Online Placement and Scaling of Geo-distributed Machine Learning Jobs via Volume-discounting Brokerage

Xiaotong Li, Ruiting Zhou*, *Member, IEEE,* Lei Jiao, *Member, IEEE,*
Chuan Wu, *Senior Member, IEEE*, Yuhang Deng and Zongpeng Li, *Senior Member, IEEE*

Abstract—Geo-distributed machine learning (ML) often uses large geo-dispersed data collections produced over time to train global models, without consolidating the data to a central site. In the parameter server architecture, "workers" and "parameter servers" for a geo-distributed ML job should be strategically deployed and adjusted on the fly, to allow easy access to the datasets and fast exchange of the model parameters at any time. Despite many cloud platforms now provide volume discounts to encourage the usage of their ML resources, different geo-distributed ML jobs that run in the clouds often rent cloud resources separately and respectively, thus rarely enjoying the benefit of discounts. We study an ML broker service that aggregates geo-distributed ML jobs into cloud data centers for volume discounts via dynamic online placement and scaling of workers and parameter servers in individual jobs for long-term cost minimization. To decide the number and the placement of workers and parameter servers, we propose an efficient online algorithm which firstly decomposes the online problem into a series of one-shot optimization problems solvable at each individual time slot by the technique of regularization, and afterwards round the fractional decisions to the integer ones via a carefully-designed dependent rounding method. We prove a parameterized-constant competitive ratio for our online algorithm as the theoretical performance analysis, and also conduct extensive simulation studies to exhibit its close-to-offline-optimum practical performance in realistic settings.

Index Terms—Geo-distributed Machine Learning; Online Placement; Volume Discount Brokerage

1 Introduction

Geo-distributed machine learning (ML) derives useful insights from large data collections continuously produced at dispersed locations with potentially time-varying volumes, without moving them to a central site. For example, e-commerce sites, including Amazon and Taobao, recommend products that are of particular interests to users by learning the user preference from the click-through data continuously collected all over the world [1], using ML techniques such as logistic regression; CometCloudCare ($\boldsymbol{C^3}$) [2] is a platform for training ML models with geo-distributed sensitive datasets, subject to location restrictions, privacy requirements, and data use agreement (DUA) guarantees.

The parameter server architecture [3][4] is widely used in distributed ML, where "workers" send parameter updates to one or multiple "parameter servers" (PSs), and the PSs maintain a global copy of the model and send the updated global parameters back to the workers. In a geo-distributed ML job, workers and PSs often reside at different geographic locations, *e.g.,* cloud data centers. To exploit data that are continuously produced for training, incremental learning is customarily employed [5][6][7], where new data are consecutively fed to the corresponding workers respectively, in order to extend the existing model's knowledge dynamically. As data volumes fluctuate across the spatial-temporal spectrum, the number of workers and their geographic placement in the ML job should be adjusted on the fly.

In practice, the owners of such ML jobs typically rent resources (*e.g.,* virtual machines or containers equipped with GPUs) from cloud data centers to run workers and PSs, and pay the cloud providers for the resources used. Many cloud platforms now provide "volume discounts" to encourage the usage of their (ML) resources: the more the resources are used and the longer the resources are used for, the lower the unit resource price becomes. For example, Rackspace offers a two-tier volume discount policy [8]; Amazon [9] and the Telecoms cloud [10] adopt a three-tier pricing strategy. For the ML job owner, it is commonly hard to decide for how long a ML job will run before completion (*e.g.,* till the model convergence); with time-varying training data generation, it is even more challenging to estimate the future resource consumption. In general, it is difficult to reserve or expect sufficient cloud resource usage for individual ML jobs to leverage the volume discount offers.

This paper proposes an ML broker service to aggregate resource demands from multiple geo-distributed ML jobs, and rent cloud resources on their behalf for volume discounts. The potential of such a broker service is eminent, in reducing the overall cost of running ML jobs and hence lowering each individual job's expenditure: (i) different cloud providers offer diverse prices and volume discounts for different resources across geo-locations, and lower price opportunities materialize only when individual jobs aggre-

- *X. Li, Y. Deng and Z. Li are with School of Computer Science, Wuhan University, Wuhan, China (e-mail: {xtlee, dyhshengji, zongpeng}@whu.edu.cn).*
- *R. Zhou is with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, China (e-mail: ruitingzhou@whu.edu.cn).*
- *L. Jiao is with the Department of Computer and Information Science, University of Oregon, Eugene, OR, USA (e-mail: jiao@cs.uoregon.edu).*
- *C. Wu is with the University of Hong Kong, Kowloon, Hong Kong, China (e-mail: cwu@cs.hku.hk).*

** Corresponding Author*

This work was supported in part by the NSFC Grants (61502504), the Technological Innovation Major Projects of Hubei Province (2017AAA125), the Science and Technology Program of Wuhan City (2018010401011288), WHU-Xiaomi AI Lab, Hong Kong RGC GRF HKU 17204715, 17225516, 17204619, C7036-15G (CRF) and C5026-18G (CRF).

gate into sets; (ii) ML jobs are typically resource-intensive, time-consuming, and costly, and a small reduction in unit resource price may lead to large economic savings for the ML job owners. Note that, while cloud brokerage has been an existing proliferating business, with approximately $\$4.5$ billion revenue in 2017 and $\$9.52$ billion expected by 2021 [11], the ML broker service is new, which differs from cloud brokers due to the ML jobs' unique requirements for large-volume data analysis and high interconnection bandwidth (to support repeated parameter exchanges), possessing great potential for accommodating the explosive growth of the ML-driven applications in the near future.

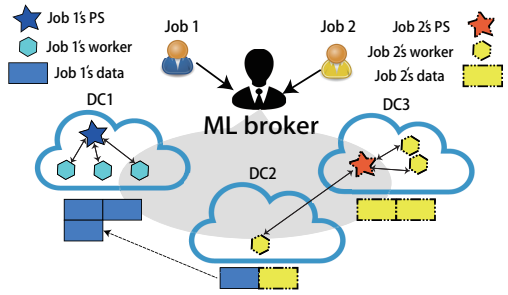


Fig. 1: Geo-distributed ML with the broker service.

To effectively operate an ML broker service, a fundamental problem is, *given the many ML jobs with time-varying volumes of training data at different geo-locations, how the broker should efficiently decide the number of workers and PSs to place at each location for each job at each time, such that the overall cost of running these jobs is minimized while the training data for each job are moved/distributed and processed.*

At the beginning of each time slot, the ML broker aggregates resource demands from all the newly arrived geo-distributed machine learning jobs, and observes the volumes of data generated at each data center. By taking the combination of resource demands, data distribution and volume discount of each type of resource into consideration, ML broker makes decisions on how to transfer training data and how to deploy workers and parameter servers at each location. An indicative illustration of our target system is shown in Fig. 1, where, through the ML broker, job 1's workers and PS are placed at data center 1, and job 2's workers are placed at data centers 2 and 3, with job 1's training data that are generated at data center 2 moved to data center 1. Note that we focus on the common parameter server architecture, and the specific instantiation of different machine learning models within this architecture is out of the scope of this paper.

We model and formulate the ML broker service's dynamic worker/PS placement problem as a mixed integer linear program (MILP). The MILP includes a complex set of variables indicating the decisions on the worker/PS placement, the amount of data to be copied into each data center for the worker's processing, and the number of inter-data-center worker-PS connections, all contributing to the cost of the ML jobs. To compute the decisions on the fly for the long-term cost minimization with guaranteed performance, we propose an efficient online algorithm based on regularization and dependent rounding techniques. We summarize our technical contributions as follows.

First, we analyze the detailed cost structure for running geo-distributed ML jobs and capture it in the broker's cost minimization problem. Particularly, via an auxiliary variable technique, we convert our original cost minimization problem with the non-linear, or in other words, the piecewise linear, volume-discounting price functions, which is hard to optimize, into an equivalent reformulation with only easier linear objective functions and additional constraints. To "reuse" existing workers/PSs over time as much as possible and to avoid re-deploying them repeatedly, we model the deployment cost as the Rectified Linear Unit [12] function of the deployment decisions in successive time slots.

Second, we relax the broker's MILP problem and design an online algorithm to compute the fractional decisions at each individual time slot by exploiting a regularization-based method. To overcome the challenge of the time-coupling deployment decisions, our algorithm decouples the relaxed problem into a series of convex sub-problems solvable at each corresponding time slot via substituting the deployment cost in the objective function by a carefully-designed, dedicated, convex term, and uses the solutions to this series of sub-problems as the solution to our original problem. Our approach achieves a provable parameterized competitive ratio, compared against the offline optimal solution that knows all the dynamic, online inputs in advance.

Third, we design a rounding algorithm to recover the integer decisions at each time slot from the fractional ones that are produced by our online algorithm. Our rounding algorithm repeatedly chooses a pair of fractional decisions and attempts to round them up and down, respectively, in a compensative manner in order to ensure that the sum of them remains largely the same as before so that the related constraints are not violated after rounding. Afterwards, taking the rounded integer decisions back into the problem and fixing them, we compute and update the rest of the variables. Our rounding approach achieves a provable integrality gap, compared against the fractional decisions before rounding. Joining our fractional online algorithm and dependent rounding algorithm, our complete online algorithm, which we name $mlBroker$, guarantees a small overall competitive ratio even in the worst cases.

Last, we conduct simulation studies to compare our online algorithm to offline optimal solutions and other representative alternatives under realistic settings, and reveal its close-to-offline-optimal performance in practice. Our online fractional algorithm achieves an empirical competitive ratio of less than 1.25, compared to the fractional offline optimum; our dependent rounding algorithm achieves a multiplicative integrality loss of less than $2.5\times$, and behaves consistently better than other baseline rounding algorithms. We find that, overall, our $mlBroker$ algorithm produces a competitive ratio of up to 3, and results in $20\%-50\%$ less total cost than a state-of-the-art algorithm and multiple well-adopted ML job placement/scaling benchmark methods. We also show $mlBroker$'s good performance in real experiments. Even with much smaller scale of data volume and fewer DCs due to economic constraints, our algorithm still obtains minimal cost, compared to other algorithms.

The rest of this paper is structured as follows. We review the related literature in Sec. 2. We define all our models and problem formulations in Sec. 3. We present the design and

the analysis of our online fractional algorithm in Sec. 4, as well as the design and the analysis of our dependent rounding algorithm in Sec. 5. Afterwards, Sec. 6 conducts both the simulations and real experiments to verify $mlBroker$'s good performance, and Sec. 7 concludes the paper.

