Virtual Machine Trading in a Federation of Clouds: Individual Profit and Social Welfare Maximization

Hongxing Li*, Chuan Wu*, Zongpeng Li† and Francis C.M. Lau*
*Department of Computer Science, The University of Hong Kong, Hong Kong, Email: {hxli, cwu, fcmlau}@cs.hku.hk †Department of Computer Science, University of Calgary, Canada, Email: zongpeng@ucalgary.ca

***Abstract*—By sharing resources among different cloud providers, the paradigm of federated clouds exploits temporal availability of resources and geographical diversity of operational costs for efficient job service. While interoperability issues across different cloud platforms in a cloud federation have been extensively studied, fundamental questions on cloud economics remain: When and how should a cloud trade resources (*e.g.*, virtual machines) with others, such that its net profit is maximized over the long run, while a close-to-optimal social welfare in the entire federation can also be guaranteed? To answer this question, a number of important, inter-related decisions, including job scheduling, server provisioning and resource pricing, should be dynamically and jointly made, while the long-term profit optimality is pursued. In this work, we design efficient algorithms for inter-cloud virtual machine (VM) trading and scheduling in a cloud federation. For VM transactions among clouds, we design a double-auction based mechanism that is strategy-proof, individual rational, ex-post budget balanced, and efficient to execute over time. Closely combined with the auction mechanism is a dynamic VM trading and scheduling algorithm, which carefully decides the true valuations of VMs in the auction, optimally schedules stochastic job arrivals with different *service level agreements* (SLAs) onto the VMs, and judiciously turns on and off servers based on the current electricity prices. Through rigorous analysis, we show that each individual cloud, by carrying out the dynamic algorithm in the online double auction, can achieve a time-averaged profit arbitrarily close to the offline optimum. Asymptotic optimality in social welfare is also achieved under homogeneous cloud settings. We carry out simulations to verify the effectiveness of our algorithms, and examine the achievable social welfare under heterogeneous cloud settings, as driven by the real-world Google cluster usage traces.**

***Index Terms*—Federated cloud, Virtual machine trading, Individual profit, Social welfare, Stochastic optimization, Double auction**

I. Introduction

The emerging federated cloud paradigm advocates sharing of disparate cloud services (in separate data centers) from different cloud providers, and interconnects them based on common standards and policies to provide a universal environment for cloud computing. Such a cloud federation exploits temporal and spatial availability of resources (*e.g.*, virtual machines) and diversity of operational costs (*e.g.*, electricity prices): when a cloud experiences a burst of incoming jobs, it may resort to VMs from other clouds with idle resources; when the electricity price for running servers and VMs is high at one cloud data center, the cloud can schedule jobs onto other cloud data centers with lower electricity charge at the moment. In this way, the aggregate job processing capacity of the cloud federation can be potentially higher than the aggregation of capacities of separate clouds operating alone, and the overall profit can be larger.

To implement the federated cloud paradigm, significant interest has arisen on developing interfaces and standards to enable cloud interoperability and job portability across different cloud platforms ([1] [2]). However, fundamental problems on cloud economics remain to be investigated. A cloud in the real world is selfish, and aims to maximize its own profit, *i.e.*, its income from handling jobs and leasing VMs to other clouds subtracting its operational costs and expenses in VM rental from other clouds. Only if its profit can be maximized and in any case not lower than when operating alone, can a cloud be incentivized to join a federation. This calls for an efficient mechanism to carry out resource trading and scheduling among federated clouds, to achieve profit maximization for individual clouds, as well as to perform well in social welfare. A number of inter-related, practical decisions are involved: (1) *VM pricing*: what mechanism should be advocated for VM sale and purchase among the clouds, and at what prices? (2) *Job scheduling*: with time-varying job arrivals at each cloud, targeting different resources and SLA requirements, should a cloud serve the jobs right away or later, to exploit time-varying electricity prices? And should a cloud serve a job using its own resources or others' resources? (3) *Server provisioning*: is it more beneficial for a cloud to keep many of its servers running to serve jobs of its own and from others, or to turn some of them down to save electricity? These decisions should be efficiently and optimally made in an online fashion, while guaranteeing long-term optimality of individual cloud's profits, as well as the social welfare.

In this paper, we design efficient algorithms for inter-cloud resource trading and scheduling, in a federation consisting of disparate cloud data centers. A double-auction based mechanism is proposed for the sell and purchase of available VMs across cloud boundaries over time. The auction is strategy-proof, individual rational, ex-post budget balanced, and computationally efficient (polynomial time complexity). Closely combined with the auction mechanism is an efficient, dynamic VM trading and scheduling algorithm, which carefully decides the true valuations of VMs to participate in the auction, optimally schedules randomly-arriving jobs with different resource requirements (*e.g.*, number of VMs) and SLAs (*e.g.*, maximum job scheduling delay) onto different data centers,

and judiciously turns on and off servers in the clouds based on the current electricity prices. The dynamic algorithm serves as an efficient strategy for each cloud to employ in the online double auction, and is proven to maximize individual profit for each cloud, over the long run of the system. The contributions of this work are summarized below.

First, among the first in the literature, we address selfishness of individual clouds in a cloud federation, and design efficient mechanisms to maximize the net profit of each cloud. This profit is not only guaranteed to be no less than that when the cloud operates alone, but also maximized over the long run to the theoretical optimum under any truthful, individual-rational and ex-post budget-balanced double auction, in the presence of time-varying job arrivals and electricity prices at the cloud.

Second, we novelly combine a truthful double auction mechanism with stochastic Lyapunov optimization techniques, and design an online VM trading and scheduling algorithm, for a cloud to optimally price the VMs and to judiciously schedule the VM and server usages. Each cloud values different VMs based on the back pressure in job queue scheduling, and bids them in the auction for effective VM acquisition.

Third, we demonstrate that by applying the dynamic algorithm in the online double auction, each cloud can achieve a time-averaged profit arbitrarily close to its offline optimum (obtained if the cloud knows complete information on incoming jobs and electricity prices in the entire time span) under any truthful, individual-rational and ex-post budget-balanced double auction. We also prove that the social welfare, *i.e.*, the time-averaged overall profit in the federation, can be asymptotically maximized when the number of clouds grows, under homogenous cloud settings. Trace-driven simulations examine the achievable social welfare with our dynamic algorithm under heterogenous settings.

In the rest of the paper, we discuss related literature in Sec. II, present the system model in Sec. III, and introduce the detailed resource trading and scheduling mechanisms in Sec. IV. A double auction mechanism is proposed in Sec. V, and a benchmark social-welfare maximization algorithm is discussed in Sec. VI. Theoretical analysis and simulation studies are presented in Sec. VII and Sec. VIII, respectively. Sec. IX concludes the paper.

II. Related work

A. Optimal Scheduling in Cloud Systems

Most existing literature ([3]–[8] and references therein) on resource scheduling in cloud systems focus on a single cloud that operates alone. A common theme is to minimize the operational costs (mainly consisting of electricity bills) in one or multiple data centers of the cloud, while providing certain performance guarantee of job scheduling, *e.g.*, in terms of average job completion times [3]–[6].

Urgaonkar *et al.* [5] propose an algorithm with joint job admission control, routing and resource allocation for power consumption reduction in a virtualized data center. Rao *et al.* [3] advocate minimization of electricity expenses by exploiting the temporal and spatial diversities of electricity prices. Yao *et al.* [6] minimize the power cost with a two-time scale algorithm for delay tolerant workloads. Ren *et al.* [4] also aim to minimize the energy cost while addressing the fairness in resource allocation. All the above works provide *average* delay guarantees for job services.

Ghodsi *et al.* [7], [8] study the fair resource allocation for multiple resource types, based on the *dominant resource fairness*, without optimizing the profit of clouds.

Different from these studies on a stand-alone cloud with centralized control, this work investigates profit maximization for individual selfish clouds in a federation, where each participant makes its own decisions. Besides, bounded scheduling delay for each job is guaranteed even in worst cases, contrasting the existing solutions that ensure average delays.

B. Resource Trading Mechanisms

A rich body of literature is devoted to resource trading in grid computing [9] and wireless spectrum leasing [10] [11]. Various mechanisms have been studied, *e.g.*, bargaining [9], fixed or dynamic pricing based on a contract or the supply-demand ratio [12], and auctions [10] [11].

A bargaining mechanism [9] typically has an unacceptable complexity by negotiating between each pair of traders. Fixed pricing, *e.g.*, Amazon EC2 on-demand instances, has been shown to be inefficient in social welfare maximization in cases of system dynamics [13]. Dynamic pricing, such as Amazon EC2 spot instances, could be inefficient too, where the participants can quote the resources untruthfully [14].

Auction stands out as a promising mechanism, on which there have been abundant solutions ([10], [11] and references therein) with truthful design and polynomial complexity. Although some recent works [13]–[15] aim to design an auction mechanism with individual rationality (non-negative profit gain) for trading in federated clouds, they do not explicitly address individual profit maximization over the long run, nor other desirable properties such as truthfulness, ex-post budget balance, and social welfare maximization. Moreover, little literature on auctions provides methods to quantitatively calculate the true valuations in each bid, which are simply assumed as known. Our design addresses these issues.

III. System Model and Auction Framework

A. Federation of Clouds

We consider a federation of F clouds, each located at a different geometric location and operates autonomously to gain profit by serving its customers' job requests, managing server provisioning and trading resources with other clouds.

Service demands: Each individual cloud $i \in \{1, \ldots, F\}$ has a front-end proxy server, which accepts job requests from its customers. There are S types of jobs serviced at each cloud, each specified by a three-tuple $< m_s, g_s, d_s >$. Here, $m_s \in \{1, \ldots, M\}$ specifies the type of the required VM instances, where M is the maximum number of VM types, and each type corresponds to a different set of configurations of CPU, storage and memory[1]; g_s is the number of type-m_s VMs that the job needs simultaneously (See Amazon

[1]We can also consider other resource configurations, *e.g.*, bandwidth, to define the VM types. Our general problem model and the proposed solutions are still applicable to those cases.

EC2 API [12]); and d_s stands for the SLA (Service Level Agreement) of job type $s \in \{1,\dots,S\}$, evaluated by the maximal response delay for scheduling a job, *i.e.*, the time-span from when the job arrives to when it starts to run on scheduled VMs. In a cloud in practice, it is common to buy servers of the same configuration and provision the same type of VMs on one machine [16]. Therefore, we suppose each cloud i has N_i^m homogenous servers to provision VMs of type $m \in \{1,\dots,M\}$, each of which can provide a maximum of C_i^m VMs of this type; the total number of servers in cloud i is $\sum_{m=1}^{M} N_i^m$.

The system runs in a time-slotted fashion. At the beginning of each time slot t, $r_i^s(t) \in \{0,\dots,R_i^s\}$ jobs arrive at cloud i, for each job type s. R_i^s is an upper-bound on the number of type-s jobs submitted to cloud i in a time slot. The arrival of jobs is an ergodic process at each cloud. We suppose the arrival rate is given, and how a customer decides which cloud to use is orthogonal to this study. Let $p_i^s(t) \in [0, p_i^{s(max)}]$ be the *given* service charge to the customer by cloud i, for accepting a job of type s in time slot t, which remains fixed within a time slot, but may vary across time slots.[2] Here, $p_i^{s(max)}$ is the max possible price for $p_i^s(t)$. Such a general charging model subsumes pricing schemes in practice: *e.g.*, time-independent $p_i^s(t)$ corresponds to the *on-demand* VM charging scheme, while time-varying $p_i^s(t)$ can represent the *spot instance* prices based on the current demand *vs*. supply [12].

