^{max}}{n^{min}}$. During these $\sum_{r' \in \mathcal{R}} w_{r'}$ time slots, as the condition $[Q_{r_h^*}^d(t) + Z_{r_h^*}^d(t)]n_h^d > Vc^{max} > Vc^d(t)$, all servers in datacenter d should be turned on in our algorithm. Hence, other types of jobs other than type r will be scheduled to run among the $\sum_{r' \in \mathcal{R}} w_{r'}$ time slots. We can also prove that a type of jobs can not be scheduled twice among $\sum_{r' \in \mathcal{R}} w_{r'}$ time slots, since if the type of jobs is scheduled, its queue length will not be larger than that of type-r jobs within the remaining time slots among the $\sum_{r' \in \mathcal{R}} w_{r'}$ time slots. This implies that the total number of different types of jobs is at least $R + 1$, which contradicts the true total number of job types, R.

We next prove the performance optimality of our algorithm. Define λ^d as the vector of time-averaged workloads of datacenter d for different types of jobs, *i.e.*,

$$\lambda_r^d = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} w_r \sum_{j \in \mathcal{J}} a_r^{j,d}(t)$$

Definition 1 (Capacity region): Under the workload arrival rate vector λ^d, if there exist preemptive or non-preemptive job scheduling and server/VM provisioning algorithms that can stabilize all workload queues $Q_r^d(t), r \in \mathcal{R}$'s without job dropping or violating the SLA requirements, we say λ^d is supportable by datacenter d. The capacity region $\mathcal{C}^d$ is the set of all supportable vectors of workload arrival rates at datacenter d.

Definition 2 ($(1+\delta)$-optimal Profit): When the workload arrival rate vector at datacenter d, λ^d, satisfies $(1+\delta)\lambda^d \in \mathcal{C}^d$, the offline optimal time-averaged profit that is achievable under both preemptive and non-preemptive algorithms without job dropping or violating the SLAs is the $(1+\delta)$-optimal profit, denoted by $P^{1+\delta}$.

The following theorem establishes the supportable workload arrival rate vector and the profit optimality achieved by our dynamic non-preemptive algorithm compared with the capacity region and $(1+\delta)$-optimal profit defined above. Note that, 1-optimal profit is exactly the offline optimum P^* for the profit-maximization problem in Eqn. (4) under the SLA constraint with no job drops.

Theorem 3. (**Performance Optimality**) *When the algorithm and system parameters satisfy conditions (29)(30) and the length of a time frame satisfies $\Gamma > w^{max}$, with the assumption that the dynamic power costs, $c^d(t), \forall d \in \mathcal{D}$, are ergodic processes, there exists some $\delta > 0$, such that the supportable workload arrival rate vector λ^d by the non-preemptive Algorithm 1 satisfies $\frac{(1+\delta)\Gamma}{\Gamma - w^{max}}\lambda^d \in \mathcal{C}$, the time-averaged profit achieved by Algorithm 1 is within a constant gap from the $\frac{\Gamma}{\Gamma - w^{max}}$-optimum,* i.e.*,*

$$\begin{aligned} &\lim_{\pi \to \infty} \frac{1}{\pi\Gamma} \sum_{n=0}^{\pi-1} \sum_{t=n\Gamma}^{(n+1)\Gamma-1} \mathbb{E}\{P(t)\} \geq P^{\frac{(1+\delta)\Gamma}{\Gamma - w^{max}}} \quad (31)\\ &- \frac{B}{V} - \frac{(\Gamma - w^{max})(\Gamma - w^{max} - 1)}{2\Gamma V} B_1 \\ &- \frac{\Gamma - 1}{2V} \sum_{r \in \mathcal{R}} [(w_r a_r^{max})^2 + (\epsilon_r)^2] \\ &- \frac{(\Gamma - w^{max})(\Gamma - w^{max} - 1)}{2\Gamma V} \sum_{d \in \mathcal{D}} N^d \cdot (c^{d,max} - c^{d,min}) \\ &- \frac{w^{max}}{\Gamma} \sum_{d \in \mathcal{D}} N^d c^{d,max}, \end{aligned}$$

with $B_1 = \sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} [w_r a_r^{max} + 2\mu_r^{d,max} + \epsilon_r]\mu_r^{max}$, where the LHS is the time-averaged profit achieved by Algorithm 1 and the RHS is the $\frac{(1+\delta)\Gamma}{\Gamma - w^{max}}$-optimal profit minus a constant. $c^{(d,max)}$ and $c^{(d,min)}$ are the maximum and minimum power consumption costs for operating one server for one time slot in datacenter $d \in \mathcal{D}$.