2 Related Work

Distributed ML Systems. Cano *et al.* [13] propose a geo-distributed machine learning model to cope with the scarce and costly cross-DSs bandwidth and privacy constraints on user data. Gaia [14] is a geo-distributed ML system that eliminates insignificant communication between data centers while ensuring convergence of ML algorithms. Vulimiri *et al.* [15] propose a solution to deal with Wide-Area Big Data problem and conduct queries and analytics on geo-distributed data. Konecny *et al.* [16] design online algorithms for federated learning. A centralized model is trained with training data remains distributed among large numbers of users. Different from the above literature, we do not focus on the optimization in distributed ML algorithms to improve training speed, we aim to cost minimization problem which focuses on geo-distributed ML job placement. Wang *et al.* [17] focus on gradient-descent based distributed learning, and propose a control algorithm to determine the best trade-off between local update and global parameter aggregation, to minimize the loss function under a given resource budget. Chen *et al.* [18] propose a Bi-layered Parallel Training (BPT-CNN) architecture in distributed computing environments, which consists of two layers to address data partitions, communications and training speed acceleration separately. A Parallel Random Forest (PRF) algorithm is proposed in [19] for big data on the Apache Spark platform based on data parallel and task parallel optimization. They utilize vertical data-partitioning and data-multiplexing method to reduce data volume and communication cost. They also carry out a dual parallel approach and invoke different task scheduler according to task Directed Acyclic Graph to realize task parallel optimization. Chen *et al.* [20] propose a Distributed Intelligent Video Surveillance (DIVS) system in an edge computing environment to address the problems of parallel training, model synchronization and workload balancing. They design a model parameter updating method and a dynamic data migration approach to realize task parallel and model parallel in a distributed EC environment. SpGEMM kernels [21] are designed to realize high speed parallel computations. A multi-level parallelism design, optimization strategies and a performance-aware model for SpGEMM are combined together to meet the challenge of the high-speed computing of large-scale data sets on the Sunway. Xu *et al.* [22] mainly focus on how to properly implement joint request mapping and response routing to improve the performance of the entire distributed cloud platform, including delay, bandwidth and power consumption. They propose a general distributed algorithm based on ADMM to solve the large scale data optimization problem. Different from the above literature, we do not focus on the optimization in distributed ML algorithms to improve training speed, we aim to cost minimization problem which focuses on geo-distributed ML job placement.

ML job scheduling and placement. Existing efforts have been focusing on individual jobs or jobs in one data center. Xu *et al.* [23] focus on the cost minimization problem of big data analytics on geo-distributed data centers, and propose a Reinforcement Learning (RL) based job scheduling algorithm by combining RL with neural network (NN). They also leverage similar idea in [24] and propose a robust blockchain-based decentralized resource management framework to deal with the energy-aware resource management problem in cloud. The authors in [25] introduce many solutions to deal with the optimization in DCNs when considering the deployment of geo-distributed data centers and data-intensive applications. Dolphin [26] is an elastic ML framework which identifies the optimal number of works for each ML job. Mirhoseini *et al.* [27] propose an adaptive method to optimize device placement for TensorFlow graphs on different types of devices such as GPUs and CPUs. Gao *et al.* [28] model placement of a deep neural network on devices as a Markov decision process, and propose a reinforcement learning algorithm to minimize the time of training. Bao *et al.* [4] study online job scheduling in one ML cluster. Peng *et al.* [29] develop a job scheduler, Optimus, for deep learning clusters. Different from the above literature, our model assumes a fixed execution window without considering the scheduling dimension, but adjust the placement in each time slot to minimize the cost. Thus we can say we are the first to design a cost-minimization algorithm for ML brokers to optimize geo-distributed job placement. Moreover, our model also consider how to aggregate ML jobs to take advantage of volume-discounting pricing policy to reduce the resource renting cost.

In addition, we consider incremental learning jobs, in which newly generated data is continuously used to further train the model during each period. It does not require access to historical data used to train the existing model, and preserves previously acquired knowledge without the effect of catastrophic forgetting [5], [6] and [7].

Volume Discount and Cloud Brokerage Services. Volume-discounting based service is widely used by many cloud resource providers such as Rackspace [8], Amazon [9] and Telecoms cloud [10]. Zheng *et al.* [30] consider aggregating jobs to take the advantage of volume discount based on duration. Since ML jobs' running time can not be accurately predicted in prior, it is hard to adopt duration-based volume discount, but we can use usage-based discount strategy to explore better performance. Wang *et.al* [31] used a tiered usage-based discount policy for different kinds of VMs to attract cloud users to rent resources as a coalition formation game. Wang *et al.* [11] propose an offline resource scaling policy based on usage-based volume discount. However, due to the property of incremental learning, all the jobs come on the fly and the proposed offline strategy is not practical, thus we propose an online brokerage to take the best advantage of usage-based volume discount for ML jobs.

Hu *et al.* [32] study online cost-effective cloud resource allocation under price discounts. Wu *et al.* [33] design a storage service which span among multiple DCs and minimize the cost by exploiting different prices in each DC. They also trade off geo-distributed replications to achieve better overall performance. Shastri *et al.* [34] exploit the spot markets, always allocate the cheapest instance for tasks, and

migrate tasks from more expensive instances to cheaper ones when spot market prices fluctuate. The above schemes cannot be directly applied to ML systems, since they do not consider training data migration and the communication between workers and PSs.

Regularization in Algorithm Design. The regularization technique was first proposed by Buchbinder *et al.* [35]. A number of studies have applied regularization in different problems. Zhang *et al.* [36] apply it to geo-distributed cloud CDNs, and Jiao *et al.* [37, 38] leverage it for online resource allocation in cloud and edge computing networks. Jia *et al.* [39] investigate the cost minimization problem for dynamic placement of VNF service chains. These studies are not tailored for ML brokerage services. Consequently, their regularization algorithms cannot handle data migration and volume discount functions. Further, in one-shot algorithm design, they only consider one set of integer variables while we jointly round two sets of integer variables.

3 Models and Formulation

3.1 Broker Service Model and Notations

We consider an ML broker who accepts distributed ML job requests from users, and rents cloud resources from R geo-distributed data centers (DCs) for running the jobs. Assume I users submit jobs to the broker over a large time span T. In each ML job i, the training data are continuously generated at different geographic locations. Without loss of generality, we assume that the original copy of the training data is collected and stored into the nearest data center. The owners of ML jobs need to offer specific ML training models, training datasets and resource demands. Specifically, the model needs to be provided is a Docker image, and the k8s can automatically download and deploy training models. Resource demands includes the number of PSs and workers, and their configurations.

The ML jobs use the parameter server (PS) architecture [3] for distributed training. We consider incremental learning of the continuously produced data in each job [5][6]: the new data are aggregated in each time slot, and the dataset collected in $t-1$ is used for training the ML model in t. Let $D_i^r(t)$ denote the size of the input training dataset in job i at t from data center r, which was aggregated in r in $t-1$. The training data in t are divided into equal-sized chunks trained by different workers. Each data chunk is further divided into equal-sized mini-batches. A worker trains a mini-batch, pushes computed model parameter updates to the PS, pulls global parameter computed from the PS, and then moves on to the next minibatch. When all mini-batches in the data chunk are trained for once, a training epoch is completed; the worker typically trains the data chunk for multiple epochs for model convergence. When training ResNet-152 model on ImageNet dataset [40][41], it takes about one second to train a mini-batch, while training a data chunk takes less than one minute [4].

Each ML job is modeled as follows. User i submits its ML job request at t_i, which consists of: (i) desired resource composition for each worker (each PS) to run the job, denoted by $n_{i,k}$ ($m_{i,k}$), the amount of type-k resource demanded by each worker (PS), $\forall k \in [K]$; (ii) processing capacity of each worker in job i, P_i, in terms of the maximum input data size that it can train for the desired number of epochs for its incremental learning in each time slot; (iii) data size for parameter update exchange between each pair of worker and parameter server in job i in a time slot, B_i. B_i is decided by the number of parameters in the ML model being trained, and the number of inter-worker-PS parameter exchanges in a time slot (decided by the job's minibatch size and computation speed of its worker/PS). We assume that there is only one PS in each job in our problem model, which in practice can represent a number of PS instances located in the same data center.

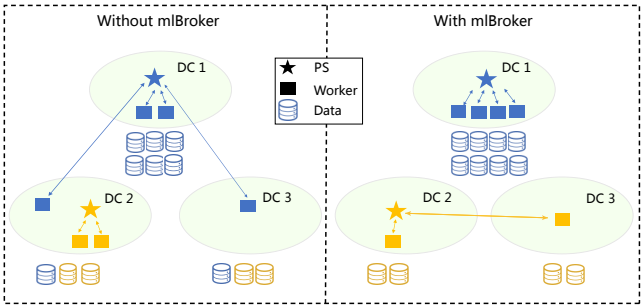


Fig. 2: mlBroker Model.

The broker rents cloud resources and schedules ML job placement in a time slotted fashion. At the beginning of each time slot t, it decides placement of workers and PSs in jobs newly submitted in $t-1$, together with deployment adjustment of workers/PSs in existing jobs submitted earlier, in order to minimize the overall cost of cloud resource rental. For example, in Fig. 2, we use blue and yellow icons to represent job 1 and job 2's workers and PSs respectively. $mlBroker$ is responsible for scaling these two jobs. As can be seen from the figure, the new training data is cumulated during last time slot, then the deployment of workers and PSs need to be rearranged accordingly. After the scheduling of $mlBroker$, the training data and computing nodes are all selectively rescheduled. The amount of training data in each job, $\sum_{r\in[R]} D_i^r(t)$, may vary from one time slot to another; the total number of workers in job i is recomputed by $\lceil \frac{\sum_{r\in[R]} D_i^r(t)}{P_i} \rceil$ in t. The length of a time slot is potentially much larger than the duration of a training epoch, for repeated training of the input dataset in each time slot. For example, one time slot can be half a day or longer, which is sufficient for all the incremental jobs in this period to be trained to convergence.

The decisions made at the broker in t include: (i) $y_i^r(t)$, the number of workers of job i to run in data center r in t, $\forall i \in I_t$, $r \in [R]$ (we use $[X]$ to denote the integer set $\{1, 2, ..., X\}$ in the paper); (ii) $s_i^r(t)$, a binary variable indicating whether job i's PS is placed in data center r in t, or not; (iii) $q_i^{rr'}(t)$, the amount of training data in job i in t, transferred from data center r to r'. Here I_t is the set of ML jobs to schedule in t, including both new and existing jobs.

To maximize the chances for volume discounts, data collected in data center r in $t-1$ may not necessarily be processed in r in t, but moved to another data center r' for processing, if more abundant workers are deployed there. Let $d^{rr'}$ denote the cost of transferring a unit amount of data from r to r' ($d^{rr'}=0$ if $r'=r$).

Our scheduling algorithm keeps track of the total amount of all resources and their usage. If a job arrives and is successfully deployed, our scheduling system records the type and amount of resources used by the job. When the job is completed, the corresponding resources are released and the scheduling system recalculates the amount of available resources.