Job scheduling: Each incoming job to cloud i enters a FIFO queue of its type — a cloud i maintains a queue to buffer unscheduled jobs of each type s, with $Q_i^s(t)$ as its length in t. When the required VMs of a job are allocated, the job departs from its queue and starts to run on the VMs. A cloud may schedule its jobs on either its own VMs or VMs leased from other clouds, for the best economic benefits. Let $\mu_{ij}^s(t)$ be the number of type-s jobs of cloud i that are scheduled for processing in cloud j at the beginning of time slot t.[3]

When a job's demanded maximum tolerable response time (the SLA) cannot be met, in cases of system overload, it is dropped. A penalty is enforced in this case, to compensate for the customer's loss. Let

$$D_i^s(t) \in \{0,\dots,D_i^{s(max)}\} \tag{1}$$

be the number of type-s jobs dropped by cloud i in t, where $D_i^{s(max)}$ is the maximum value of $D_i^s(t)$. Let ξ_i^s be the penalty to drop one such job, which is at least the maximum price charged to customers when accepting the jobs, *i.e.*, $\xi_i^s \geq p_i^{s(max)}$.

Hence, the number of unscheduled jobs buffered at each cloud $i \in \{1,\dots,F\}$ can be updated with the following queueing law:

$$\begin{aligned} Q_i^s(t+1) = \max\{Q_i^s(t) - \sum_{j=1}^{F} \mu_{ij}^s(t) - D_i^s(t), 0\} \\ + r_i^s(t), \quad \forall s \in \{1,\dots,S\}. \end{aligned} \tag{2}$$

Job scheduling should satisfy the following SLA constraint:

Each type-s job in cloud i is either scheduled or dropped (subject to a penalty) before its maximum tolerable response delay d_s,

$$\forall s \in \{1,\dots,S\}. \tag{3}$$

We apply the $\epsilon-$persistence queue technique [18], to create a virtual queue Z_i^s associated with each job queue Q_i^s ($\forall i \in \{1,\dots,F\}$):

$$\begin{aligned} Z_i^s(t+1) = \max\{Z_i^s(t) + \mathbf{1}_{\{Q_i^s(t)>0\}} \cdot [\epsilon_s - \sum_{j=1}^{F} \mu_{ij}^s(t)] - D_i^s(t) \\ - \mathbf{1}_{\{Q_i^s(t)=0\}} \cdot \sum_{j=1}^{F} \frac{C_j^{m_s} \cdot N_j^{m_s}}{g_s}, 0\}, \forall s \in \{1,\dots,S\}. \end{aligned} \tag{4}$$

Here, $\epsilon_s > 0$ is a constant. $\mathbf{1}_{\{Q_i^s(t)>0\}}$ and $\mathbf{1}_{\{Q_i^s(t)=0\}}$ are indicator functions such that

$$\mathbf{1}_{\{Q_i^s(t)>0\}} = \begin{cases} 1 & \text{if } Q_i^s(t) > 0 \\ 0 & \text{Otherwise} \end{cases}; \ \mathbf{1}_{\{Q_i^s(t)=0\}} = \begin{cases} 1 & \text{if } Q_i^s(t) = 0 \\ 0 & \text{Otherwise} \end{cases}.$$

Length of this virtual queue approximately reflects the cumulated response delay of jobs from the respective job queue. The constant ϵ_s is added so as to approximately account for the cumulated delay of unscheduled jobs when the job queue is not empty, while the approximated cumulated delay is reduced with $\sum_{j=1}^{F} \frac{C_j^{m_s} \cdot N_j^{m_s}}{g_s}$ when all jobs are scheduled, *i.e.*, the job queue is empty. Our algorithm seeks to bound the lengths of job queues and virtual queues, with properly set ϵ_s, and hence the maximum response delay of jobs can be bounded, *i.e.*, constraint (3) is satisfied.

Server provisioning: We consider electricity cost, for running and cooling the servers [19], as the main component of the operational cost in a cloud. Other costs, *e.g.*, space rental and labour, remain relatively fixed for a long time, and are of less interest. Given that electricity prices vary at different locations and from time to time [3] [20], we model the operational cost $\beta_i(t)$ in each cloud i as a general ergodic process over time, varying across time slots between $\beta_i^{(min)}$ and $\beta_i^{(max)}$.

Each cloud strategically decides the number of active servers at each time, to optimize its profit. Let $n_i^m(t)$ be the number of active servers provisioning type-m VMs at cloud i in t. The available server capacities at each cloud $i \in \{1,\dots,F\}$ constrain the feasible job scheduling at time t:

$$\sum_{j=1}^{F} \sum_{s:m_s=m, s\in\{1,\dots,S\}} g_s \mu_{ji}^s(t) \leq C_i^m \cdot n_i^m(t), \ \forall m \in \{1,\dots,M\}, \tag{5}$$

$$n_i^m(t) \leq N_i^m, \ \forall m \in \{1,\dots,M\}. \tag{6}$$

(5) states that the overall demand for type-m VMs in cloud i from itself and other clouds should be no larger than the maximum number of available type-m VMs on the active servers in cloud i. Here $g_s \mu_{ji}^s(t)$ is the total number of VMs needed by type-s jobs scheduled from cloud j to cloud i in t. Motivated by practical job execution efficiency, we only consider scheduling a job to VMs from a single cloud, but not VMs across different clouds. (6) ensures that the number of active servers is limited by the total number of on-premise servers of the corresponding VM configuration at each cloud.

[2]The optimal pricing mechanism for selling the VMs to customers is an orthogonal topic and discussed in another paper in [17].

[3]Once a job is scheduled to run, it will be served and allocated with all required resources until its completion and will not be migrated to other clouds.

B. Inter-cloud VM Trading with Double Auction

In an inter-cloud resource market, VMs constitute the items for trading. For each type of VMs, multiple clouds may have them on sale while multiple other clouds can request them. A double auction is a natural fit to implement efficient trading in this case, allowing both selling and buying clouds to actively participate in pricing, on behalf of their own benefits. In our dynamic system, a *multi-unit double auction* is carried out among the clouds at the beginning of each time slot, deciding the VM trades within that time slot.

Buyers & Sellers: A cloud can be both a buyer and a seller. A buy-bid $< b_i^m(t), \gamma_i^m(t) >$ records the unit price and maximum quantity at which cloud i is willing to buy VMs of type m, in t. Similarly, a sell-bid $< s_i^m(t), \eta_i^m(t) >$ records the unit price and maximum quantity at which cloud i is willing to sell VMs of type m in t.

Let $\tilde{b}_i^m(t)$ and $\tilde{s}_i^m(t)$ be cloud i's true valuation of buying and selling a type-m VM respectively (the max/min price it is willing to pay/accept). Similarly, let $\tilde{\gamma}_i^m(t)$ and $\tilde{\eta}_i^m(t)$ be cloud i's true valuation of the quantity to buy and sell VMs of type m respectively (the maximum volume of VMs it is willing to purchase/sell). A cloud i may strategically manipulate the bid prices and volumes, in the hope of maximizing its profit. We show in Sec. VII that the double auction proposed in Sec. IV is truthful, such that each bid price reveals the true valuation.

Auctioneer: We assume that there is a broker in the cloud federation, assuming the role of the auctioneer. After collecting all the buy and sell bids, the auctioneer executes a double auction to be detailed in Sec. V, to decide the set of successful buy and sell bids, their clearing prices and the numbers of VMs to trade in each type. Let $\hat{b}_i^m(t)$ be the actual charge price for cloud i to buy one type-m VM, and $\hat{\gamma}_i^m(t)$ be the actual number of VMs purchased. Similarly, let $\hat{s}_i^m(t)$ be the actual income cloud i receives for selling one type-m VM, and $\hat{\eta}_i^m(t)$ be the actual number of VMs sold.

Let $\alpha_{ij}^m(t)$ be the number of type-m VMs that cloud $i \in \{1,\dots,F\}$ purchases from cloud $j \in \{1,\dots,F\}$ in t, as decided by the auctioneer:

$$\hat{\gamma}_i^m(t) = \sum_{j\in\{1,\dots,F\}, j\neq i} \alpha_{ij}^m(t),\ \forall m \in \{1,\dots,M\}, \tag{7}$$

$$\hat{\eta}_i^m(t) = \sum_{j\in\{1,\dots,F\}, j\neq i} \alpha_{ji}^m(t),\ \forall m \in \{1,\dots,M\}. \tag{8}$$

Since VMs are purchased for serving jobs, the job scheduling decisions $\mu_{ij}^s(t)$ at each cloud $i \in \{1,\dots,F\}$, are related to the number of VMs it purchases:

$$\sum_{s:s\in\{1,\dots,S\}, m_s=m} g_s \cdot \mu_{ij}^s(t) = \alpha_{ij}^m(t),$$
$$\forall m \in \{1,\dots,M\}, \forall i,j \in \{1,\dots,F\}, i \neq j. \tag{9}$$

Three economic properties are desirable for designing the auctioneer's mechanism. (i) *Truthfulness*: Bidding true valuations is a dominant strategy, and consequently, both bidder strategies and auction design are simplified. (ii) *Individual Rationality*: Each cloud obtains a non-negative profit by participating in the auction. (iii) *Ex-post Budget Balance*: The auctioneer has a non-negative surplus, *i.e.*, the total payment from all winning buy-bids is no less than the total charge for all winning sell-bids in each time slot.