Remark: If δ scales down infinitely close to 0, our algorithm achieves a constant gap from the $\frac{\Gamma}{\Gamma - w^{max}}$-optimum. Moreover, if $V \to \infty$, $\Gamma \to \infty$ and $\frac{\Gamma}{V} < \infty$, Algorithm 1 has a constant gap from the 1-optimum, *i.e.*, the offline optimal profit achieved by the profit-maximization problem in (4).

V. Performance Evaluation

A. Simulation Setup

Geo-distributed Datacenters. We evaluate an IaaS cloud operating three geo-distributed datacenters located in three regions of North America: *North Virginia, Oregon, Northern California*. The default configuration of the datacenters is as follows. The number of servers in each datacenter is 1000. There are 6 types of virtual machines. Each server can host

40 type-1 VMs, 30 type-2 VMs, 20 type-3 VMs, 15 type-4 VMs, 10 type-5 VMs or 5 type-6 VMs, which follow the numbers of different types of VMs that a server on Linode [6] can host. The power consumption of each active server is $1KW/h$ and the power usage effectiveness (PUE) of each datacenter is 1.6. We use real-world traces of hourly dynamic electricity prices [12] in different regions.

Job Types. The cloud provides choices among different types of VMs lasting for different time lengths. The time length, *i.e.*, the number of units of workload, is chosen among $[1, 4]$. There are $6 \times 4 = 24$ types of jobs in total. We emulate users in three zones. The utility of users in zone j when their $a_r^{j,d}$ type-r jobs are served by datacenter d is represented by a log function $U_r^{j,d} = \frac{p_r^{d,max}}{C_j} log(1 + C_j \cdot a_r^{j,d})$, according to the marginal utility diminishing law in economics. C_j represents the diminishing rate of the marginal utility of users in zone j. The larger C_j is, the fewer VMs are preferred by users in zone j. We set $[C_1, C_2, C_3] = [2 \times 10^{-4}, 4 \times 10^{-4}, 6 \times 10^{-4}]$. The maximum acceptable price for a type-r job in datacenter d is set equal to the maximum power cost for completing the job.

For comparison purposes, we implement two other strategies: (1) Static pricing with the same job scheduling and server provisioning strategies as in Algorithm 1, comparison against which will show the advantage of dynamic pricing over static pricing (such as the pricing strategy in Amazon EC2's on-demand instance market). (2) A heuristic pricing and job scheduling algorithm, which operates as follows in each time slot: *(a) Pricing.* In each datacenter, the workload of each type of jobs is still maintained in a workload queue Q_r^d. When the overall amount of workloads in Q_r^d is smaller than a threshold S_r^d, the price charged to a type-r job is set to the smallest user's willingness-to-pay that will not make newly accepted workloads exceed the queue threshold S_r^d at this time slot; when the overall amount of workloads is equal to S_r^d, the maximum price $p_r^{d,max}$ is set and no new jobs are accepted. *(b) Job and Server Scheduling.* The heuristic calculates the average price for one unit of workload, charged to jobs in each workload queue, and multiplies the average price for queue Q_r^d by the number of type-h_r VMs that one server can host in each datacenter, to obtain the profit for configuring one server to run type-h_r VMs in each datacenter. Each datacenter configures servers to run the type of VMs that achieves the largest profit and schedules the corresponding type of jobs. The cost for running one server in the current time slot in each datacenter is also calculated. If the largest profit for running one server is larger than the cost in one datacenter, the corresponding type of jobs are scheduled to servers in the datacenter; otherwise, jobs are not scheduled. The heuristic pricing and job scheduling is an algorithm without optimization for profit.

B. Profit and Cost

We run our dynamic algorithm for $T = 240$ time slots with parameters $V = 5 \times 10^5$, $\epsilon_r = 50 * w_r$, $\eta_r = 1000 \cdot p_r^{d,max}$, and $\Gamma = 100 w^{max}$. Fig. 1 presents the profit, revenue, power cost and penalty due to job drops in the cloud in each time slot. The normalized value is calculated by dividing the original value in each time slot by the maximum revenue within this period. We observe that a stable profit is achieved by our dynamic algorithm. It can be seen that no penalty is incurred, *i.e.*, no job drop occurs, which verifies our analysis on no job drop presented in Sec. IV.