TABLE 1: Notation

I	# of jobs	R	# of DCs
K	# of resource types	T	# of time slots
t_i	job i's arrival time	I_t	job set in t
X	integer set $\{1, 2, ...X\}$		
$D_i^r(t)$	input data size in job i in t from DC r		
P_i	processing capability of each worker of job i		
$n_{i,k}$	amount of type-k resource required by job i's worker		
$m_{i,k}$	amount of type-k resource required by job i's PS		
$y_i^r(t)$	# of allocated workers for job i in DC r at t		
$s_i^r(t)$	whether job i's parameter server is placed in DC r at t		
$d^{rr'}$	unit data transmission cost from r to r'		
c_i	deployment cost for job i		
$v_i^r(t)$	whether job i's worker(s) or PS is deployed in r in t		
$z_i^r(t)$	whether deployment cost occurs for job i in r at t		
$F_k^r(h)$	volume discount function of type-k resource in DC r		
$q_i^{rr'}(t)$	amount of training data in job i transferred from r to r' at t		
B_i	parameter size exchanged between each worker and the PS in job i per time slot		
$g_i^{rr'}(t)$	# of worker-PS connections in job i, from worker(s) in DC r to the PS in DC r' in t		

3.2 Cost Structure

In each time slot, the broker is subject to four categories of costs for running user jobs.

1) Data transfer costs for copying training datasets from origin data centers to other data centers for processing. The overall data transfer cost in t is computed as:

$$C_1(t) = \sum_{i\in I_t} \sum_{r\in[R]} \sum_{r'\in[R]} d^{rr'} q_i^{rr'}(t) \tag{1}$$

2) Resource costs for renting computing resources in the data centers to run workers and PSs.

The data centers are owned by a common cloud provider or different providers. Each data center provides K types of resources (*e.g.*, different types of virtual machines or containers, storage, etc.). Each data center adopts a pricing scheme F with volume discount, as follows:

$$F_k^r(h) = \begin{cases} a_k^r \cdot h & h \in [0, l_k^r], \\ b_k^r \cdot h + e_k^r & h > l_k^r. \end{cases} \tag{2}$$

Here $F_k^r(h)$ is the per-unit-time price function of data center r for type-k resource, where h is the amount of type-k resource rented. As shown in Fig. 3, l_k^r is the threshold of type-k resource usage in data center r for applying volume discount, decided by the respective cloud provider. a_r^k and b_k^r are the price per unit of type-k resource in data center r, respectively, and $b_k^r < a_k^r$: when the consumption of type-k resource is smaller than l_r^k, a_r^k is applied; otherwise, the lower unit price b_k^r is used, and $e_k^r = a_k^r l_k^r - b_k^r l_k^r$. The price function is a non-decreasing, concave piecewise function, representing those volume-discount schemes in practice [8].

The overall computing resource cost in t is:

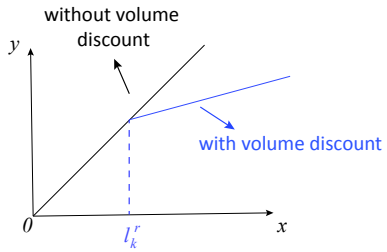


Fig. 3: Volume-discount based price function.

$$C_2(t) = \sum_{r\in[R]} \sum_{k\in[K]} F_k^r\Big(\sum_{i\in I_t} \big(n_{i,k} y_i^r(t) + m_{i,k} s_i^r(t)\big)\Big) \tag{3}$$

3) Deployment costs for placing workers/PSs in data centers where the respective jobs were not deployed in the previous time slot. If there is no worker (PS) of job i running in data center r in $t-1$ (*i.e.*, $y_i^r(t-1) + s_i^r(t-1) = 0$) and one or more workers (or the PS) are to be placed in r in t, a cost c_i occurs for copying job i's ML model and training program from the broker to data center r and launching the respective program.[1] If job i has deployed worker(s) (PS) in data center r at $t-1$, then any new instance of worker (PS) of the job in r in t can copy the image from an existing one, and we ignore deployment cost in this case.[2] We also ignore the cost for removing a worker (PS) from a data center, when the number of workers (PS) in t is reduced from that in $t-1$.

Let constant U denote the upper bound of $y_i^r(t-1) + s_i^r(t-1)$, and binary variable $v_i^r(t)$ indicate whether any worker or the PS of job i is deployed in data center r in t. We have $U \cdot v_i^r(t) \geq y_i^r(t-1) + s_i^r(t-1)$; $v_i^r(t) = 0$ only if the right-hand side (RHS) is zero, and $v_i^r(t) = 1$, otherwise. Using the Rectified Linear Unit [12] function, we let binary variable $z_i^r(t)$ represent the deployment cost for job i in data center r in t:

$$z_i^r(t) = \max\{v_i^r(t) - v_i^r(t-1), 0\}. \tag{4}$$

The overall deployment cost for all jobs in t is:

$$C_3(t) = \sum_{i\in I_t} \sum_{r\in[R]} c_i z_i^r(t) \tag{5}$$

4) Communication cost for transmitting model parameters between workers and PSs across data centers in each training iteration. Let $g_i^{rr'}(t)$ denote the number of worker-PS connections in job i, from worker(s) in data centers r to the PS in data center r' in t. There is one connection between each worker and the PS in each job, which both pulls and pushes traffic of parameters traverses. The overall communication cost for parameter exchange can be formulated as:

1. We suppose the ML model and training programs for both PS and worker are copied to a new data center no matter when a new worker or the PS is to be deployed there. A newly launched worker pulls latest parameters from the PS, whose bandwidth cost is counted into cost 4); a new PS copies the model parameters from its previous deployment.

2. We suppose PS migration only occurs when all workers have pulled the latest parameters from it; hence, a new PS in a data center can copy latest global model parameters from a worker already running in the datacenter.

$$C_4(t) = \sum_{i\in I_t}\sum_{r\in[R]}\sum_{r'\in[R]} d^{rr'}B_i g_i^{rr'}(t) \tag{6}$$

3.3 Cost Minimization Problem

1) Reformulation of piecewise function.

We can formulate the worker/PS placement problem faced by the broker into an optimization program, with the objective of minimizing the sum of costs in (1)(3)(5)(6). We note piecewise functions $F_k^r(h)$ in C_3 in Eqn. (5) can be reformulated as

$$\begin{aligned} &F_k^r\Big(\sum_{i\in I_t}\big(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)\big)\Big)\\ =&\min\Big\{a_k^r\sum_{i\in I_t}\big(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)\big),\\ &b_k^r\sum_{i\in I_t}\big(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)\big)+e_k^r\Big\}. \end{aligned}$$

Define an auxiliary variable $x_k^r(t)$ and let $x_k^r(t) = F_k^r(\cdot)$. Minimizing $C_2(t)$ is equivalent to the following mixed integer linear program (MILP):

$$\text{minimize} \sum_{r\in[R]}\sum_{k\in[K]} x_k^r(t) \tag{7}$$

subject to:

$$\begin{aligned} x_k^r(t) &\ge a_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)) - M\cdot u_{k,r}^1(t), \forall r,k,\\ x_k^r(t) &\ge b_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)) + e_k^r - M\cdot u_{k,r}^2(t), \forall r,k,\\ u_{k,r}^1(t)&+u_{k,r}^2(t) = 1, \forall r,k,\\ u_{k,r}^1(t), u_{k,r}^2(t) &\in \{0,1\}. \end{aligned}$$

Here M is a constant and is an upper bound of $|b_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t))+e_k^r - a_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t))|, \forall r\in[R], k\in[K], t\in[T]$. $u_{k,r}^1(t)$ and $u_{k,r}^2(t)$ are two new binary variables, one and only one of which is 1 at any time. The minimum of the MILP occurs when $x_k^r(t)$ equals the minimum of the two segments of the piecewise function. Hence, we have $\min\sum_{r\in[R]}\sum_{k\in[K]} x_k^r(t) = \min\sum_{r\in[R]}\sum_{k\in[K]} F_k^r\Big(\sum_{i\in I_t}\big(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)\big)\Big)$. Let $C_2^*(t)$ denote the new form of the renting cost:

$$C_2^*(t) = \sum_{r\in[R]}\sum_{k\in[K]} x_k^r(t)$$

2) Cost minimization problem.

The broker's cost minimization problem in all T time slots can be formulated as follows: We assume that these costs are in proportion to each other, while we can also compute their weighted sum. The notation of all variables can be found in Table 1.

$$\mathbf{P}: \quad \text{minimize} \sum_{\mathbf{t}\in[\mathbf{T}]}\big(\mathbf{C_1(t)}+\mathbf{C_2^*(t)}+\mathbf{C_3(t)}+\mathbf{C_4(t)}\big) \tag{8}$$

subject to:

$$P_i y_i^{r'}(t) \ge \sum_{r\in[R]} q_i^{rr'}(t), \forall i,r',t, \tag{8a}$$

$$\sum_{r'\in[R]} q_i^{rr'}(t) = D_i^r(t), \forall i,r,t, \tag{8b}$$

$$x_k^r(t) \ge a_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)) - M\cdot u_{k,r}^1(t), \quad \forall r,k,t, \tag{8c}$$

$$x_k^r(t) \ge b_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)) + e_k^r - M\cdot u_{k,r}^2(t), \quad \forall r,k,t, \tag{8d}$$

$$u_{k,r}^1(t)+u_{k,r}^2(t) = 1, \forall r,k,t, \tag{8e}$$

$$\sum_{r\in[R]} s_i^r(t) = 1, \forall i,t, \tag{8f}$$

$$U\cdot v_i^r(t) \ge y_i^r(t)+s_i^r(t), \forall i,r,t, \tag{8g}$$

$$z_i^r(t) \ge v_i^r(t) - v_i^r(t-1), \forall i,r,t, \tag{8h}$$

$$\sum_{r'\in[R]} g_i^{rr'}(t) = y_i^r(t), \forall i,r,t, \tag{8i}$$

$$\sum_{r\in[R]} g_i^{rr'}(t) \ge s_i^{r'}(t)\lceil\frac{\sum_{\bar{r}\in[R]} D_i^{\bar{r}}(t)}{P_i}\rceil, \forall i,r',t, \tag{8j}$$

$$\begin{aligned} &y_i^r(t), g_i^{rr'}(t) \in \{0,1,2,...\}, u_{k,r}^1(t), u_{k,r}^2(t), s_i^r(t) \in \{0,1\},\\ &v_i^r(t), z_i^r(t) \in \{0,1\}, x_k^r(t), q_i^{rr'}(t) \ge 0, \forall i,r,r',k,t. \end{aligned} \tag{8k}$$

where $\forall i,r,r',k,t$ denote $\forall i\in I_t, r\in[R], r'\in[R], k\in[K], t\in[T]$. Constraint (8a) guarantees that job i's workers' processing capability in data center r' is sufficient for processing all the job's data received from all data centers. (8b) ensures that all the data collected in data center r are processed (by workers potentially in different data centers) in each time slot. Constraints (8c), (8d) and (8e) are from the reformulation of the piecewise price functions in (7). Constraint (8f) ensures that there is one PS in each ML job. (8g) and (8h) are based on discussions in formulating $C_3(t)$ and Eqn. (4). (8i) describes that the number of worker-PS connections of job i from workers in r to PS placed in the same or another data center equals the number of workers in r. (8j) indicates that in each job i, the number of worker-PS connections from workers in different data centers to the PS placed in data center r' is no smaller than the total number of workers in t. We do not consider overall resource capacity constraints in the DCs, as a cloud typically provides sufficient resources for serving a large number of customers, while the broker is only one.