TABLE I
NOTATION: INPUT QUANTITIES AND INTERMEDIATE VARIABLES

F	# of clouds	S	# of service types
M	# of VM types	m_s	VM type of service type s
d_s	Max. response delay of service type s	g_s	# of VMs required by service type s

$r_i^s(t)$	# of type-s jobs arrived at cloud i, slot t
R_i^s	Max. # of type-s jobs arrived at cloud i per slot
$p_i^s(t)$	Service price for each job of type s at cloud i, slot t
$p_i^{s(max)}$	Max. service price for each type-s job at cloud i per slot
$\beta_i(t)$	Cost for operating an active server at cloud i, slot t
$\beta_i^{(min)}$	Min. cost for operating an active server at cloud i per slot
$\beta_i^{(max)}$	Max. cost for operating an active server at cloud i per slot
ξ_i^s	Penalty for dropping a type-s job at cloud i
$D_i^{s(max)}$	Max. # of type-s jobs cloud i drops per slot
C_i^m	Max. # of type-m VMs an active server at cloud i provisions
N_i^m	Total # of servers provisioning type-m VMs at cloud i
$Q_i^s(t)$	Length of queue buffering type-s jobs at cloud i, slot t
$Z_i^s(t)$	Length of virtual queue of type-s jobs at cloud i, slot t
ϵ_s	Constant positive parameter for $Z_i^s(t), \forall i \in \{1,\dots,F\}$
$Q_i^{s(max)}$	Maximum length of queue $Q_i^s(t)$
$Z_i^{s(max)}$	Maximum length of virtual queue $Z_i^s(t)$
V	User-defined constant positive parameter for dynamic algorithm

TABLE II
NOTATION: DECISION VARIABLES AT INDIVIDUAL CLOUDS

$\mu_{ij}^s(t)$	# of type-s jobs scheduled from cloud i to cloud j, slot t
$n_i^m(t)$	# of active servers providing type-m VMs at cloud i, slot t
$D_i^s(t)$	# of dropped type-s jobs at cloud i, slot t
$\tilde{s}_i^m(t)$	True value of selling one type-m VM from cloud i, slot t
$\tilde{\eta}_i^m(t)$	True value of volume to sell type-m VMs from cloud i, slot t
$s_i^m(t)$	Bid price for selling one type-m VM from cloud i, slot t
$\eta_i^m(t)$	Max. # of type-m VMs cloud i can sell, slot t
$\tilde{b}_i^m(t)$	True value of buying one type-m VM by cloud i, slot t
$\tilde{\gamma}_i^m(t)$	True value of volume to buy type-m VMs by cloud i, slot t
$b_i^m(t)$	Bid price for buying one type-m VM by cloud i, slot t
$\gamma_i^m(t)$	Max. # of type-m VMs cloud i can buy, slot t

TABLE III
NOTATION: DECISION VARIABLES AT THE AUCTIONEER

$\hat{s}_i^m(t)$	Actual price of selling one type-m VM from cloud i, slot t
$\hat{\eta}_i^m(t)$	Actual # of type-m VMs sold from cloud i, slot t
$\hat{b}_i^m(t)$	Actual price of buying one type-m VM by cloud i, slot t
$\hat{\gamma}_i^m(t)$	Actual # of type-m VMs bought by cloud i, slot t
$\alpha_{ij}^m(t)$	Actual # of type-m VMs sold from cloud j to i, slot t
$\theta_j^m(t)$	The j^{th} highest buy-bid price for type-m VMs at auctioneer
$\vartheta_j^m(t)$	The j^{th} lowest sell-bid price for type-m VMs at auctioneer
$L_j^m(t)$	Max. # of type-m VMs to sell, in sell-bid with j^{th} lowest price at auctioneer in t

We will demonstrate in Sec. VII that our proposed double auction mechanism in Sec. V can achieve the above three properties.

C. Individual Selfishness

Each cloud in the federation aims to maximize its time-averaged profit (revenue minus cost) over the long run of the system, while striking to fulfill the resource and SLA requirements of each job.

Revenue: A cloud has two sources of revenue: i) job service charges paid by its customers, and ii) the proceeds from VM sales. The time-averaged revenue of cloud $i \in \{1,\dots,F\}$ by undertaking different types of jobs from its customers is

$$\Phi_1^i = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{s=1}^{S} \mathbb{E}\{p_i^s(t) \cdot r_i^s(t)\}. \tag{10}$$

We assume the front-end charges, $p_i^s(t)$, from a cloud to its customers, are given. Hence, this part of the revenue is fixed in each time slot. The time-averaged income of cloud $i \in \{1,\ldots,F\}$ from selling VMs to other clouds is:

$$\Phi_2^i = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{m=1}^{M}\mathbb{E}\{\hat{s}_i^m(t)\cdot\hat{\eta}_i^m(t)\}. \quad (11)$$

Cloud i can control this income by adjusting its sell-bids, *i.e.*, $s_i^m(t)$ and $\eta_i^m(t)$, $\forall m \in \{1,\ldots,M\}$, at each time.

Cost: The cost of cloud i consists of three parts: i) operational costs incurred for running its active servers, ii) the penalties for dropping jobs, and iii) the expenditure on buying VMs from other clouds. The time-averaged cost for operating servers at each cloud $i \in \{1,\ldots,F\}$ is decided by the number of active servers in each time, *i.e.*,

$$\Psi_1^i = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{\beta_i(t)\cdot\sum_{m=1}^{M}n_i^m(t)\}. \quad (12)$$

The time-averaged penalty at each cloud $i \in \{1,\ldots,F\}$ is determined by the number of dropped jobs over time, *i.e.*, $D_i^s(t),\ \forall s \in \{1,\ldots,S\}$, $t \in [0, T-1]$:

$$\Psi_2^i = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{s=1}^{S}\mathbb{E}\{\xi_i^s\cdot D_i^s(t)\}. \quad (13)$$

The time-averaged expenditure for VM purchases is decided by the actual VM trading prices and numbers, as decided by the buy-bids $(b_i^m(t), \gamma_i^m(t))$ from cloud $i \in \{1,\ldots,F\}$:

$$\Psi_3^i = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{\sum_{m=1}^{M}\hat{b}_i^m(t)\cdot\hat{\gamma}_i^m(t)\}. \quad (14)$$

Profit Maximization: The profit maximization problem at cloud $i \in \{1,\ldots,F\}$ can be formulated as follows:

$$\max \quad \Phi_1^i + \Phi_2^i - \Psi_1^i - \Psi_2^i - \Psi_3^i \quad (15)$$
$$s.t. \quad \textbf{Constraints (1)-(9)}.$$

D. Social Welfare

Social welfare is the overall profit of the cloud federation:

$$\sum_{i=1}^{F}(\Phi_1^i + \Phi_2^i - \Psi_1^i - \Psi_2^i - \Psi_3^i).$$

Since the income and expenditure due to VM trades among the clouds cancel each other, the formula above equals $\sum_{i=1}^{F}(\Phi_1^i - \Psi_1^i - \Psi_2^i)$. The social welfare maximization problem is:

$$\max \quad \sum_{i=1}^{F}(\Phi_1^i - \Psi_1^i - \Psi_2^i) \quad (16)$$
$$s.t. \quad \text{Constraints (1)-(6)},\ \forall i \in \{1,\ldots,F\}$$

which globally optimizes server provisioning and job scheduling in the federation and maximally serves all the incoming jobs at the minimum cost, regardless of the specific inter-cloud VM trading mechanism.

When a double auction mechanism is truthful, individual rational and ex-post budget balancing, it is shown that efficiency in terms of social welfare maximization cannot be achieved concurrently [21]. We hence make a necessary compromise in social welfare in our auction design, *i.e.*, the sum of maximal individual profits derived by (15) will be smaller than the optimal social welfare from (16). Nevertheless, we will show in Sec. VII and Sec. VIII that our mechanisms still manage to achieve a satisfactory social welfare in the long run.

Tables I, II and III summarize important notation in the paper, for ease of reference.

E. Workflow

During each timeslot, the entire VM-trading framework works as follows,

- *Step 1*: Each individual cloud evaluates the prices and volumes to buy and sell VMs, and proposes its buy-bid and sell-bid to the auctioneer.
- *Step 2*: The auctioneer executes the double auction and determines the winner in current round.
- *Step 3*: The VM pricing for buying and selling VMs is calculated for each individual cloud by the auctioneer. VMs are allocated (traded) from one individual cloud to another.
- *Step 4*: Based on the auctions results, each individual cloud determines its job scheduling and server provisioning.

IV. Dynamic individual-profit maximization algorithm

We next present a dynamic algorithm for each cloud to trade VMs and scheduling jobs/servers, which is in fact applicable under any truthful, individual-rational and ex-post budget balanced double auction mechanism. We will also tailor a double auction mechanism on the auctioneer in the next section. Fig. 1 illustrates the relation among these algorithm modules.

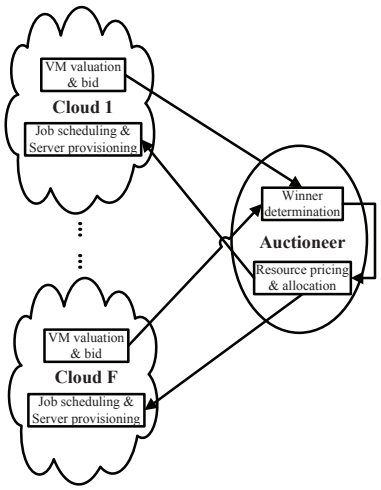


Fig. 1. Key algorithm modules.

The goal of the dynamic algorithm at each cloud i is to maximize its time-averaged profit, *i.e.*, to solve optimization (15), by dynamically making decisions in each time slot. We apply the *drift-plus-penalty* framework in Lyapunov optimization theory [22], and derive a one-shot optimization problem to be solved by cloud i in each time slot t as follows. We will prove in Sec. VII that by optimally solving the one-shot optimization at each cloud during each time slot, the dynamic algorithm can achieve a time-averaged individual profit arbitrarily close to its offline optimum (computed with complete knowledge in the entire time span), for each cloud.

A. The One-shot Optimization Problem

Define the set of queues at cloud i in each time slot t as

$$\Theta_i(t) = \{Q_i^s(t), Z_i^s(t) | s \in \{1,\ldots,S\}\}.$$

Since the network stability is achieved only if all queues in the network are kept stable [22] (We will show the network stability by our algorithm with Lemma 1 in Sec. VII) and the job scheduling/dropping decisions determine the update of job queues and virtual queues simultaneously, we jointly consider both job queues and virtual queues in the Lyapunov optimization framework and define the Lyapunov function as follows:

$$L(\Theta_i(t)) = \frac{1}{2}\sum_{s=1}^{S}[(Q_i^s(t))^2 + (Z_i^s(t))^2].$$

Then the one-slot conditional Lyapunov drift [22] is

$$\Delta(\Theta_i(t)) = L(\Theta_i(t+1)) - L(\Theta_i(t)).$$

Squaring the queuing laws (2) and (4), we can derive the following inequality (details can be found in our technical report [23]):

$$\begin{aligned}
&\Delta(\Theta_i(t)) - V\cdot[\sum_{m=1}^{M}[\hat{s}_i^m(t)\hat{\eta}_i^m(t) - \hat{b}_i^m(t)\hat{\gamma}_i^m(t) - \beta_i(t)n_i^m(t)] \\
&+\sum_{s=1}^{S}[p_i^s(t)\cdot r_i^s(t) - D_i^s(t)\xi_i^s]] \\
\le& B_i + \sum_{s=1}^{S}[Q_i^s(t)r_i^s(t) + Z_i^s(t)\epsilon_s - Vp_i^s(t)\cdot r_i^s(t)] \\
&-\varphi_1^i(t) - \varphi_2^i(t) - \varphi_3^i(t), \qquad (17)
\end{aligned}$$

where $V > 0$ is a user-defined positive parameter for gauging the optimality of time-averaged profit, $B_i = \frac{1}{2}\sum_{s=1}^{S}[(\sum_{j=1}^{F}C_j^{m_s}N_j^{m_s}/g_s + D_i^{s(max)})^2 + (R_i^s)^2 + (\epsilon_s)^2 + (D_i^{s(max)} + \sum_{j=1}^{F}C_j^{m_s}N_j^{m_s}/g_s)^2]$ is a constant, and

$$\varphi_1^i(t) = V\sum_{m=1}^{M}[\hat{s}_i^m(t)\hat{\eta}_i^m(t) - \hat{b}_i^m(t)\hat{\gamma}_i^m(t) - \beta_i(t)n_i^m(t)],$$

$$\varphi_2^i(t) = \sum_{s=1}^{S}\sum_{j=1}^{F}\mu_{ij}^s(t)[Q_i^s(t) + Z_i^s(t)],$$

$$\varphi_3^i(t) = \sum_{s=1}^{S}D_i^s(t)[Q_i^s(t) + Z_i^s(t) - V\cdot\xi_i^s].$$