TABLE II
PROFIT UNDER DIFFERENT STATIC PRICES

Portion	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Profit	1033	362	134	51	5	0	0	0	0

Fig. 2 shows the profits achieved by the three algorithms respectively. The heuristic pricing and scheduling algorithm sets S_r^d to be equal to the maximum number of type-h_r VMs that datacenter d can provide, divided by the number of job types requiring type-r VMs. The static pricing fixes the prices for each type of jobs in each datacenter above the lower bound of the power cost for completing such a job. Table II gives the profit achieved by the static pricing algorithm, by setting the static price to be different proportions of the maximum power cost. From the table we see that when the static price is 0.1 of the maximum power cost, the profit is larger than in other cases. Hence, we use 0.1 of the maximum power cost as the static price, in the comparisons with other algorithms in Fig. 2. The normalized profit is calculated by dividing the profit in each time slot by the maximum profit in one time slot within this period among the three algorithms. We can observe that our dynamic pricing algorithm outperforms the other two algorithms, and achieves stable profit over time.

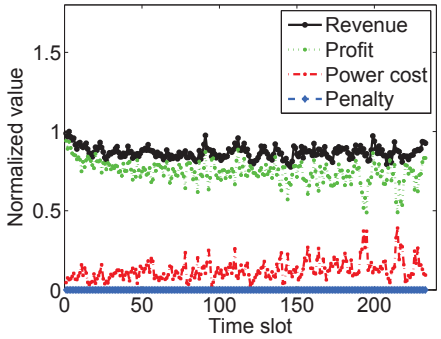

Fig. 1. Revenue, power cost, profit and penalty in each time slot.

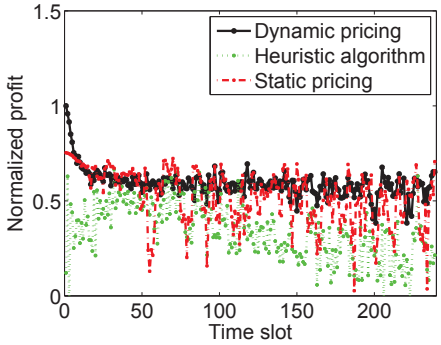

Fig. 2. Comparison of profits among different algorithms.

C. Impact of V and Γ

Fig. 3 and 4 further illustrate how the time-averaged profit achieved by our algorithm varies with different choices of V

and Γ, respectively. The time-averaged profit is normalized by being divided by the time-averaged profit under parameters $V = 5\times10^5$, $\Gamma = 100w^{max}$. Fig. 3 shows that as V increases, the time-averaged profit increases, verifying the role of V given in Theorem 3. Γ is the number of time slots in a time frame. Fig. 4 suggests that, when Γ is larger than $10w^{max}$, its value has no substantial impact on profits, revealing the fact that our two-time-scale dynamic algorithm is not sensitive to the exact length of time frames. As V increases to infinity and Γ is large enough, the time-averaged profit is arbitrarily close to a constant gap from the offline optimum.

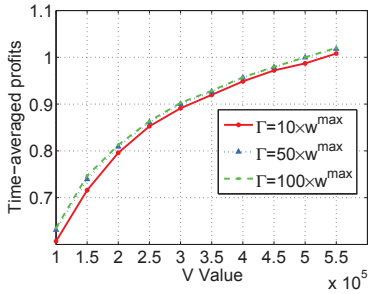

Fig. 3. Time-averaged profits under different values of V.

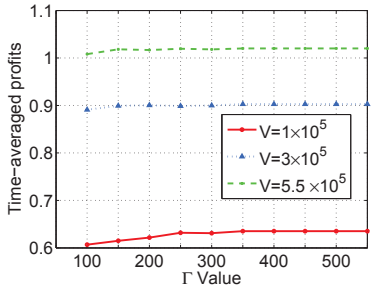

Fig. 4. Time-averaged profits under different values of Γ.