Theorem 1. *The cost minimization problem in (8) is an NP-hard problem.*

Proof. The switching cost functions in our problem are discrete and non-convex, and can be higher-degree polynomials, collectively hard to optimize. Even in the offline setting and even without the switching cost, our problem is a more complex version of the "0-1 integer programming problem (ILP)". The 0-1 ILP, in which unknowns are binary, and only the restrictions must be satisfied, is one of Karp's 21 NP-complete problems [42]. As for (8), when we only consider $s_i^r(t)$ and (8f) and let all other variables be zero, the problem can be transferred to a special case, which is a 0-1 ILP. Therefore, the 0-1 ILP can be reduced in polynomial time to our problem, and then our problem is an NP-hard problem. □

3.4 Algorithmic Idea

In order to efficiently solve the cost minimization problem in (8), we design a novel online algorithm *mlBroker*, leveraging the *regularization* and *dependent rounding* techniques, to tackle the aforementioned difficulties. As shown in Fig. 4, the basic idea of our online algorithm design is divided into two steps.

- **Step 1**: In Sec. 4, we first relax the integrality constraints of (8k) in $\boldsymbol{P}$ to allow real solutions. Then by leveraging the regularization technique [35], we substitute the time-coherent deployment costs with carefully-designed logarithmic forms. We transfer the relaxed problem $\boldsymbol{P}_f$ into a more solvable problem $\tilde{\boldsymbol{P}}_f$, which can be easily decoupled into a series of time-independent convex subproblems $\{\tilde{\boldsymbol{P}}_{ft}, \forall t\}$, based on the previous and current system information. $A_{fractional}$ computes the optimal fractional solution for each $\{\tilde{\boldsymbol{P}}_{ft}, \forall t\}$, utilizing the interior point method.
- **Step 2**: In Sec. 5, by taking the integral constraints into consideration, we round the fractional solutions of $\{\boldsymbol{y}(t), \boldsymbol{s}(t), \boldsymbol{g}(t), \boldsymbol{v}(t), \boldsymbol{u}(t)\}$ generated by $\{\tilde{\boldsymbol{P}}_{ft}, \forall t\}$ back into integral ones with a feasible dependent rounding algorithm A_{round}. Note that $\sum_t OPT(\tilde{\boldsymbol{P}}_{ft}) = OPT(\tilde{\boldsymbol{P}}_f)$. $A_{fractional}(P_f)$ is the the overall cost in (9) generated by $A_{fractional}$, and $mlBroker(P)$ is the the overall cost in (8) generated by our complete online algorithm $mlBroker$. Finally, the competitive ratio of our complete algorithm $mlBroker$ is $r_1 r_2$.

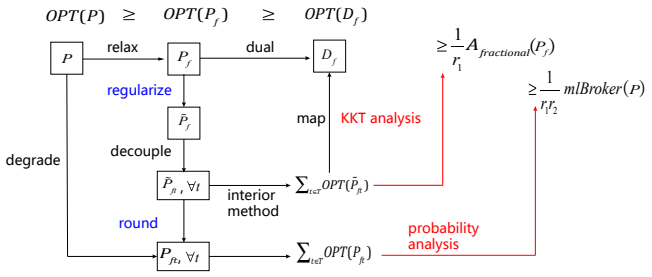


Fig. 4: Basic idea of our online algorithm *mlBroker* and main route of our performance analysis.

In the following, we will discuss our complete online algorithm $mlBroker$ in details, consisting of an one-shot regularization-based fractional algorithm $A_{fractional}$ (see Sec. 4) and a dependent rounding algorithm A_{round} (see Sec. 5).

4 An Online Fractional Algorithm

4.1 Decomposing to One-shot Problems

The offline cost minimization problem (8) is an MILP. Even the complete system information in the whole life span $[T]$ is known, solving such an MILP is non-trivial and NP-hard [43], (see Theorem 1). Toward the design of an efficient online algorithm, we first relax the integrality constraints in (8k). Let P_f denote the relaxed LP:

$$\mathbf{P_f}: \quad \text{minimize} \sum_{\mathbf{t}\in[\mathbf{T}]} \left(\mathbf{C_1(t)} + \mathbf{C_2^*(t)} + \mathbf{C_3(t)} + \mathbf{C_4(t)}\right) \quad (9)$$

subject to:

$$(8a)-(8j), \forall t$$
$$y_i^r(t), g_i^{rr'}(t), u_{k,r}^1(t), u_{k,r}^2(t), s_i^r(t), v_i^r(t), z_i^r(t),$$
$$x_k^r(t), q_i^{rr'}(t) \geq 0, \forall i, r, r', k, t. \quad (9k)$$

The main difficulty in solving (9) in an online manner lies in constraint (8h) (related to deployment cost $C_3(t)$) which couples every two consecutive time slots. The job placement decisions made at one time slot influence the deployment cost in the next time slot. Without knowing future input (*i.e.*, data volume and geo-distribution), we seek to guarantee that the total cost of our online algorithm over the whole time span does not exceed a small number times that of the offline optimal algorithm which knows all input in $[T]$ a priori, *i.e.*, a bounded *competitive ratio*.

To design the online algorithm, we utilize the regularization method [35], which is achieved by adding a smooth convex function to a given objective function and then greedily solving the new online problem, in order to obtain good competitive bound and stable performance. We then decompose (9) into sub problems, independently solvable one at a time. We remove constraint (8h) and the non-negativity constraint $z_i^r(t) \geq 0$, and rewrite $C_3(t)$ as $\sum_{r\in[R]} \sum_{i\in I_t} c_i \max\{v_i^r(t) - v_i^r(t-1), 0\}$ in the objective. This objective term is non-convex; we further substitute $\max\{v_i^r(t) - v_i^r(t-1), 0\}$ with a carefully-designed regularized convex function. The non-convex term can be approximately interpreted as the *L_1-distance*, as the non-negative change between the two variables $v_i^r(t)$ and $v_i^r(t-1)$. The relative entropy function is known as an efficient regularizer for $L_1 - distance$ in online learning literature [35]: $v_i^r(t) \ln \frac{v_i^r(t)}{v_i^r(t-1)} + v_i^r(t-1) - v_i^r(t)$.

In order to ensure the feasibility when $v_i^r(t-1) = 0$, we add a small constant term ϵ on both the denominator and the nominator of the fraction within the $\ln$ operator. Moreover, we define a factor $\sigma = \ln(1 + \frac{1}{\epsilon})$ and multiply the improved relative entropy function by $\frac{1}{\sigma}$, in order to scale down the deployment cost, since we add the constant term ϵ. The regularization function is strictly increasing with $v_i^r(t)$, convex, and differentiable.

Then, $C_3(t)$ is reformulated as follows:

$$C_3(t) = \sum_{i\in I_t} \sum_{r\in[R]} \frac{c_i}{\sigma} \left((v_i^r(t)+\epsilon) \ln \frac{v_i^r(t)+\epsilon}{v_i^r(t-1)+\epsilon} + v_i^r(t-1) - v_i^r(t)\right)$$

With the above trade off between movement cost and relative entropy plus a linear term, the coupling between time slot $t-1$ and t in the contraints can be removed; the resulting new relaxed offline problem $\widetilde{P}_f$ can be readily decomposed into a series of sub-problems $\widetilde{P}_f = \sum_{t\in[T]} \widetilde{P}_{ft}$, each to be solved in a single time slot. The sub problem, $\widetilde{P}_{ft}$, to be solved in time slot t, is as follows, where values of the previous decisions, $v_i^r(t-1)$'s, are given as inputs:

$$\text{minimize} \quad C_1(t) + C_2^*(t) + C_4(t)$$
$$+ \sum_{i\in I_t} \sum_{r\in[R]} \frac{c_i}{\sigma} \left((v_i^r(t)+\epsilon) \ln \frac{v_i^r(t)+\epsilon}{v_i^r(t-1)+\epsilon} - v_i^r(t)\right) \quad (10)$$

subject to:

$$(8a) - (8g), (8i), (8j), (9k), without\ \forall t \in [T]$$

Note that in the objective, we further omit the plus term $v_i^r(t-1)$ in the part corresponding to $C_3(t)$, which does not affect the optimal solutions of $v_i^r(t)$'s derived by solving $\widetilde{P}_{ft}$.

4.2 Competitive Ratio Analysis

Algorithm 1 An Online Regularization-Based Fractional Algorithm $A_{fractional}$

Input: $R, K, \epsilon, \{t_i, P_i, \{n_{i,k}, m_{i,k}\}_{k\in[K]}, B_i\}_{\forall i\in[I]}, \boldsymbol{d}, \boldsymbol{c}, \boldsymbol{F}$
Output: $\boldsymbol{q}(t), \boldsymbol{x}(t), \tilde{\boldsymbol{s}}(t), \tilde{\boldsymbol{v}}(t), \tilde{\boldsymbol{y}}(t), \tilde{\boldsymbol{g}}(t), \tilde{\boldsymbol{u}}^{\boldsymbol{1}}(t), \tilde{\boldsymbol{u}}^{\boldsymbol{2}}(t)$

1: Initialization: $x_k^r(t)=0$, $y_i^r(t)=0$, $s_i^r(t)=0$, $q_i^{rr'}(t)=0$, $g_i^{rr'}(t)=0$, $v_i^r(t)=0$, $u_{k,r}^1(t)=0$, $u_{k,r}^2(t)=0\ \forall i\in[I], r, r'\in[R], t\in[T]$;
2: **for** $t\in[T]$ **do**
3: Observe values of $D_i^r(t), v_i^r(t-1), \forall i\in I_t, r\in[R]$;
4: Use the interior point method to solve $\tilde{P}_{ft}$ in (10);
5: **return** the fractional solutions $\boldsymbol{x}(t)$, $\boldsymbol{q}(t)$, $\tilde{\boldsymbol{y}}(t)$, $\tilde{\boldsymbol{s}}(t)$, $\tilde{\boldsymbol{g}}(t)$, $\tilde{\boldsymbol{v}}(t)$, $\tilde{\boldsymbol{u}}^1(t)$, $\tilde{\boldsymbol{u}}^2(t)$
6: **end for**

Each subproblem $\tilde{P}_{ft}$ is a convex program; we can use many mature algorithms to solve it in polynomial time, such as interior point methods [35]. Our online fractional algorithm $A_{fractional}$ is presented in Alg. 1, which produces fractional solution ($\boldsymbol{x}(t)$, $\boldsymbol{q}(t)$, $\tilde{\boldsymbol{y}}(t)$, $\tilde{\boldsymbol{s}}(t)$, $\tilde{\boldsymbol{g}}(t)$, $\tilde{\boldsymbol{v}}(t)$, $\tilde{\boldsymbol{u}}^{\boldsymbol{1}}(t)$, $\tilde{\boldsymbol{u}}^{\boldsymbol{2}}(t)$) by solving $\widetilde{P}_{ft}$ at each time slot t, with the knowledge of previous and current system information.