Based on the drift-plus-penalty framework [22], a dynamic algorithm can be derived for each cloud i, which observes the job and virtual queues ($\Theta_i(t)$), job arrival rates ($r_i^s(t),\ \forall s \in \{1,\ldots,S\}$), the current cost for server operation ($\beta_i(t)$) in each time slot, and minimizes the RHS of the inequality (17), such that a lower bound for time-averaged profit of cloud i is maximized. Note that $B_i + \sum_{s=1}^{S}[Q_i^s(t)r_i^s(t) + Z_i^s(t)\epsilon_s - Vp_i^s(t)\cdot r_i^s(t)]$ in the RHS of (17) is fixed in time slot t. Hence, to maximize a lower bound of the time-averaged profit for cloud i, the dynamic algorithm should solve the one-shot optimization problem in each time slot t as follows:

$$\begin{aligned}
\max\quad & \varphi_1^i(t) + \varphi_2^i(t) + \varphi_3^i(t) \qquad (18)\\
s.t.\quad & \text{Constraints (1), (5)-(9).}
\end{aligned}$$

The maximization problem in (18) can be decoupled into two independent optimization problems:

$$\max\ \varphi_1^i(t) + \varphi_2^i(t) \quad s.t.\ \text{Constraints (5)-(9)}, \qquad (19)$$

which is related to optimal decisions on i) buy/sell bids for different types of VMs, and ii) scheduling of active servers and jobs to these servers; and

$$\max\ \varphi_3^i(t) \quad s.t.\ \text{Constraint (1)}, \qquad (20)$$

which is related to optimal decisions on iii) jobs to drop. In the following, we design algorithms to derive the optimal decisions based on problem (19) and problem (20).

It should be noted that, each individual cloud is maximizing its profit under a truthful double auction framework, with which the dominant strategy is bidding with true evaluation of the VMs while without considering the other clouds' actions. Once the true value is found and bided with, the individual cloud has already achieved the best it could obtain for problem (19). The solution to job scheduling and server provisioning is just a follow-up action once it gets the auction results. The optimal decisions to all variables in problem (19), *i.e.*, a) at which price/volume the VMs should be bided, and b) how the job scheduling and server provisioning are conducted with the auctions results, are in fact jointly determined during the VM evaluation process.

B. VM Valuation and Bid

Optimization problem (19) is related to the actual charges that cloud i pays for each type of VMs purchased, $\hat{b}_i^m(t)$ and $\hat{s}_i^m(t)$ ($\forall m \in \{1,\ldots,M\}$), and the actual numbers of traded VMs, $\hat{\gamma}_i^m(t)$ and $\hat{\eta}_i^m(t)$ ($\forall m \in \{1,\ldots,M\}$), from the double auction. These values are determined by the auctioneer according to buy-bids $(b_i^m(t), \gamma_i^m(t))$ and sell-bids $(s_i^m(t), \eta_i^m(t))$ submitted by all clouds, and its double auction mechanism. That is, each cloud i first proposes its buy-bids and sell-bids to the auctioneer, and then receives the auction results, based on which the *job scheduling and server provisioning* decisions are made. We first investigate how each cloud proposes its buy-bids and sell-bids, and then decide optimal job scheduling and server provisioning in Sec. IV-C.

A truthful double auction (to be introduced in Sec. V) is employed at the auctioneer, where sellers and buyers bid their *true values* of the prices and quantities, in order to maximize their individual utilities. (19) is the utility maximization problem for each cloud. If we can find true values of each cloud i, $\tilde{b}_i^m(t), \tilde{\gamma}_i^m(t)$, $\tilde{s}_i^m(t)$ and $\tilde{\eta}_i^m(t)$, and let the cloud bid using these values, the achieved utility in (19) is guaranteed to be the largest, as compared to bidding any other values.[4]

We decide the true values of the bids for each cloud i, according to their definitions in double auctions [10] [11]. The true value of the price to buy (sell) a type-m VM, $\tilde{b}_i^m(t)$ ($\tilde{s}_i^m(t)$), is such a value that, if a VM is purchased (sold) at a price (i) equal to this value, then cloud i's profit remains the same, compared to not obtaining the VM; (ii) higher than this value, a profit loss (gain) at cloud i occurs; and (iii) lower than this value, a profit gain (loss) results. In a multi-unit double auction, the true value of the maximum number of type-m VMs cloud i can buy (sell), $\tilde{\gamma}_i^m(t)$ ($\tilde{\eta}_i^m(t)$), is the maximum number of type-m VMs the cloud is willing to buy (sell) at the true value of the price, *i.e.*, $\tilde{b}_i^m(t)$ ($\tilde{s}_i^m(t)$).

Using the above rationale and based on problem (19), the true values of the buy/sell prices for cloud i can be derived as (detailed derivation steps are given in [23])

[4]In a truthful auction, the best strategy for each bidder is to bid with its true value in order to maximize its utility, since each winning buyer (seller) is charged (paid) at a price no higher (lower) than its true value of buy-bid (sell-bid).

$$\tilde{b}_i^m(t) = \frac{Q_i^{s_m^*}(t) + Z_i^{s_m^*}(t)}{V \cdot g_{s_m^*}}, \tag{21}$$

and

$$\tilde{s}_i^m(t) = \max\{\frac{Q_i^{s_m^*}(t) + Z_i^{s_m^*}(t)}{V \cdot g_{s_m^*}}, \beta_i(t)/C_i^m\}, \tag{22}$$

respectively, where

$$s_m^* = arg \max_{s' \in \{1,\ldots,S\}, m_{s'}=m} \{W_i^{s'}(t)\}, \tag{23}$$

$$\text{and} \qquad W_i^{s'}(t) = \frac{Q_i^{s'}(t) + Z_i^{s'}(t)}{g_{s'}}. \tag{24}$$

Here, $W_i^{s'}(t)$ denotes the weight for scheduling one type-s' job (to run on type-$m_{s'}$ VM(s)) by cloud i in t, and s_m^* specifies the job type with the largest weight (ties broken arbitrarily), among all types of jobs requiring type-m VMs. $W_i^{s'}(t)$ is determined by the following factors: (i) the sum of queue backlogs, $Q_i^{s'}(t) + Z_i^{s'}(t)$, representing the level of urgency for scheduling type-s' jobs in t, since $Q_i^{s'}(t)$ is the number of unscheduled type-s' jobs and $Z_i^{s'}(t)$ reflects the cumulated response delay; (ii) the number of concurrent VMs each type-s' job requires, $g_{s'}$, which decides the job-scheduling difficulty.

The intuition behind (21) and (22) includes: (i) the true value of the price to buy a type-m VM depends on the combined effect of urgency and difficulty for scheduling jobs requiring this type of VMs, and is computed based on the maximum weight that any type of jobs requiring type-m VMs may achieve; (ii) the true value of the price to sell one type-m VM from cloud i is the same as that of the price to buy, if the latter exceeds the current cost of operating a type-m VM in the cloud; otherwise, it is set to the operational cost.

The true values of the number of type-m VMs to buy and to sell at cloud i are

$$\tilde{\gamma}_i^m(t) = \sum_{j=1}^{F} C_j^m \cdot N_j^m, \tag{25}$$

$$\text{and} \qquad \tilde{\eta}_i^m(t) = C_i^m \cdot N_i^m, \tag{26}$$

respectively. They state that the maximum number of type-m VMs cloud i is willing to buy (sell) at the price in (21) (in (22)), is the number of all potential type-m VMs in the federation. The rationale is as follows: The clearing price for transactions of type-m VMs in the double auction is at most the buyer's true value in (21) and at least the seller's true value in (22), if the corresponding buy/sell bids are successful. By definition of the true value, if the actual charge per VM is lower (higher) than the true value, a profit gain happens at the buyer (seller), and the more VMs purchased (sold), the larger the profit gain. Therefore, a cloud is willing to buy or sell at the largest quantity possible, for profit maximization.[5]

The buy-bid price for type-m VMs calculated in Eqn. (21) is proportional to the sum of the lengths of queues with the maximum scheduling weight, *i.e.*, $Q_i^{s_m^*}(t) + Z_i^{s_m^*}(t)$, out of all queues of job types asking for type-m VMs, while inversely proportional to $V \cdot g_{s_m^*}$. According to our analysis in Sec. VII.D, we typically use large V, in order for the system to approach individual profit and social welfare optimality over the long run. Since V is large and the sum of queue lengths should overwhelm $V \cdot g_{s_m^*}$, if the cloud wins in the auction, there must be many jobs in its queue, such that all the VMs bought will be used (*i.e.*, no leftover capacity). On the other hand, if there are not enough jobs in the queue of type s_m^*, the buy-bid price would be low and the buyer would not win in the auction.

To conclude, in each time slot t, cloud i submits its bids as $b_i^m(t) = \tilde{b}_i^m(t)$, $s_i^m(t) = \tilde{s}_i^m(t)$, $\gamma_i^m(t) = \tilde{\gamma}_i^m(t)$ and $\eta_i^m(t) = \tilde{\eta}_i^m(t)$, for each type of VMs $m \in \{1, \ldots, M\}$.

C. Server Provisioning, Job scheduling and Dropping

After receiving results of the double auction (actual charges $\hat{b}_i^m(t)$, $\hat{s}_i^m(t)$, $\forall m \in \{1, \ldots, M\}$, and the actual numbers of traded VMs $\hat{\gamma}_i^m(t)$, $\hat{\eta}_i^m(t)$, $\forall m \in \{1, \ldots, M\}$, $\alpha_{ji}^{m_s}(t), \forall s \in \{1, \ldots, S\}, \forall j \in \{1, \ldots, F\}$), cloud i schedules its jobs on its local servers and (potentially) purchased VMs from other clouds, decides job drops and the number of active servers to provision, by solving optimization problems (19) and (20).