VI. Related Work

A number of studies apply auctions to price computing resources in a cloud system [13] [14] [15]. Wang *et al.* [13] model VM pricing as a multi-unit combinatorial auction, which is executed round by round without considering that users may occupy a VM for more than one decision interval. Wang *et al.* [14] model a dynamic auction where bidders may request to occupy a VM for more than one decision interval, such that the auction in one round is correlated with that in another round. Zhang *et al.* [15] provide a truthful online auction framework to process users' instantaneous and heterogeneous bids for resources. They both assume that the capacity of the cloud is fixed, without addressing server provisioning in the system.

Another group of work studies cloud resource scheduling under given pricing strategies [16] [17]. Wang *et al.* [16] study how a cloud should allocate its resources between the on-demand market and the auction market. Zhang *et al.* [17] propose a dynamic scheduling and consolidation mechanism that allocates VM resources to each spot market to maximize the cloud provider's total revenue. Differently, our work jointly models dynamical resource pricing and scheduling.

Most work that apply the Lyapunov optimization framework for workload scheduling in cloud systems implicitly assume workload that would only occupy the sources within the duration of one decision interval [5] [18]. We are aware of only one study by Maguluri *et al.* [19] that investigates the scheduling of variable-length jobs in cloud systems, using Lyapunov optimization. Their scheduling aims to stabilize queues in the system, while we target close-to-offline-optimal performance in profit maximization.

VII. Conclusion

This paper proposes an online algorithm for joint VM pricing, job scheduling and server provisioning in a cloud consisting of geo-distributed datacenters. The algorithm takes into consideration the case that the execution time of each job may be longer than the interval of online decisions. The lower bound of the time-averaged profit achieved by the algorithm is proven to approach the offline optimum minus a constant, which diminishes when appropriate parameters are chosen. We also analyze the conditions for the cloud not to drop jobs due to violating the delay constraints. Empirical studies under realistic settings validate our theoretical results.

References

[1] Windows Azure, http://www.windowsazure.com/en-us/.
[2] AMAZON EC2, http://aws.amazon.com/ec2.
[3] GCE, https://cloud.google.com/products/compute-engine.
[4] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.
[5] Y. Yao, L. B. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data Centers Power Reduction: A Two Time Scale Approach for Delay Tolerant Workloads," in *Proc. of INFOCOM*, March 2012.
[6] "LINODE," http://www.linode.com/faq.cfm#how-do-i-get-my-fair-share-of-cpu.
[7] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, p. 14, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1618525.1618528
[8] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," in *ACM SIGCOMM C. C. R.*, vol. 39, no. 1, January 2009, pp. 68–73.
[9] M. Neely, "Opportunistic Scheduling with Worst Case Delay Guarantees in Single and Multi-Hop Networks," in *Proc. of INFOCOM*, Mar. 2012.
[10] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource Allocation and Cross-Layer Control in Wireless Networks*, 2006, vol. 1, no. 1.
[11] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, and F. Lau, "Dynamic Pricing and Profit Maximization for Clouds with Geo-distributed Datacenters," Tech. Rep., `http://i.cs.hku.hk/~jzhao/CloudFederation.pdf`.
[12] Federal Energy Regulatory Commission, http://www.ferc.gov/.
[13] Q. Wang, K. Ren, and X. Meng, "When Cloud Meets ebay: Towards Effective Pricing for Cloud Computing," in *IEEE INFOCOM*, 2012.
[14] W. Wang, B. Liang, and B. Li, " Revenue Maximization with Dynamic Auctions in IaaS Cloud Markets," in *Proc. of IWQoS*, 2013.
[15] H. Zhang, B. Li, H. B. Jiang, F. M. Liu, A. V. Vasilakos, and J. C. Liu, " A Framework for Truthful Online Auctions in Cloud Computing with Heterogeneous User Demands," in *IEEE INFOCOM*, Apr. 2013.
[16] W. Wang, B. Li, and B. Liang, "Towards Optimal Capacity Segmentation with Hybrid Cloud Pricing," in *Proc. of ICDCS*, 2012.
[17] Q. Zhang, E. Grses, R. Boutaba, and J. Xiao, " Dynamic resource allocation for spot markets in clouds," in *Proc. of Hot-ICE*, 2011.
[18] K. Le, J. Zhang, J. Meng, R. Bianchini, Y. Jaluria, and T. Nguyen, "Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds," in *Supercomputing*, Nov. 2011.
[19] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters," in *Proc. of INFOCOM*, March 2012.