Theorem 2. *The online fractional algorithm $A_{fractinoal}$ produces a feasible solution of P_f in polynomial time.*

Proof. The main body in $mlBroker$ is using the interior point method [35] to solve $\widetilde{P}_{ft}$ in (10), which is a method to solve convex problem in polynomial time, thus Theorem 2 holds. □

Theorem 3. *The competitive ratio of $A_{fractinoal}$ is $r_1 = 2+(1+\epsilon)\ln(1+\frac{1}{\epsilon})$, which is an upper-bound of the ratio of the overall cost in (9) generated by $A_{fractional}$ to the cost produced by the offline optimal solution of P_f.*

Proof. We formulate the dual of the relax LP. Let $\alpha_i^{r'}(t)$, $\beta_i^r(t)$, $\theta_k^r(t)$, $\rho_k^r(t)$, $\xi_k^r(t)$, $\gamma_i(t)$, $\phi_i^r(t)$, $\mu_i^r(t)$, $\psi_i^r(t)$ and $\eta_i^{r'}(t)$ denote the Lagrangian dual variables associated with (8a) - (8j), respectively. Then the dual program D_f of P_f is as follows:

$$\text{maximize} \quad \sum_{t\in[T]}\Big(\sum_{i\in[I]}\sum_{r\in[R]} D_i^r(t)\beta_i^r(t) + \sum_{k\in[K]}\sum_{r\in[R]} e_k^r\rho_k^r(t) + \sum_{i\in[I]}\gamma_i(t)\Big) \tag{11}$$

subject to:

$$\theta_k^r(t)+\rho_k^r(t)\le 1, \forall r,k,t, \tag{11a}$$
$$-\alpha_i^{r'}(t)+\beta_i^r(t)\le d^{rr'}, \forall i,r,r',t, \tag{11b}$$
$$P_i\alpha_i^r(t) - a_k^r n_{i,k}\theta_k^r(t) - b_k^r n_{i,k}\rho_k^r(t) - \phi_i^r(t) - \psi_i^r(t)\le 0, \quad \forall i,r,k,t, \tag{11c}$$
$$-a_k^r m_{i,k}\theta_k^r(t) - b_k^r m_{i,k}\rho_k^r(t) + \gamma_i(t) - \phi_i^r(t) - \lceil\frac{\sum_{r\in[R]} D_i^r(t)}{P_i}\rceil\eta_i^r(t)\le 0, \forall i,r,k,t, \tag{11d}$$
$$\psi_i^r(t)+\eta_i^{r'}(t)\le d^{rr'}B_i, \forall i,r,r',t, \tag{11e}$$
$$U\phi_i^r(t) - \mu_i^r(t) + \mu_i^r(t+1) + \le 0, \forall i,r,t, \tag{11f}$$
$$\mu_i^r(t)\le c_i, \forall i,r,t, \tag{11g}$$
$$M\theta_k^r(t)+\xi_k^r(t)\le 0, \forall r,k,t, \tag{11h}$$
$$M\rho_k^r(t)+\xi_k^r(t)\le 0, \forall r,k,t, \tag{11i}$$
$$\alpha_i^{r'}(t), \theta_k^r(t), \rho_k^r(t), \phi_k^r(t), \mu_k^r(t), \eta_i^r(t)\ge 0, \forall i,r,r',k,t. \tag{11j}$$

Let P_f^* and D_f^* denote the optimum of P_f and D_f respectively. We use P_f and D_f to denote the objective value of a feasible solution of P_f and D_f, respectively. Note we already have $D_f \le D_f^* = P_f^*$ due to the strong duality. P_f can be obtained by $ORFA$. If we can further find a feasible dual solution of (11) and prove $P_f \le r_1 D_f$ for a certain constant r_1, then this r_1 is the competitive ratio.

Therefore, the key point is to find a feasible dual solution of (11). Our goal is to assign values of a set of dual variables within the feasible region defined by the constraints. Now, we define $\tilde{D}_f$ as the dual problem of the regularized problem $\tilde{P}_f$. Let $\tau_k^r(t)$, $\nu_i^r(t)$, $\varphi_i^r(t)$, $\pi_i^{rr'}(t)$, $\varsigma_i^{rr'}(t)$, $\iota_k^r(t)$, $\kappa_i^r(t)$, $\delta_{k,r}^1(t)$, $\delta_{k,r}^2(t)$ denote dual variables associated with $x_k^r(t)$, $y_i^r(t)$, $s_i^r(t)$, $q_i^{rr'}(t)$, $g_i^{rr'}(t)$, $v_i^r(t)$, $u_{k,r}^1(t)$, $u_{k,r}^2(t)$. We write the following Karush-Kuhn-Tucker (KKT) conditions that characterize the optimal solution of $\widetilde{P}_t$ and $\widetilde{D}_t$ in Table. 2, where $\forall i,r,k,t$ mean $\forall i\in[I], r\in[R], k\in[K], t\in[T]$, respectively.

Following from the KKT conditions (K1)-(K19), a feasible solution of the regularized dual problem can be derived as follows: $\widetilde{\alpha_i^{r'}}(t)=\alpha_i^{r'}(t)$, $\widetilde{\beta_i^r}(t)=\beta_i^r(t)$, $\widetilde{\theta_k^r}(t)=\theta_k^r(t)$, $\widetilde{\rho_k^r}(t)=\rho_k^r(t)$, $\widetilde{\gamma_i}(t)=\gamma_i(t)$, $\widetilde{\phi_i^r}(t)=\phi_i^r(t)$, $\widetilde{\psi_i^r}(t)=\psi_i^r(t)$, $\widetilde{\eta_i^r}(t)=\eta_i^r(t)$, $\widetilde{\xi_i^{r'}}(t)=\xi_i^{r'}(t)$ and $\widetilde{\mu_i^r}(t)=\frac{c_i}{\sigma}\ln\frac{1+\sigma}{\widetilde{v_i^r}(t-1)}$. In order to verify that it satisfies the constraints (11a)-(11i), we take (11g) as an example. We rewrite the left-hand side, then we have the following:

$$\begin{aligned}
&-\widetilde{\mu_i^r}(t)+\widetilde{\mu_i^r}(t+1)+U\widetilde{\phi_i^r}(t)\\
=&-\frac{c_i}{\sigma}\ln\frac{1+\sigma}{\widetilde{v_i^r}(t-1)}+\frac{c_i}{\sigma}\ln\frac{1+\sigma}{\widetilde{v_i^r}(t)}+U\widetilde{\phi_i^r}(t)\\
=&\frac{c_i}{\sigma}\ln\frac{\widetilde{v_i^r}(t)+\epsilon}{\widetilde{v_i^r}(t-1)+\epsilon}+U\widetilde{\phi_i^r}(t)\le 0,
\end{aligned}$$

which indicates (11g) is feasible. All the rest constraints can be proved feasible analogously.

Step 1: We bound the static costs, *i.e.*, $\sum_{t\in[T]} C_1(t) + C_2^*(t) + C_4(t)$:

$$\sum_{t\in[T]} C_1(t) + C_2^*(t) + C_4(t) \tag{12}$$

TABLE 2: K.K.T Optimality Conditions

$1-\theta_k^r(t)-\rho_k^r(t)-\tau_k^r(t)=0,\forall i,r,k$	(K1)
$P_i\alpha_i^r(t)-a_k^r n_{i,k}\theta_k^r(t)-b_k^r n_{i,k}\rho_k^r(t)-\phi_i^r(t)$ $-\psi_i^r(t)+\nu_i^{r'}(t)=0,\forall i,r,k$	(K2)
$-a_k^r m_{i,k}\theta_k^r(t)-b_k^r m_{i,k}\rho_k^r(t)+\gamma_i(t)-\phi_i^r(t)$ $-\lceil\frac{\sum_{r\in[R]}D_i^r(t)}{P_i}\rceil\eta_i^r(t)+\varphi_i^r(t)=0,\forall i,r,k$	(K3)
$-\alpha_i^r(t)+\beta_i^r(t)-d^{rr'}+\pi_i^{rr'}(t)=0,\forall i,r,r'$	(K4)
$\eta_i^r(t)+\psi_i^r(t)-d^{rr'}B_i+\varsigma_i^{rr'}(t)=0,\forall i,r,r'$	(K5)
$-\mu_i^r(t)+\mu_i^r(t+1)+U\phi_i^r(t)+\iota_k^r(t)=0,\forall r,k$	(K6)
$M\theta_k^r(t)+\xi_k^r(t)+\delta_{k,r}^1(t)=0$	(K7)
$M\rho_k^r(t)+\xi_k^r(t)+\delta_{k,r}^2=0$	(K8)
$\alpha_i^{r'}(t)(P_i y_i^{r'}(t)-\sum_{r\in[R]}q_i^{rr'}(t))=0,\forall i,r,r'$	(K9)
$\beta_i^r(t)(\sum_{r'\in[R]}q_i^{rr'}(t)-D_i^r(t))=0,\forall i,r,r'$	(K10)
$\theta_k^r(t)a_k^r\sum_{i\in[I_t]}\left(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)\right)$ $-\theta_k^r(t)x_k^r(t)-\theta_k^r(t)Mu_{k,r}^1=0,\forall r,k$	(K11)
$\rho_k^r(t)b_k^r\sum_{i\in[I_t]}\left(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)\right)$ $+\rho_k^r(t)e_k^r-\rho_k^r(t)x_k^r(t)-\rho_k^r(t)Mu_{k,r}^2=0,\forall r,k$	(K12)
$\gamma_i(t)(\sum_{r\in[R]}s_i^r(t)-1)=0,\forall i$	(K13)
$\xi_{k,r}^t(1-u_{r,k}^1-u_{k,r}^2(t))$	(k14)
$\phi_i^r(t)(U\cdot v_i^r(t)-y_i^r(t)-s_i^r(t))=0,\forall i,r$	(K15)
$\psi_i^r(t)(\sum_{r'\in[R]}g_i^{rr'}(t)-y_i^r(t))=0,\forall i,r,r'$	(K16)
$\eta_i^{r'}(t)(\sum_{r\in[R]}g_i^{rr'}(t)-s_i^{r'}(t)\lceil\frac{\sum_{r\in[R]}D_i^r(t)}{P_i}\rceil)=0,\forall i,r,r'$	(K17)
$y_i^r(t)\nu_i^r(t)=0,\forall i,r$	(K18)
$g_i^{rr'}(t)\varsigma_i^{rr'}(t)=0,\forall i,r,r'$	(K19)
$q_i^{rr'}(t)\pi_i^{rr'}(t)=0,\forall i,r,r'$	(K20)
$s_i^r(t)\varphi_i^r(t)=0,\forall i,r$	(K21)
$x_k^r(t)\tau_k^r(t)=0,\forall i,r$	(K22)
$\tau_k^r(t),\nu_i^r(t),\varphi_i^r(t),\pi_i^{rr'}(t),\varsigma_i^{rr'}(t),\iota_k^r(t),\delta_{k,r}^1(t),\eta_i^r(t)$ $\delta_{k,r}^2(t),\kappa_i^r(t),\alpha_i^r(t),\theta_k^r(t),\rho_k^r(t),\mu_i^r(t),\phi_i^r(t)\geq 0,\forall i,r,r',k$	