1) Server provisioning: We start with deriving $n_i^m(t)$, $\forall m \in \{1, \ldots, M\}$, by assuming known values of $\hat{s}_i^m(t)$, $\hat{\eta}_i^m(t)$, $\hat{b}_i^m(t)$, $\hat{\gamma}_i^m(t)$, $\alpha_{ij}^m(t)$ and $\mu_{ij}^s(t)$ (we will present the value of $n_i^m(t)$ in terms of these variables). In this case, problem (19) is equivalent to the following minimization problem:

$$\begin{aligned} \min \quad & V\beta_i(t) \sum_{m=1}^{M} n_i^m(t) \\ s.t. \quad & \text{Constraints (5), (6) and (9).} \end{aligned}$$

Since $V\beta_i(t) \geq 0$, the best strategy is to assign the minimal feasible value to $n_i^m(t)$, $\forall m \in \{1, \ldots, M\}$, that satisfies constraints (5) and (9), which can be combined into

$$\sum_{s \in \{1,\ldots,S\}, m_s = m} \mu_{ii}^s(t) \cdot g_s + \sum_{j \neq i} \alpha_{ji}^m(t) \leq C_i^m n_i^m(t).$$

Hence, the optimal number of activated servers at cloud i to provision type-m VM can be calculated as

$$n_i^m(t) = (\sum_{s \in \{1,\ldots,S\}, m_s = m} \mu_{ii}^s(t) \cdot g_s + \sum_{j \neq i} \alpha_{ji}^m(t))/C_i^m. \tag{27}$$

These many servers can provide enough type-m VMs for serving local jobs and selling to other clouds.

2) Job scheduling: We now derive $\mu_{ij}^s(t), \forall j \in \{1, \ldots, F\}$, $s \in \{1, \ldots, S\}$, by assuming known values of $\hat{s}_i^m(t)$, $\hat{\eta}_i^m(t)$, $\hat{b}_i^m(t)$, $\hat{\gamma}_i^m(t)$ and $\alpha_{ij}^m(t)$, with $n_i^m(t)$ given in Eqn. (27). Problem (19) is equivalent to the following maximization problem:

$$\begin{aligned} \max \quad & \sum_{s=1}^{S} \sum_{j=1}^{F} \mu_{ij}^s(t)[Q_i^s(t) + Z_i^s(t)] \\ & - V\beta_i \sum_{s \in \{1,\ldots,S\}, m_s = m} \mu_{ii}^s(t) \cdot \frac{g_s}{C_i^m} \\ s.t. \quad & \text{Constraints (5), (6) and (9).} \end{aligned}$$

This is a maximum-weight scheduling problem, with $Q_i^s(t) + Z_i^s(t)$ as the per-job scheduling weight for each $\mu_{ij}^s(t)$

[5] It may appear counter-intuitive that a cloud is willing to buy all type-m VMs in the federation, regardless of its number of unscheduled jobs requiring type-m VMs, *i.e.*, $\sum_{s \in \{1,\ldots,S\}, m_s = m} Q_i^s(t)$. Interestingly, our proof in Sec. VII shows that bidding so in each time slot can achieve a time-averaged profit over the long run that approximates the offline optimum, and our simulation in Sec. VIII shows that it performs better as compared to a bidding strategy that asks for the exact number of VMs to serve the unscheduled jobs.

$(j \neq i)$ and $Q_i^s(t)+Z_i^s(t)-\frac{V\beta_i(t)g_s}{C_i^m}$ as the per-job scheduling weight for each $\mu_{ii}^s(t)$. There are two cases:

▷ $j = i$: In this case, by combining constraints (5), (6) and (9), we have

$$\sum_{s:m_s=m,s\in\{1,\ldots,S\}} g_s\mu_{ii}^s(t) \leq C_i^m N_i^m - \sum_{j\neq i}\alpha_{ji}^m(t).$$

Based on the above maximum-weight problem, we know that the best strategy is to assign all the remaining type-m_s VMs in cloud i, $C_i^{m_s}n_i^{m_s}(t)-\sum_{j\neq i}\alpha_{ji}^{m_s}(t)$ (the maximum number of on-premise type-m_s VMs minus those sold to other clouds), to serve its own jobs of service type $s_{m_s}^*$ with the largest per-VM scheduling weight $\frac{Q_i^s(t)+Z_i^s(t)}{g_s}-\frac{V\beta_i(t)}{C_i^{m_s}}$ if it is positive (equivalently, the largest $\frac{Q_i^s(t)+Z_i^s(t)}{g_s}$ if $\frac{Q_i^s(t)+Z_i^s(t)}{g_s} > \frac{V\beta_i(t)}{C_i^{m_s}}$), among all job types requiring type-m_s VMs. Otherwise, cloud i does not serve any jobs using its own servers in t. Hence, we derive the optimal number of cloud i's type-s jobs scheduled to run on the cloud's local servers as

$$\mu_{ii}^s(t) = \begin{cases} \frac{C_i^{m_s}\cdot N_i^{m_s}-\sum_{j\neq i}\alpha_{ji}^{m_s}(t)}{g_s} & \text{if } \frac{Q_i^s(t)+Z_i^s(t)}{\cdot g_s} > \frac{V\beta_i(t)}{C_i^{m_s}} \\ & \text{and } \ s = s_{m_s}^* \\ 0 & \text{Otherwise} \end{cases}. \tag{28}$$

▷ $j \neq i$: $\mu_{ij}^s(t)$ can be directly derived by $\alpha_{ij}^{m_s}(t)$, which is the number of type-m_s VMs cloud i purchased from cloud j (constraint (5) is satisfied by our server provisioning decision in Eqn. (27), and constraint (6) is met by Eqn. (28) and (27)), based on constraint (9). Similar to the previous case, we know that the best strategy is to assign all the type-m_s VMs purchased, $\alpha_{ij}^{m_s}(t)$, to serve jobs of service type $s_{m_s}^*$ with the largest per-VM scheduling weight $\frac{Q_i^s(t)+Z_i^s(t)}{g_s}$, as defined in Eqn. (23) and (24). Hence, we derive the optimal solution to the number of type-s jobs to run at cloud $j(\neq i)$ as

$$\mu_{ij}^s(t) = \begin{cases} \alpha_{ij}^{m_s}(t)/g_s & \text{if } s = s_{m_s}^* \\ 0 & \text{Otherwise} \end{cases}. \tag{29}$$

3) Job dropping: Problem (20) is a maximum-weight problem with weight $Q_i^s(t)+Z_i^s(t)-V\cdot\xi_i^s$ for job-dropping decision variable $D_i^s(t)$, $\forall s \in \{1,\ldots,S\}$, in the objective function. If the weight $Q_i^s(t)+Z_i^s(t)-V\cdot\xi_i^s > 0$ (*i.e.*, if the level of urgency for scheduling type-s jobs $Q_i^s(t)+Z_i^s(t)$ exceeds the weighted job-drop penalty $V\cdot\xi_i^s$), type-s jobs in queue Q_i^s should be dropped at the maximum rate, *i.e.*, $D_i^s(t) = D_i^{s(max)}$, in order to maximize the objective function value; otherwise, there is no drop, *i.e.*, $D_i^s(t) = 0$. Therefore, the optimal number of type-s jobs dropped by cloud i in t is

$$D_i^s(t) = \begin{cases} D_i^{s(max)} & \text{if } Q_i^s(t)+Z_i^s(t) > V\cdot\xi_i^s \\ 0 & \text{Otherwise} \end{cases}. \tag{30}$$

In the above results, we note that the derived job scheduling and drop numbers do not need to be bounded by the number of unscheduled jobs in the corresponding job queue, *i.e.*, $\mu_{ij}^s(t)$ and $D_i^s(t)$ are not required to be bounded by $Q_i^s(t)$ according to Eqn. (2). Nevertheless, the actual number of jobs to schedule/drop when running the algorithm, is upper bounded by the minimum between the length of the job queue and the maximum drop rate $D_i^{s(max)}$.

D. The Dynamic Algorithm

Alg. 1 summarizes the dynamic algorithm for each cloud to carry out in each time slot, in order to maximize its time-averaged profit over the long run.

Algorithm 1 Dynamic Profit Maximization Algorithm at cloud i in Time Slot t

Input: $r_i^s(t)$, $Q_i^s(t)$, $Z_i^s(t)$, g_s, m_s, ξ_i^s, C_i^m, N_i^m and $\beta_i(t)$, $\forall s \in \{1,\ldots,S\}$.
Output: $b_i^m(t)$, $s_i^m(t)$, $\gamma_i^m(t)$, $\eta_i^m(t)$, $D_i^s(t)$, $\mu_{ij}^s(t)$ and $n_i^m(t)$, $\forall m \in \{1,\ldots,M\}, s \in \{1,\ldots,S\}, j \in \{1,\ldots,F\}$.

1: **VM valuation and bid**: Decide $b_i^m(t)$, $s_i^m(t)$, $\gamma_i^m(t)$ and $\eta_i^m(t)$ with Eqn. (21)-(26);
2: **Server provisioning, job scheduling and dropping**: Decide $\mu_{ij}^s(t)$, $D_i^s(t)$ and $n_i^m(t)$ with Eqn. (29), (28), (30) and (27);
3: Update $Q_i^s(t)$ and $Z_i^s(t)$ with Eqn. (2) and (4).

We analyze the computation and communication complexities of Alg. 1 as follows.

Computation complexity: We study the computation complexity for each algorithm module respectively.

▷ *VM valuation and bid*: The algorithm should first calculate the value of s_m^* for each VM type $m \in \{1,\ldots,M\}$ with Eqn. (23) by comparing the weights $W_i^{s'}(t)$ among different types of jobs. In fact, the weight for each job type $s \in \{1,\ldots,S\}$ is only evaluated once since it is only involved in the calculation of s_m^* where $m = m_s$. Hence, the computation overhead to find s_m^*, $\forall m \in \{1,\ldots,M\}$, is $O(S)$. Based on the value of s_m^*, the buy/sell bids of type-m VMs can be decided by Eqn. (21)-(26) in constant time. For all M VM types, the computation overhead is $O(M)$. Hence, the overall computation complexity for this algorithm module is $O(S+M)$.

▷ *Server provisioning, job scheduling and dropping*: With s_m^*, $\forall m \in \{1,\ldots,M\}$, calculated in the above algorithm module, we can directly know the value of $s_{m_s}^*$, $\forall s \in \{1,\ldots,S\}$. Then, the job scheduling decision $\mu_{ij}^s(t)$ for job type s can be made in constant time based on Eqn. (29) and (28). For all S job types, the computation overhead is $O(S)$.

The server provisioning decisions can be found in constant time based on the job scheduling decisions and the auction results, according to Eqn. (27) for type-m VMs. For all M VM types, the computation overhead is $O(M)$.

Job dropping is also decided in constant time for type-s jobs based on Eqn. (30). For all S job types, the computation complexity is $O(S)$.

▷ *Queue update*: For each job type s, the job queue $Q_i^s(t)$ and virtual queue $Z_i^s(t)$ can be updated in constant time based on Eqn. (2) and (4). Hence, for all S job types, the computation overhead is $O(S)$.

In summary, the computation complexity of Alg. 1 is $O(S+M)$.

Communication complexity: The input to Alg. 1 is mostly derived from local information. There is no direct information exchange among individual clouds. The only communication overhead is incurred when a cloud sends its VM bids to the auctioneer and receives the auction results for each VM type. Since there are M VM types, the communication complexity is $O(M)$ for each cloud.

V. Double Auction Mechanism

We next design a double auction mechanism for inter-cloud VM trading, which not only is truthful, individual rational and ex-post budget balanced, but also can enable satisfactory social welfare (Theorems 2-4 and 8, Sec. VII).