$$
\begin{aligned}
&=\sum_{t\in[T]}\Big(\sum_{i\in[I]}\sum_{r\in[R]}\sum_{r'\in[R]}d^{rr'}q_i^{rr'}(t)+\sum_{r\in[R]}\sum_{k\in[K]}x_k^r(t)\\
&\quad+\sum_{i\in[I]}\sum_{r\in[R]}\sum_{r'\in[R]}d^{rr'}B_i g_i^{rr'}(t)\Big) &\text{(12a)}\\
&\leq\sum_{t\in[T]}\Big(\sum_{i\in[I]}\sum_{r\in[R]}\sum_{r'\in[R]}(\beta_i^r(t)-\alpha_i^r(t))q_i^{rr'}(t)\\
&\quad+\sum_{r\in[R]}\sum_{k\in[K]}(b_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t))+e_k^r)(\theta_k^r(t)+\rho_k^r(t))\\
&\quad+\sum_{i\in[I]}\sum_{r\in[R]}\sum_{r'\in[R]}(\eta_i^r(t)+\phi_i^r(t))g_i^{rr'}(t)\Big) &\text{(12b)}\\
&\leq\sum_{t\in[T]}\Big(\sum_{i\in[I]}\sum_{r\in[R]}D_i^r(t)\beta_i^r(t)-\sum_{i\in[I]}\sum_{r\in[R]}D_i^r(t)\alpha_i^r(t)\\
&\quad+\sum_{i\in[I]}\sum_{r\in[R]}(P_i\alpha_i^r(t)+\frac{P_i}{D_i^r(t)}\gamma_i(t))\lceil\frac{\sum_{r\in[R]}D_i^r(t)}{P_i}\rceil\\
&\quad+\sum_{r\in[R]}\sum_{k\in[K]}e_k^r\rho_k^r(t)+P_i^r(t)\alpha_i^r(t)+\gamma_i(t)+e_k^r\theta_k^r(t)\Big) &\text{(12c)}\\
&\leq 2\sum_{t\in[T]}\Big(\sum_{i\in[I]}\sum_{r\in[R]}D_i^r(t)\beta_i^r(t)+\sum_{i\in[I]}\gamma_i(t)+\sum_{r\in[R]}\sum_{k\in[K]}e_k^r\rho_k^r(t)\Big) &\text{(12d)}\\
&=2D.
\end{aligned}
$$

(12a) holds due to the definition of these costs. (12b) follows from (K1), (K4), (K5), (K12) and (K20). (12c) follows from (K1), (K2), (K3), (K10) and (K17). (12d) follows from $K\cdot R\leq I$ and some basic reformulations.

Step 2: We then bound the reconfiguration cost, C_3:

$$
\begin{aligned}
&\sum_{t\in[T]}C_3(t) &\text{(13)}\\
&=\sum_{t\in[T]}\sum_{i\in I_t}\sum_{r\in[R]}c_i^r z_i^r(t) &\text{(13a)}\\
&=\sum_{t\in[T]}\sum_{i\in I_t}\sum_{r\in[R]}c_i^r(v_i^r(t)-v_i^r(t-1)) &\text{(13b)}\\
&\leq\sum_{t\in[T]}\sum_{i\in I_t}\sum_{r\in[R]}\sigma(v_i^r(t)+\epsilon)\frac{c_i^r}{\sigma}\ln\frac{v_i^r(t)+\epsilon}{v_i^r(t-1)+\epsilon} &\text{(13c)}\\
&\leq\sum_{t\in[T]}\sum_{i\in I_t}\sum_{r\in[R]}\sigma(v_i^r(t)+\epsilon)(-U\phi_i^r(t)) &\text{(13d)}\\
&\leq\sigma(1+\epsilon)\sum_{t\in[T]}\sum_{i\in I_t}\sum_{r\in[R]}D_i^r(t)\beta_i^r(t) &\text{(13e)}\\
&\leq(1+\epsilon)\ln(1+\frac{1}{\epsilon})D. &\text{(13f)}
\end{aligned}
$$

(13a) and (13b) holds due to the definition of reconfiguration. (13c) holds due to the fact $a-b\leq aln\frac{a}{b},\forall a,b>0$. (13d) follows from (K6) and (13e) is supported by (K4) and the fact that $-U\phi_i^r(t)\leq 0\leq\alpha_i^r(t)\leq\beta_i^r(t)\leq D_i^r(t)\beta_i^r(t)$. (13f) holds since $\sigma=\ln(1+\frac{1}{\sigma})$ and $\sum_{t\in[T]}\sum_{i\in[I]}\sum_{r\in[R]}D_i^r(t)\beta_i^r(t)\leq D$, which is intuitively less than D. According to Steps 1 and 2, Theorem 3 holds. □

5 A Randomized Dependent Rounding Algorithm

The algorithm $A_{fractional}$ computes fractional solutions for the relaxed primal program. However, the number of workers and parameter servers should be integers, so do the number of communication links and binary states that indicate whether migration occurs and which segment to choose in the volume discount piecewise function. We need to round the fractional solutions $\tilde{\boldsymbol{y}}(t)$, $\tilde{\boldsymbol{s}}(t)$, $\tilde{\boldsymbol{g}}(t)$, $\tilde{\boldsymbol{v}}(t)$, $\tilde{\boldsymbol{u}}^{\boldsymbol{1}}(t)$, $\tilde{\boldsymbol{u}}^{\boldsymbol{2}}(t)$ into integer solutions to satisfy constraint (8k), as well as other constraints where they appear. Due to those coupling constraints, the variables are dependent on each other, and the traditional independent rounding scheme, by which each variable is rounded up or down independently, does not apply (which may well lead to constraint violation). We therefore design a novel randomized dependent rounding algorithm to explore the inherent dependence among the variables.

5.1 Algorithm Design

Our algorithm consists of three steps: first, we round $\tilde{\boldsymbol{y}}$ according to constraints (8a), (8c) and (8d); next, we round $\tilde{\boldsymbol{s}}$ into $\{0,1\}$ according to constraints (8f); then, given the rounded values of $\boldsymbol{y}$ and $\boldsymbol{s}$, we minimize (10) to derive the integer solutions of $\bar{\boldsymbol{u}}^{\boldsymbol{1}}$, $\bar{\boldsymbol{u}}^{\boldsymbol{2}}$, $\bar{\boldsymbol{g}}$, $\bar{\boldsymbol{v}}$ and the real-valued solutions of $\boldsymbol{q}$ and $\boldsymbol{x}$ under constraints (8a)-(8e), (8g), (8i)–(8k). The algorithm, A_{round}, is given in Alg. 2.

Step 1: The key idea of rounding $\tilde{\boldsymbol{y}}$ is to compensate the round-down variables with the round-up ones to ensure that the values of $\sum_{i\in[I_t]}n_{i,k}y_i^r(t)$ and $\sum_{r\in[R]}P_i y_i^r(t)$ remain similar as before rounding, in order to ensure constraints (8a)–(8d) can still be satisfied. We first round $\tilde{\boldsymbol{y}}$ to ensure $\sum_{i\in[I_t]}n_{i,k}y_i^r(t)$ or $\sum_{r\in[R]}P_i y_i^r(t)$ remains similar as before, respectively (lines 1-3 and line 4, Alg. 2); then for each $\tilde{y}_i^r(t)$, we pick the largest value among its $K+1$ rounded values, and use that as the integer solution $\bar{y}_i^r(t)$ (line 5, Alg. 2).

In $Round1\_y$, we use $\{n_{i,k}\}_{\forall i\in[I_t]}$ as the input value of argument $\boldsymbol{\omega}$ and we maintain a set of jobs for each data

Algorithm 2 The Dependent Rounding Algorithm in t, A_{round}

Input: $\tilde{\boldsymbol{y}}(t), \tilde{\boldsymbol{s}}(t), \boldsymbol{D}, \{t_i, P_i, \{n_{i,k}, m_{i,k}\}_{k\in[K]}, B_i\}_{\forall i\in[I]}$
Output: $\bar{\boldsymbol{s}}(t)$, $\bar{\boldsymbol{u}}^{\boldsymbol{1}}(t)$, $\bar{\boldsymbol{u}}^{\boldsymbol{2}}(t)$, $\bar{\boldsymbol{y}}(t)$, $\bar{\boldsymbol{v}}(t)$, $\bar{\boldsymbol{g}}(t)$, $\boldsymbol{q}(t)$, $\boldsymbol{x}(t)$

for each resource $k \in [K]$ **do**
  $\bar{\boldsymbol{y}}_k^1(t)$= $\boldsymbol{Round1\_y}\big(\tilde{\boldsymbol{y}}(t), \boldsymbol{n}_k\big)$;
end for
$\bar{\boldsymbol{y}}^2(t)$= $\boldsymbol{Round2\_y}\big(\tilde{\boldsymbol{y}}(t)\big)$;
Set $\bar{y}_i^r(t) = \max\{\{\bar{y}_{ki}^{1r}(t)\}_{\forall k\in[K]}, \bar{y}_i^{2r}(t)\}, \forall i \in I_t, r \in [R]$;
for $i \in I_t$ **do**
  Select $r_i \in [R]$ randomly using $\tilde{s}_i^r(t), \forall r \in [R]$, as the probability distribution;
  Set $\bar{s}_i^{r_i}(t) = 1$; Set $\bar{s}_i^r(t) = 0$ for all $r \neq r_i$;
end for
Compute $\bar{\boldsymbol{u}}^{\boldsymbol{1}}(t)$, $\bar{\boldsymbol{u}}^{\boldsymbol{2}}(t)$, $\bar{\boldsymbol{v}}(t)$, $\bar{\boldsymbol{g}}(t)$, $\boldsymbol{q}(t)$, $\boldsymbol{x}(t)$ which minimizes $\boldsymbol{P}_{rt}$ under constraints (8a)-(8e), (8g), (8i)–(8k), and the rounded solutions of $\bar{\boldsymbol{y}}(t)$ and $\bar{\boldsymbol{s}}(t)$;
return $\bar{\boldsymbol{y}}$, $\bar{\boldsymbol{s}}$, $\bar{\boldsymbol{u}}^{\boldsymbol{1}}$, $\bar{\boldsymbol{u}}^{\boldsymbol{2}}$, $\bar{\boldsymbol{v}}$, $\bar{\boldsymbol{g}}$, $\boldsymbol{q}$, $\boldsymbol{x}$

center r, $Y_{\mathbb{R}}^r(t) \triangleq \{i | y_i^r(t) \notin \mathbb{Z}, i \in I_t\}$, which contains all jobs whose number of workers deployed in data center r in t is not an integer (lines 2-7, Alg. 3). A probability $p_i(t)$ is associated with each job in this set, initialized to be the fractional part of $\tilde{y}_i^r(t)$, *i.e.*, $\tilde{y}_i^r(t) - \lfloor \tilde{y}_i^r(t) \rfloor$. We repeatedly randomly pick two jobs from the set, and compute two weights Ψ_1 and Ψ_2 for the two jobs according to their corresponding $p_i(t)$ and ω_i values (lines 9-11). Then we use probabilities decided by Ψ_1 and Ψ_2 to update $p_i(t)$'s for the two jobs (lines 12-15). After this, if a job i's $p_i(t)$ becomes an integer (0 or 1), we round $\tilde{y}_i^r(t)$ up if $p_i(t)$ is 1 and down if $p_i(t)$ equals 0, and remove the job from set $Y_{\mathbb{R}}^r(t)$ (lines 16-21). When there is a single job left in the set, we just round its $\tilde{y}_i^r(t)$ up (lines 23-25).

With carefully designed update rules, $p_i(t)$'s are always within $[0, 1]$ and will eventually become 0 or 1 (*i.e.*, the while loop will terminate). For example, if $1 - p_{i_1}(t) \leq \frac{\omega_{i_2}}{\omega_{i_1}} p_{i_2}(t)$, then $\Psi_1 = 1 - p_{i_1}(t)$ and $p_{i_1}(t) = p_{i_1}(t) + 1 - p_{i_1}(t) = 1$, *i.e.*, $p_{i_1}(t)$ becomes 1 and $Y_{\mathbb{R}}^r(t) = Y_{\mathbb{R}}^r(t) - 1$. Meanwhile, $p_{i_2}(t) \geq \frac{\omega_{i_1}}{\omega_{i_2}} \Psi_1$, so $0 \leq p_{i_2}(t) = p_{i_2}(t) - \frac{\omega_{i_2}}{\omega_{i_1}} \Psi_{i_1}(t) \leq 1$, *i.e.*, $p_{i_2}(t)$ is still within $[0, 1]$. In Alg. 3, line 13 and line 15 are executed randomly; if one between $p_{i_1}(t)$ and $p_{i_2}(t)$ goes up, the other must go down, and the sum of $\omega_{i_1,r} \tilde{y}_{i_1}^r(t) + \omega_{i_2,r} \tilde{y}_{i_2}^r(t)$ remains the same (proved in Lemma 1).

$Round2\_y$ is almost the same with $Round1\_y$ (hence omitted due to space limit), except for the following: (i) We maintain a set of data centers for each job i, $Y_{\mathbb{R}}^i(t) \triangleq \{r | y_i^r(t) \notin \mathbb{Z}, r \in [R]\}$, which contains all data centers in which the number of job i's workers deployed in t is not an integer. $Y_{\mathbb{R}}^r(t)$ in Alg. 3 is replaced with $Y_{\mathbb{R}}^i(t)$ and we exchange line 1 with line 3. (ii) We just input all fractional solutions $\tilde{\boldsymbol{y}}(t)$ to $Round2\_y$, since P_i is the same for $r \in [R]$ when considering job i. In $Round2\_y$, we can ensure $\sum_{r\in R} P_i y_i^r(t)$ remains at similar values after rounding.

We round worker number of the last remaining job in set $Y_{\mathbb{R}}^r(t)$ ($Y_{\mathbb{R}}^i(t)$) up in $Round1\_y$ ($Round2\_y$) and set $\bar{y}_i^r(t)$ to be the largest among the $K+1$ rounded values of $\tilde{y}_i^r(t)$ (in A_{round}), in order to ensure deploying sufficient workers to process the datasets at each time t (*i.e.*, feasibility of constraints (8a)(8b)).

Algorithm 3 Randomized Algorithm for Rounding $\tilde{\boldsymbol{y}}$ in t, $Round1\_y$

Input: $\tilde{\boldsymbol{y}}(t)$, $\boldsymbol{\omega}$
Output: $\bar{\boldsymbol{y}}(t)$

1: **for** each $r \in [R]$ **do**
2: Set $Y_{\mathbb{R}}^r(t) = \varnothing$;
3: **for** $i \in I_t$ **do**
4: **if** $y_i^r(t) \neq \lfloor y_i^r(t) \rfloor$ **then**
5: $Y_{\mathbb{R}}^r(t) = Y_{\mathbb{R}}^r(t) \cup \{i\}$, $p_i(t) \triangleq \tilde{y}_i^r(t) - \lfloor \tilde{y}_i^r(t) \rfloor$;
6: **end if**
7: **end for**
8: **while** $|Y_{\mathbb{R}}^r(t)| > 1$ **do**
9: Randomly select two jobs i_1, i_2 from $Y_{\mathbb{R}}^r(t)$;
10: Define $\Psi_1 \triangleq \min\{1 - p_{i_1}(t), \frac{\omega_{i_2}}{\omega_{i_1}} p_{i_2}(t)\}$;
11: Define $\Psi_2 \triangleq \min\{p_{i_1}(t), \frac{\omega_{i_2}}{\omega_{i_1}}(1 - p_{i_2}(t))\}$;
12: With probability $\frac{\Psi_2}{\Psi_1+\Psi_2}$, set
13: $p_{i_1}(t) = p_{i_1}(t) + \Psi_1, p_{i_2}(t) = p_{i_2}(t) - \frac{\omega_{i_1}}{\omega_{i_2}} \Psi_1$;
14: With probability $\frac{\Psi_1}{\Psi_1+\Psi_2}$, set
15: $p_{i_1}(t) = p_{i_1}(t) - \Psi_2, p_{i_2}(t) = p_{i_2}(t) + \frac{\omega_{i_1}}{\omega_{i_2}} \Psi_2$;
16: **if** $p_{i_1}(t) = 0$ or $p_{i_1}(t) = 1$ **then**
17: Set $\bar{y}_i^r(t) = p_{i_1}(t) + \lfloor \tilde{y}_i^r(t) \rfloor$, $Y_{\mathbb{R}}^r(t) = Y_{\mathbb{R}}^r(t) \setminus \{i_1\}$;
18: **end if**
19: **if** $p_{i_2}(t) = 0$ or $p_{i_2}(t) = 1$ **then**
20: Set $\bar{y}_i^r(t) = p_{i_2}(t) + \lfloor \tilde{y}_i^r(t) \rfloor$, $Y_{\mathbb{R}}^r(t) = Y_{\mathbb{R}}^r(t) \setminus \{i_2\}$;
21: **end if**
22: **end while**
23: **if** $|Y_{\mathbb{R}}^r(t)| = 1$ **then**
24: Set $\bar{y}_i^r(t) = \lceil \tilde{y}_i^r(t) \rceil$ for the only $i \in Y_{\mathbb{R}}^r(t)$;
25: **end if**
26: **end for**
27: **return** $\bar{\boldsymbol{y}}(t)$

Step 2: Since $\sum_{i\in[R]} \tilde{s}_i^r = 1$ (constraint (8f)), we use $\tilde{s}_i^r, \forall r \in [R]$ as a probability distribution, and select one $r_i \in [R]$ accordingly. We set $\bar{s}_i^{r_i}(t) = 1$ and $\bar{s}_i^r(t) = 0$ if $r \neq r_i$ (lines 6-9), for each job $i \in I_t$.

Step 3: Given the rounded values of $\boldsymbol{y}$ and $\boldsymbol{s}$, we compute values of the other integer and real variables as follows (line 10), to minimize $\boldsymbol{P}_{rt}$ ($\boldsymbol{P}_{rt}$ is almost the same with (10) except that the values of $\boldsymbol{y}(t)$ and $\boldsymbol{s}(t)$ are known).

– We compute real-valued $\boldsymbol{q}(t)$ by minimizing $C_1(t)$ under constraints (8a) and (8b), which is a linear program.

– We compute real-valued $\boldsymbol{x}(t)$ and integers $\tilde{\boldsymbol{u}}^{\boldsymbol{1}}(\boldsymbol{t})$, $\tilde{\boldsymbol{u}}^{\boldsymbol{2}}(t)$ under constraints (8c)-(8e), to minimize $C_2^*(t) = \sum_{r\in[R]} \sum_{k\in[K]} x_k^r(t)$. The minimal $x_k^r(t)$ is $\min\{a_k^r \sum_{i\in I_t}(n_{i,k} y_i^r(t) + m_{i,k} s_i^r(t)), b_k^r \sum_{i\in I_t}(n_{i,k} y_i^r(t) + m_{i,k} s_i^r(t)) + e_k^r\}$; $u_{k,r}^1(t) = 1$, $u_{k,r}^2(t) = 0$, if $a_k^r \sum_{i\in I_t}(n_{i,k} y_i^r(t) + m_{i,k} s_i^r(t)) > b_k^r \sum_{i\in I_t}(n_{i,k} y_i^r(t) + m_{i,k} s_i^r(t)) + e_k^r$, and $u_{k,r}^1(t) = 0$, $u_{k,r}^2(t) = 1$, otherwise.

– $\bar{\boldsymbol{v}}(t)$ is decided as the smallest binary values satisfying constraint (8g), to minimize the regularization function in (10).

– Integers $\bar{\boldsymbol{g}}$ are computed according to $g_i^{rr'}(t) = y_i^r(t) \times s_i^{r'}(t)$, which satisfy constraints (8i) and (8j).