The true values of buy and sell bids at each participating cloud (Eqn. (21)-(26)) are not related to the detailed auction mechanism. The true values of the maximum numbers of VMs a cloud is willing to trade ($\hat{\gamma}_i^m(t)$ and $\hat{\eta}_i^m(t)$ in (25) and (26)) are time-independent constants determined by system parameters C_i^m and N_i^m. These parameters, and thus $\hat{\gamma}_i^m(t)$ and $\hat{\eta}_i^m(t)$, are easily known to other clouds, and hence it is not meaningful for a buyer/seller to bid otherwise. We correspondingly design a double auction where $\gamma_i^m(t)$ in each buy-bid is fixed to the value in (25) and $\eta_i^m(t)$ in each sell-bid is always the value in (26), while the buy/sell prices, $b_i^m(t)$'s and $s_i^m(t)$'s, can be decided by the respective buyers/sellers.

The following mechanism is carried out by the auctioneer at the beginning of each time slot t, to decide the actual trading price and number for each type of VMs $m \in \{1, \ldots, M\}$.

1. Winner Determination: The auctioneer sorts all received buy-bids for type-m VMs in descending order in the buy prices. Let $\theta_j^m(t)$ be the j^{th} highest. Two buy-bids with the *largest* and *second largest* prices, $\theta_1^m(t)$, $\theta_2^m(t)$, are identified (ties broken arbitrarily). The sell-bids for type-m VMs are sorted in ascending order in the sell prices. Let $\vartheta_j^m(t)$ be the j^{th} lowest, with $L_j^m(t)$ as the corresponding maximum number of VMs to sell[6], such that $\vartheta_1^m(t) \leq \vartheta_2^m(t) \leq \ldots \leq \vartheta_N^m(t)$. Let j' be the critical index in the sorted sequence of sell-bids, such that $\vartheta_{j'}^m(t)$ is the largest sell price not exceeding $\theta_2^m(t)$, *i.e.*,

$$\vartheta_{j'}^m(t) \leq \theta_2^m(t), \text{ and } \vartheta_{j'+1}^m(t) > \theta_2^m(t). \tag{31}$$

If there are at least two sell-bids $\vartheta_1^m(t)$ and $\vartheta_2^m(t)$ no higher than the second largest buy price $\theta_2^m(t)$, the highest buy-bid $\theta_1^m(t)$ wins, and the sell-bids with the lowest to the $(j'-1)^{\text{th}}$ lowest sell prices ($\vartheta_j^m(t) \leq \vartheta_{j'}^m(t)$, not including j') win. Otherwise, no buy/sell bid wins.

2. Pricing and Allocation: It is a NP-hard problem to clear the double auction market with discriminatory prices [24]. We apply a uniform clearing price to winning buy/sell bids of type-m VMs, as follows.

▷ The price charged to each buyer cloud i of type-m VMs is

$$\hat{b}_i^m(t) = \begin{cases} \theta_2^m(t) & \text{if bid } b_i^m(t) \text{ wins,} \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

▷ The price paid to each seller cloud i of type-m VMs is

$$\hat{s}_i^m(t) = \begin{cases} \vartheta_{j'}^m(t) & \text{if bid } s_i^m(t) \text{ wins,} \\ 0 & \text{otherwise.} \end{cases} \tag{33}$$

▷ The number of type-m VMs bought by cloud i is

$$\hat{\gamma}_i^m(t) = \begin{cases} \sum_{j=1}^{j'-1} L_j^m(t) & \text{if bid } b_i^m(t) \text{ wins,} \\ 0 & \text{otherwise.} \end{cases} \tag{34}$$

▷ The number of type-m VMs sold by cloud i is

$$\hat{\eta}_i^m(t) = \begin{cases} \eta_i^m(t) & \text{if bid } s_i^m(t) \text{ wins,} \\ 0 & \text{otherwise.} \end{cases} \tag{35}$$

[6]For example, if the j^{th} lowest bid comes from cloud i, we have that $L_j^m(t) = \eta_i^m(t)$ while $\eta_i^m(t) = \tilde{\eta}_i^m(t) = C_i^m \cdot N_i^m$ (See Eqn. (26)).

▷ The number of type-m VMs sold from cloud j to cloud i is

$$\alpha_{ij}^m(t) = \begin{cases} \eta_j^m(t) & \text{if bids } b_i^m(t) \text{ and } s_j^m(t) \text{ win,} \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

For example, consider a federation of 4 clouds with buy and sell prices bid in Table IV, each seeking to buy/sell one VM. Clouds 2 and 3 bid the two largest buy prices \$20 and \$15, which are higher than sell prices from clouds 1 and 4. Hence the buyer cloud 2 and the seller cloud 4 win, while the clearing buy and sell prices are \$15 and \$13, respectively.

TABLE IV
Double auction bids: an illustrative example

	Cloud 1	Cloud 2	Cloud 3	Cloud 4
Buy-bid	\$10	\$20	\$15	\$8
Sell-bid	\$13	\$22	\$16	\$9

VI. Dynamic Social-Welfare Maximization Algorithm: a Benchmark

We also present a dynamic algorithm that maximizes the time-averaged social welfare in the federation (optimization problem (16)) based on the Lyapunov optimization framework. This algorithm is used as a benchmark to examine the efficiency of Alg. 1 in social welfare.

Similar to the derivation of Alg. 1, we first derive a one-shot optimization problem for the federation to solve based on the *drift-plus-penalty* framework of Lyapunov optimization, and then derive the dynamic benchmark algorithm to solve it optimally in each time slot. Detailed derivation of the benchmark algorithm is included in [23].

1) Server provisioning: The optimal number of activated servers at cloud i to provision type-m VM is

$$n_i^m(t) = [\sum_{j=1}^{F} \sum_{s \in \{1,\ldots,S\}, m_s = m} \mu_{ji}^s(t) \cdot g_s]/C_i^m. \tag{37}$$

2) Job scheduling: The optimal solution to the number of type-s jobs of cloud i to run at cloud j is

$$\mu_{ij}^s(t) = \begin{cases} C_j^{m_s} \cdot N_j^{m_s}/g_s & \text{if } \frac{Q_i^s(t)+Z_i^s(t)}{g_s} > \frac{V\beta_i(t)}{C_i^{m_s}} \\ & \text{and } < i, s > = < \acute{i}_m, \acute{s}_m >, \\ 0 & \text{Otherwise,} \end{cases} \tag{38}$$

where

$$< \acute{i}_m, \acute{s}_m > = arg \max_{i \in \{1,\ldots,F\}, s \in \{1,\ldots,S\}, m_s = m} \{W_i^s(t)\}, \tag{39}$$

and $W_i^s(t)$ is the weight defined in Eqn. (24).

3) Job dropping: The optimal number of type-s jobs dropped by cloud i in t is

$$D_i^s(t) = \begin{cases} D_s^{(max)} & \text{if } Q_i^s(t) + Z_i^s(t) > V \cdot \xi_i^s \\ 0 & \text{Otherwise.} \end{cases} \tag{40}$$

Alg. 2 summarizes the dynamic algorithm for the federation to carry out (*e.g.*, on a centralized controller) in each time slot, in order to maximize its time-averaged social welfare over the long run.

Algorithm 2 Dynamic Social Welfare Maximization Algorithm in Time Slot t

Input: $r_i^s(t)$, $Q_i^s(t)$, $Z_i^s(t)$, g_s, m_s, ξ_i^s, C_i^m, N_i^m and $\beta_i(t)$, $\forall i \in \{1, \ldots, F\}, s \in \{1, \ldots, S\}$.

Output: $D_i^s(t)$, $\mu_{ij}^s(t)$ and $n_i^m(t)$, $\forall i \in \{1, \ldots, F\}, m \in \{1, \ldots, M\}, s \in \{1, \ldots, S\}$.

1: **Job scheduling and server provisioning**: Decide $\mu_{ij}^s(t)$ and $n_i^m(t)$ with Eqn. (38) and (37);
2: **Job dropping**: Decide $D_i^s(t)$ with Eqn. (40);
3: Update $Q_i^s(t)$ and $Z_i^s(t)$ with Eqn. (2) and (4).

VII. Performance Analysis

We next analyze the performance guarantee provided by our dynamic individual-profit maximization algorithm and the double auction mechanism.

A. Properties of the Double Auction Mechanism

Theorem 1 (True Valuation): The VM valuations on buy-bids, *i.e.*, Eqn. (21) and (25), and sell-bids, *i.e.*, Eqn. (22) and (26), are true values.

This theorem is proved based on the definition of the true values and the optimization problem (18) solved in each time slot by each cloud in [23].

Theorem 2 (Truthfulness): Bidding truthfully is the dominant strategy of each cloud in the double auction in Sec. V, *i.e.*, no cloud can achieve a higher profit in (18) by bidding with values other than its true values of the buy and sell bids, in Eqn. (21)(25)(22)(26).

We prove this theorem by contradiction and show that, in all cases, no cloud can do better with problem (18) by bidding untruthfully. Details are in [23]. The truthfulness can simplify the bidders' strategies when proposing their bids and make the auction mechanism strategy-proof.

Theorem 3 (Individual Rationality): No winning buyer pays more than its buy-bid price, and no winning seller is paid less than its sell-bid price, *i.e.*, $\hat{b}_i^m(t) \leq b_i^m(t)$ and $\hat{s}_i^m(t) \geq s_i^m(t),\ \forall i \in \{1,\ldots,F\}, m \in \{1,\ldots,M\}$.

This theorem can be proved based on the winner determination and pricing schemes in our auction mechanism, with details in [23]. Given that the buy-bid (sell-bid) price is the true value of the buyer (seller), this theorem implies that a cloud can receive a non-negative profit gain, if it successfully sells or buys VMs. The rationale is as follows.

The true value at the buyer (seller) equals the respective cloud's evaluation of the VMs in order to maximize its long-term profit when the cloud's VMs are only utilized to process its own jobs (*i.e.*, operating alone). If the cloud participates in the VM trading and is charged at its true value, the cloud receives zero gain from VM-trading (or equivalently, achieves the same utility as if it operates alone). Hence, we conclude that *a cloud's profit obtained in a federation with potential VM trades with others, is always no lower than that obtained when operating alone,* since the VMs in our framework will be bought (sold) at prices no higher (lower) than their true values. The individual rationality provides incentives for individual clouds to participate in the auction for inter-cloud VM-trading.

Theorem 4 (Ex-post Budget Balance): At the auctioneer, the total payment collected from the buyers is no smaller than the overall price paid to the sellers, *i.e.*,

$$\sum_{i=1}^{F}[\hat{b}_i^m(t)\cdot\hat{\gamma}_i^m(t) - \hat{s}_i^m(t)\cdot\hat{\eta}_i^m(t)] \geq 0,\ \forall m \in \{1,\ldots,M\}.$$

This theorem is proved based on Eqn. (31) - (33), with details in [23]. The ex-post budget balance property guarantees the willingness of the auctioneer to hold the auction since it has no need to make any payment for the trading and can make a non-negative gain.

B. SLA Guarantee

Lemma 1: Let $Q_i^{s(max)} = V\xi_i^s + R_i^s$ and $Z_i^{s(max)} = V\xi_i^s + \epsilon_s$. If $D_i^{s(max)} \geq \max\{R_i^s, \epsilon_s\}$, each job queue $Q_i^s(t)$ and each virtual queue $Z_i^s(t)$ are upper-bounded by $Q_i^{s(max)}$ and $Z_i^{s(max)}$, respectively, in $t \in \{0,\ldots,T-1\}$, $\forall i \in \{1,\ldots,F\}, s \in \{1,\ldots,S\}$.

This lemma can be proved by analyzing the job drop decision in (30) and the queue updates in (2)(4). The condition $D_i^{s(max)} \geq \max\{R_i^s, \epsilon_s\}$ ensures that, when the queue lengths grow to satisfy the job drop condition, any further increase on the queues, *e.g.*, R_i^s and ϵ_s, can be balanced by dropping enough number of jobs at the rate of $D_i^{s(max)}$. Detailed proof is included in [23]. Since each queue can be strictly upper-bounded in each time slot, our algorithm can guarantee the queue stabilities and the network stability [22].

Theorem 5 (SLA Guarantee): Each job of type $s \in \{1,\ldots,S\}$ is either scheduled or dropped with Alg. 1 before its maximum response delay d_s, if we have $\epsilon_s = \frac{Q_i^{s(max)} + Z_i^{s(max)}}{d_s}$ by adjusting the value of V.

This theorem can be proved based on Lemma 1 and the ϵ-persistence queue techniques [18]. The condition on ϵ_s is to ensure that the queue lengths can grow to satisfy the job drop condition, *i.e.*, $Q_i^s + Z_i^s(t) > V\xi_i^s$, if some jobs remain unscheduled in the last d_s slots. Note that a cloud only drops jobs strategically, to balance the loss due to the job drop penalties and the gain in saving VMs for other jobs. For more details, please refer to [23].

C. Optimality of Individual Profit and Social Welfare

Theorem 6 (Individual Profit Optimality): Let Ω_i^* be the offline optimum of time-averaged profit of cloud $i \in \{1,\ldots,F\}$, obtained in a truthful, individual-rational, ex-post budget-balanced double auction, with complete information on its own job arrivals and prices in the entire time span $[0, T-1]$. The dynamic Algorithm 1 can achieve a time-averaged profit Ω_i for cloud i within a constant gap B_i/V to Ω_i^*, *i.e.*,

$$\Omega_i \geq \Omega_i^* - B_i/V,$$

where $V > 0$ and $B_i = \frac{1}{2}\sum_{s=1}^{S}[[\sum_{j=1}^{F} C_j^{m_s} N_j^{m_s}/g_s + D_i^{s(max)}]^2 + [R_i^s]^2 + [\epsilon_s]^2 + [D_i^{s(max)} + \sum_{j=1}^{F} C_j^{m_s} N_j^{m_s}/g_s]^2]$ is a constant.

The proof to this theorem is rooted in the Lyapunov optimization theory [22]. Detailed proof is included in [23]. The gap B_i/V can be close to zero by fixing ϵ_s and increasing V if the users can tolerate a longer maximum possible delay (To be discussed in Sec. VII.D).

Theorem 7 (Social Welfare Optimality of Alg. 2): Let Π^* be the offline optimum of the time-averaged social welfare in (16), obtained with full information of the federation over the entire time span $[0, T-1]$. The time-averaged social welfare achieved by all clouds by running Alg. 2, approaches the offline-optimal social welfare Π^*, by a constant gap B/V, *i.e.*,

$$\Pi \geq \Pi^* - B/V,$$

where $V > 0$ and $B = \sum_{i=1}^{F} B_i$. B_i is defined in Theorem 6, $\forall i \in \{1, \ldots, F\}$.

The proof to this theorem is also based on the Lyapunov optimization theory [22]. Detailed proof is included in [23]. Similar with Theorem 6, the gap B/V can be close to zero by fixing ϵ_s and increasing V if the users can tolerate a longer maximum possible delay (To be discussed in Sec. VII.D).

Theorem 8 (Asymptotic Optimality in Social Welfare of Alg. 1): Let Π^* be the offline optimum of the time-averaged social welfare in (16), obtained with full information of the federation over the entire time span $[0, T-1]$. Suppose all clouds are homogenous, *i.e.*, with the same number of servers (N_i^m) and the same maximum per-server VM provisioning (C_i^m) for each VM type m, with i.i.d. service prices, job arrivals and operational costs. When the number of clouds, F, grows, the sum of time-averaged profits achieved by all clouds by running Alg. 1 under the double auction mechanism in Sec. V, approaches the offline-optimal social welfare Π^*, by a constant gap B/V, *i.e.*,

$$\Pi \geq \Pi^* - B/V,$$

where $V > 0$ and $B = \sum_{i=1}^{F} B_i$. B_i is defined in Theorem 6, $\forall i \in \{1, \ldots, F\}$.

To prove the theorem, we demonstrate that, when the number of clouds goes to infinity, the one-shot social welfare obtained with Alg. 1 is the same as that achieved by the dynamic benchmark Alg. 2 in the same time slot. Details are in [23].

D. Tradeoff between Optimality and Maximum Tolerable Delay

With Theorems 6, 7 and 8, we can derive that the gap, B_i/V (B/V), between the achievable individual profit (social welfare) with our algorithm and its offline optimum can be made close to zero, when the value of V increases (recall that B_i and B are constants). However, the prerequisite for increasing the value of V is that the users can tolerate a longer maximum possible delay, due to Lemma 1 and Theorem 5 as follows. In order to meet the SLA requirements, Theorem 5 states that the constant ϵ_s should be equal to $\frac{Q_i^{s(max)} + Z_i^{s(max)}}{d_s}$. With Lemma 1, we know that the maximum queue lengths, $Q_i^{s(max)}$, $Y_i^{s(max)}$ and $Z_i^{s(max)}$, are proportional to V. Hence, if we need a larger V, the users should be able to accept a longer maximum tolerable delay, *i.e.*, d_s, in order to meet the condition in Theorem 5. In summary, there is a tradeoff between the achievable optimality and the maximum tolerable delay.

VIII. Performance Evaluation

A. Simulation Setup

We carry out trace-driven simulation studies based on Google cluster-usage data [25] [26], which record jobs submitted to the Google cluster, with information on their resource demands (CPU, RAM, etc.) and relative charges. We translate the data into concrete job arrival rates, resource types and prices, to drive our simulations as follows.

We consider 24 types of jobs ($< m_s, g_s, d_s >$), 6 VM types (m_s) combined from {*small, median, large*} *CPU* and {*small, large*} *Memory*, and two *SLA* levels (d_s), corresponding to a larger maximum tolerable response delay and a smaller maximum tolerable response delay at half of the former. Each job requires either 1 VM or 2 VMs concurrently (g_s).

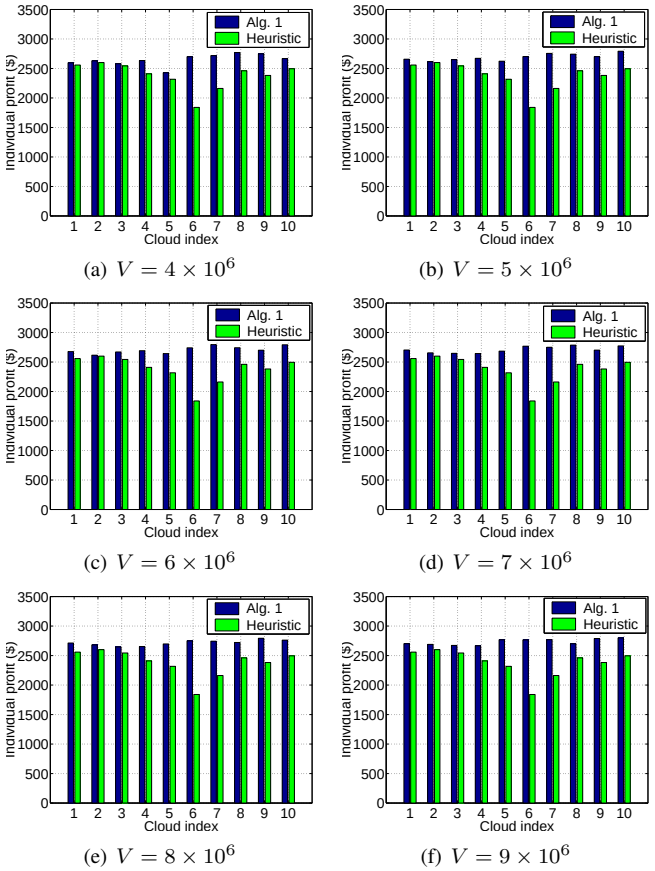
There are 10 clouds in the federation. One time slot is 1 hour. The number of servers in each cloud that provision VMs of each type ranges within $[800, 1000]$. Each server can provide 30 *small-memory* VMs or 10 *large-memory* VMs. The VM charge to the customer is decided by multiplying g_s by the relative VM price in the Google data, and then by the unit VM price in the range of [0.05, 0.08] $/h. The penalty for dropping a job is set to the maximum per-job VM charge in the system. Operational costs are set according to the electricity prices at 10 different geographic locations provided in [20], which vary on a hourly basis. Each server consumes power at 1 KW/h.

The number of job arrivals in each hour to the federation is set according to the cumulated job requests of each type submitted to the Google cluster during that hour, in the rough range of $[40000, 90000]$ requests per hour. We randomly assign each arrived job to one of the 10 clouds, following a heavy-tailed distribution. In operating the virtual queues, we set $\epsilon_s = 1000$ for jobs requiring low response delay, and $\epsilon_s = 500$ for those of long delays. The maximum number of job drops per hour is 1000 for all job types.

For comparison purposes, we also implement a simpler heuristic algorithm for each cloud to bid in the double auction and to schedules its jobs/servers: The cloud decides a value for each unscheduled job in a queue as the penalty to drop it if the next time slot is the deadline for scheduling, or the charged price upon its arrival otherwise. The true values of buy/sell prices for a type-m VM at this cloud are set to the same, as the largest average value of jobs in a queue, among all job queues requiring type-m VM(s). The quantity of VMs in a buy-bid is set to the number of unscheduled jobs in the queue with the largest average value as computed above. The quantity of VMs in a sell-bid is the overall number of VMs of the type that the cloud can provide. All VMs purchased via the auction are used to serve jobs from the queue with the maximum average value. A cloud maintains the minimum number of servers to support those jobs, and only drops a job when its maximum tolerable response delay is reached.

B. Individual Profit and Social Welfare

We compare the time-averaged profit achieved at each cloud with our dynamic algorithm in Alg. 1 and with the heuristic algorithm, after the system has been running for 2000 hours. Fig. 2 shows that our algorithm can achieve a higher profit than the heuristic, at each of the 10 clouds, when the value of V is no less than 4×10^6. The observation from Fig. 3 is that when V is larger, the individual profit with our algorithm is even better, since it is closer to the offline optimum.



(a) $V = 4 \times 10^6$ (b) $V = 5 \times 10^6$

(c) $V = 6 \times 10^6$ (d) $V = 7 \times 10^6$

(e) $V = 8 \times 10^6$ (f) $V = 9 \times 10^6$

Fig. 2. Comparisons of individual profit with different values of V.

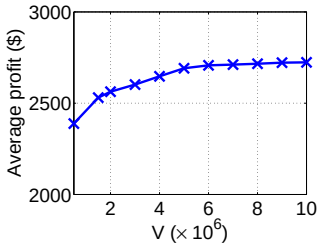


Fig. 3. Average profit with Alg. 1 and different values of V.

We next compare the social welfare achieved with Alg. 1, the heuristic, and the dynamic benchmark Alg. 2. Fig. 4 shows that social welfare achieved with Alg. 1 is mostly within 7.7% of that by the benchmark algorithm, even under our heterogenous settings. It outperforms the heuristic by 19.2%. The social welfare is larger at larger V's in cases of both Alg. 1 and the benchmark algorithm, verifying Theorems 6 and 8 in that they approach the respective offline optimum when V grows.

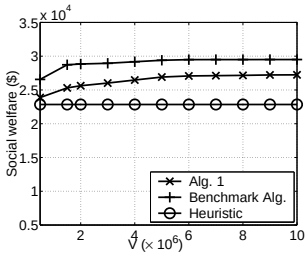
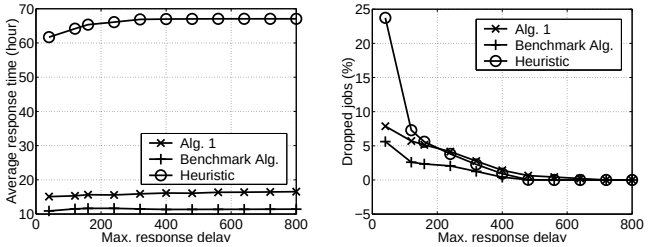


Fig. 4. Comparisons of social welfare.

C. Response Delay and Job Drop

We next investigate the scheduling delays experienced by jobs. In our system, a maximum tolerable response delay is set as the SLA objective for each type of jobs. Here, we study the average response delay actually experienced by the jobs, when the longer maximum tolerable response delay is set to different values. Fig. 6(a) shows that both Alg. 1 and the benchmark algorithm incur a low average response delay (well ahead of scheduling deadlines), as compared to that of the heuristic. The reasons are: i) the heuristic algorithm always greedily keeps jobs in queues for future scheduling until near the deadline; and ii) both Alg. 1 and the benchmark algorithm evaluate the scheduling urgency better than the heuristic does, such that jobs are tended to be served well before the deadlines.

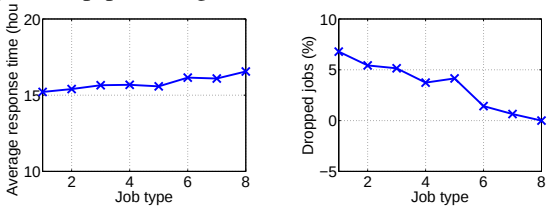


(a) Average response delay (b) Averaged job-drop percentage

Fig. 5. Comparisons of average job scheduling delay and drop percentage.

We also study the percentage of admitted jobs in the entire federation that are eventually dropped with the three algorithms. Fig. 6(b) reveals that the drop rate decreases quickly with the increase of the allowed maximum tolerable response delay, and Alg. 1 and the benchmark algorithm again outperform the heuristic, due to their well-designed scheduling strategies.

We next examine the average response delay and job-drop percentage for different job types with Alg. 1. The maximum response delay is set as 300 hours. Here, for ease of presentation, we only consider the jobs requiring small CPU resource. Hence, the memory demand is the bottleneck and will determine the volume of VMs each server can provide. Table V summarizes the examined job types. From Fig. 6, we see the trend that jobs with shorter maximum tolerable delay and/or lower resource demand, *i.e.*, smaller g_s and/or lower memory requirement, have shorter average response delays but higher drop percentages. It can be explained as follows: i) a shorter maximum tolerable delay makes the jobs more urgent for scheduling (thus a shorter average response delay), but also more likely to hit their deadlines (thus a higher drop percentage); and ii) a lower resource demand renders the jobs easier to serve (thus a shorter average response delay), however, also decreases the drop penalty for these jobs (thus a higher drop percentage).



(a) Average response delay (b) Averaged job-drop percentage

Fig. 6. Comparisons of average job scheduling delay and drop percentage for different job types with Alg. 1.

Job type	1	2	3	4	5	6	7	8
CPU	S	S	S	S	S	S	S	S
Memory	S	S	S	S	L	L	L	L
g_s	1	1	2	2	1	1	2	2
d_s	150	300	150	300	150	300	150	300

TABLE V
JOB TYPES EXAMINED IN FIG. 6.

IX. CONCLUSION

This paper investigates both individual-profit maximizing and social-welfare efficient strategies at individual selfish clouds in a cloud federation, in VM trades across cloud boundaries. We tailor a truthful, individual-rational, ex-post budget-balanced double auction as the inter-cloud trading mechanism, and design a dynamic algorithm for each cloud to decide the best VM valuation and bidding strategies, and to schedule job service/drop and server provisioning in the most economic fashion, under time-varying job arrivals and operational costs. The proposed algorithm can obtain a time-averaged profit for each cloud within a constant gap to its offline maximum, as well as a close-to-optimum social welfare in the entire federation, based on both solid theoretical analysis and trace-driven simulation studies under realistic setting. As future work, we are interested in broadening our investigations to front-end job pricing and competition for customers among the clouds, and the connection between front-end charging strategies and inter-cloud trading strategies in a cloud federation. In this work we assume that all jobs are served within one time slot after being scheduled. We will extend the study to more realistic scenarios where the time needed to complete a job can span multiple time slots.

ACKNOWLEDGEMENT

This research is supported in part by a grant from Hong Kong RGC under the contract HKU 718513.

REFERENCES

[1] B. Rochwerger, D. Breitgand, E. L. E, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galn, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 54, pp. 535 – 545, 2009.

[2] E. Elmroth and L. Larsson, "Interfaces for placement , migration, and monitoring of virtual machines in federated clouds." in *Proc. of IEEE Computer Society GCC'09*, 2009.

[3] 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 *Prof. of IEEE INFOCOM'10*, 2010.

[4] S. Ran, Y. He, and F. Xu, "Provably-efficient job scheduling for energy and fairness in geographically distributed data centers," in *Prof. of IEEE ICDCS'12*, 2012.

[5] R. Urgaonkar, U. Kozat, K. Igarashi, and M. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Prof. of IEEE/IFIP NOMS'10*, 2010.

[6] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data centers power reduction: A two time scale approach for delay tolerant workloads," in *Proc. of IEEE INFOCOM'12*, 2012.

[7] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. of USENIX NSDI'11*, 2011.

[8] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *Proc. of ACM SIGCOMM'12*, 2012.

[9] R. Buyya, D. Abramson, and J. Giddy., "Nimrod/g: An architecture of a resource management and scheduling system in a global computational grid," in *Proc. of HPC Asia'00*, 2000.

[10] X. Zhou and H. Zheng, "Trust: A general framework for truthful double spectrum auctions," in *Proc. of IEEE INFOCOM'09*, 2009.

[11] H. Xu, J. Jin, and B. Li, "A secondary market for spectrum," in *Proc. of IEEE INFOCOM'10, Mini Conference*, 2010.

[12] [Online]. Available: http://aws.amazon.com/ec2

[13] M. Mihailescu and Y. M. Teo, "Dynamic resource pricing on federated clouds," in *Proc. of IEEE/ACM CCGrid'10*, 2010.

[14] ——, "The impact of user rationality in federated clouds," in *Proc. of IEEE/ACM CCGrid'12*, 2012.

[15] E. R. Gomes, Q. B. Vo, and R. Kowalczyk, "Pure exchange markets for resource sharing in federated clouds," *Concurrency Computat.: Pract. Exper.*, vol. 24, pp. 977 – 991, 2012.

[16] [Online]. Available: http://www.linode.com/faq.cfm

[17] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, and F. C. M. Lau, "Dynamic pricing and profit maximization for the cloud with geo-distributed data centers." in *Proc. of IEEE INFOCOM'14*, 2014.

[18] M. J. Neely, "Opportunistic scheduling with worst case delay guarantees in single and multi-hop networks," in *Proc. of IEEE INFOCOM'11*, 2011.

[19] U. Hoelzle and L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines.* Morgan & Claypool, 2009.

[20] [Online]. Available: www.ferc.gov

[21] R. B. Myerson and M. A. Satterthwaite, "Efficient mechanisms for bilateral trading," *Journal of Economics Theory*, vol. 29, pp. 265–281, 1983.

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

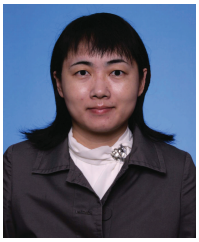
[23] H. Li, C. Wu, Z. Li, and F. C. M. Lau, "Virtual machine trading in a federation of clouds: Individual profit and social welfare maximization," http://arxiv.org/abs/1304.6491, Tech. Rep., 2013.

[24] T. SANDHOLM and S. SURI, "Market clearability," in *Proc. of IJI-CAI'01*, 2001.

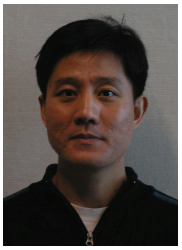
[25] J. Wilkes, "More Google cluster data," Nov. 2011, URL: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[26] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Tech. Rep., 2011, revised 2012.03.20. URL: http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

Hongxing Li received B.Sc. and M.Eng. degrees in 2005 and 2008 from the Department of Computer Science and Technology, Nanjing University, China, and his Ph.D. degree in 2012 from the Department of Computer Science, The University of Hong Kong, Hong Kong. His research interests include wireless networks and cloud computing.



Chuan Wu received her B.Eng. and M.Eng. degrees in 2000 and 2002 from Department of Computer Science and Technology, Tsinghua University, China, and her Ph.D. degree in 2008 from the Department of Electrical and Computer Engineering, University of Toronto, Canada. She is currently an associate professor in the Department of Computer Science, the University of Hong Kong, Hong Kong. Her research interests include measurement, modeling, and optimization of large-scale peer-to-peer systems and online/mobile social networks. She is a member of IEEE and ACM.



Zongpeng Li received his B.E. degree in Computer Science and Technology from Tsinghua University (Beijing) in 1999, his M.S. degree in Computer Science from University of Toronto in 2001, and his Ph.D. degree in Electrical and Computer Engineering from University of Toronto in 2005. Since August 2005, he has been with the Department of Computer Science in the University of Calgary. In 2011-2012, Zongpeng was a visitor at the Institute of Network Coding, Chinese University of Hong Kong. His research interests are in computer networks and network coding. Zongpeng was named an Edward S. Rogers Sr. Scholar in 2004, won the Alberta Ingenuity New Faculty Award in 2007, was nominated for the Alfred P. Sloan Research Fellow in 2007, and received the Best Paper Award at PAM 2008 and at HotPOST 2012.

Francis C.M. Lau received his PhD in computer science from the University of Waterloo. He is currently a professor in computer science in The University of Hong Kong. He is the editor-in-chief of the Journal of Interconnection Networks. His research interests include computer systems, networks, programming languages, and application of computing in arts.