5.2 Integrality Gap Analysis

Next we first prove the rationality of the obtained integer solutions, and then prove the performance ratio of our online algorithm *mlBroker*.

Lemma 1. *The solution of ($\bar{\boldsymbol{y}}$, $\bar{\boldsymbol{s}}$, $\bar{\boldsymbol{u}}^{\boldsymbol{1}}$, $\bar{\boldsymbol{u}}^{\boldsymbol{2}}$, $\bar{\boldsymbol{v}}$, $\bar{\boldsymbol{g}}$, $\boldsymbol{q}, \boldsymbol{x}$) is feasible to problem (8).*

Proof. As $\tilde{\boldsymbol{y}}(t)$ exists, $\bar{\boldsymbol{y}}(t)$ always exists since it is a feasible solution of executing $Round1\_y$ and $Round2\_y$. During each iteration in $Round1\_y$ (Alg. 3), no matter how line 13 and line 15 are executed, the sum of $\omega_{i_1,r}\widetilde{y_{i_1}^r}(t) + \omega_{i_2,r}\widetilde{y_{i_2}^r}(t)$ maintains the same. For example, in the case of Line 13, we have $\omega_{i_1,r}(\widetilde{y_{i_1}^r}(t) + \Psi_1) + \omega_{i_2,r}(\widetilde{y_{i_2}^r}(t) - \frac{\omega_{i_1,r}}{\omega_{i_2,r}}\Psi_1) = \omega_{i_1,r}\widetilde{y_{i_1}^r}(t) + \omega_{i_2,r}\widetilde{y_{i_2}^r}(t)$, which means that we have $\sum_{i\in I_t\setminus Y_{\mathbb{R}}^r(t)}\omega_{i,r}\bar{y}_i^r(t) = \sum_{i\in I_t\setminus Y_{\mathbb{R}}^r(t)}\omega_{i,r}\widetilde{y_i^r}(t)$ after the loop of Lines 8-22. We can also get $\sum_{i\in I_t}\omega_{i,r}\bar{y}_i^r(t) = \sum_{i\in I_t\setminus Y_{\mathbb{R}}^r(t)}\omega_{i,r}\bar{y}_i^r(t) + \sum_{i\in Y_{\mathbb{R}}^r(t)}\omega_{i,r}\bar{y}_i^r(t) \geq \sum_{i\in I_t}\omega_{i,r}\widetilde{y_i^r}(t)$ after executing Lines 23-25. In $Round2\_y$, we can similarly get $\sum_{r\in[R]}(\widetilde{y_i^{r_1}}(t) + \widetilde{y_i^{r_2}}(t)) \geq \sum_{r\in[R]}(\bar{y}_i^{r_1}(t) + \bar{y}_i^{r_2}(t))$.

Since all the round-down $y_i^r(t)$ can be compensated with round-up ones, the total processing capacity of all workers of job i in t, $\sum_{r\in[R]}P_i y_i^r(t)$, is still no smaller than the total amount of data to be processed in t to guarantee the feasible of constraint (8a) and (8b). Therefore, feasible values of $q_i^{rr'}(t)$'s satisfying constraints (8a) and (8b) can be found. Also, since $\tilde{\boldsymbol{s}}(t)$ exists, $\bar{\boldsymbol{s}}(t)$ always exits since it is a feasible solution of executing Alg. 2. We then calculate the rest of variables according to $\tilde{\boldsymbol{y}}(t)$ and $\tilde{\boldsymbol{s}}(t)$. Given the value of $\bar{\boldsymbol{y}}(t)$ and $\bar{\boldsymbol{s}}(t)$, the minimum of the two segments in piecewise functions is determined. For example, if the value of the first segment is smaller than the second one for specific k, r and t, then we set $\bar{\boldsymbol{u}}_{k,r}^1(t) = 0$, $\bar{\boldsymbol{u}}_{k,r}^2(t) = 1$ and $x_k^r(t) = a_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t) + m_{i,k}s_i^r(t))$, and the constraints (8b)-(8d) are satisfied. The solution of $\bar{\boldsymbol{v}}(t)$ can be determined with the value of $\bar{\boldsymbol{y}}(t)$ and $\bar{\boldsymbol{s}}(t)$. Since the value of $g_i^{rr'}(t)$ can be determined by $g_i^{rr'}(t) = y_i^r(t) * s_i^{r'}(t)$, the constraints (8i) and (8j) are satisfied. Therefore, all the constraints remain feasible for each time slot T. The lemma is proved. □

We next give the approximation ratio of A_{round}, an upper bound of the ratio of the objective value of P_f in t computed by solutions produced by A_{round} to the objective value of P_f in t computed by solutions returned by $A_{fractional}$ (*i.e.*, we use solutions derived by solving $\widetilde{P}_{ft}$ to compute values of the original objective function in P_f).

Theorem 4. *The approximation ratio of A_{round} is $r_2 = \varrho_1 + \varrho_2 + \varrho_3 + \varrho_4$, where*

$$\varrho_1 = \max_{r'\in[R],r\in[R]}(1+\lambda)d^{rr'},$$

$$\varrho_2 = \min\{\max_{t\in[T],i\in I_t,r\in[R],k\in[K]}\frac{a_k^r((1+\lambda)n_{i,k} + \frac{m_{i,k}}{\varpi_{it}})}{P_i},$$
$$\max_{t\in[T],i\in I_t,r\in[R],k\in[K]}\frac{(b_k^r + e_k^r)((1+\lambda)n_{i,k} + \frac{m_{i,k}}{\varpi_{it}})}{P_i}\},$$

$$\varrho_3 = \max_{t\in[T],i\in I_t,}\frac{(2+\lambda)c_i}{P_i\varpi_{it}}, \varrho_4 = (1+\lambda)\max_{i\in[I]}\frac{B_i}{P_i}.$$

where $\lambda = \max_{t\in[T],i\in I_t}\lambda_{it}$, $\lambda_{it} = \frac{P_i(|R|+1)}{\min_{r\in[R]}D_i^r(t)}, \forall t\in[T], i\in I_t$, and $\varpi_{it} = \lceil\frac{\sum_{r\in[R]}D_i^r(t)}{P_i}\rceil, \forall t\in[T], i\in I_t$.

Proof. We need to mention two important properties achieved by the $Round\_y$ here, which will be utilized in the approximation ratio analysis. Let $\bar{p}_{i,r}(t) \in \{0,1\}$ be a binary random variable indicating the rounded value of $p_{i,r}(t)$ produced by $Round_y$.

(P1) Marginal Distribution Property. The probability $p_{i,r}(t)$ of each element $i \in Y_{\mathbb{R}}^r(t)$ satisfies $\Pr[\bar{p}_{i,r}(t) = 1] = p_{i,r}(t), \forall r\in[R], t\in[T]$.

(P2) Weight Preservation Property. The rounded probability $\bar{p}_{i,r}(t)$ and the corresponding weight ω_i of each element $i \in Y_{\mathbb{R}}^r(t), \forall r \in [R], t \in [T]$ satisfy $\sum_{i\in Y_{\mathbb{R}}^r(t)}\omega_i p_{i,r}(t) = \sum_{i\in Y_{\mathbb{R}}^r(t)}\omega_{ir}\bar{p}_{i,r}(t)$, $\omega_i = P_i$ for constraint (8a) and $\omega_i = n_{i,k}$ for constraint (8c).

The connection between the values of $\sum_{r\in[R]}P_i y_i^r(t)$ before and after rounding $\tilde{y}_i^r(t)$'s, are given as follows. For $\bar{y}_i^r(t)$ and $\widehat{y}_i^r(t)$, $\forall i \in I_t, r \in [R], t \in [T]$, produced by invoking A_{round} and $Round2\_y$ respectively, with $\tilde{\boldsymbol{y}}(t)$ as arguments, we have

$$\begin{aligned}
&\sum_{r\in[R]}P_i\bar{y}_i^r(t) \leq \sum_{r\in[R]}P_i(\widehat{y}_i^r(t)+1)\\
=&\sum_{r\in[R]\setminus Y_{\mathbb{R}}^i(t)}P_i\widehat{y}_i^r(t) + \sum_{r\in Y_{\mathbb{R}}^i(t)}P_i\widehat{y}_i^r(t) + \sum_{r\in[R]}P_i\\
\leq&\sum_{r\in[R]\setminus Y_{\mathbb{R}}^i(t)}P_i\widetilde{y_i^r}(t) + \sum_{r\in[R]}P_i\lfloor\widetilde{y_i^r}(t)\rfloor\\
&+\sum_{r\in[R]}P_i\lceil\widetilde{p_{i,r}}(t)\rceil + \sum_{r\in[R]}P_i\\
\leq&\sum_{r\in[R]}P_i\widetilde{y_i^r}(t) + \max_{r\in[R]}P_i(|R|+1)\\
=&\sum_{r\in[R]}P_i\widetilde{y_i^r}(t) + \lambda_{it}\min_{r\in[R]}D_i^r(t)\\
=&\sum_{r\in[R]}P_i\widetilde{y_i^r}(t) + \lambda_{it}\min_{r\in[R]}\sum_{r'\in[R]}q_i^{rr'}(t)\\
\leq&\sum_{r\in[R]}P_i\widetilde{y_i^r}(t) + \lambda_{it}P_i\widetilde{y_i^r}(t)\\
\leq&(1+\lambda)\sum_{r\in[R]}P_i\widetilde{y_i^r}(t) \qquad (14)
\end{aligned}$$

where $\lambda = \max_{t\in[T],i\in I_t}\lambda_{it}$, and $\lambda_{it} = \frac{P_i(|R|+1)}{\min_{r\in[R]}D_i^r(t)}$, $\forall t \in [T], i \in I_t$. When substituting P_i with $n_{i,r}$ through the same derivation process, we can get $\sum_{i\in I_t}n_{i,r}\bar{y}_i^r(t) \leq (1+\lambda)\sum_{i\in I_t}n_{i,r}\widetilde{y_i^r}(t)$.

We bound all the costs, which are computed by the final integral solution ($\bar{\boldsymbol{s}}$, $\bar{\boldsymbol{M}}$, $\bar{\boldsymbol{y}}$, $\bar{\boldsymbol{v}}$, $\bar{\boldsymbol{g}}$), based on the above properties. Let $\widetilde{P}(mlBroker)$ denote the objective value of (10) by its fractional solution. Let $\mathbb{E}[C_1]$, $\mathbb{E}[C_2]$, $\mathbb{E}[C_3]$ and $\mathbb{E}[C_4]$ denote the expectation of four types of cost with integral solutions respectively.

$$\mathbb{E}[C_1] = \sum_{t\in[T]}\sum_{i\in[I]}\sum_{r\in[R]}\sum_{r'\in[R]}d_i^{rr'}q_i^{rr'}(t) \qquad (15)$$

$$\leq\sum_{t\in[T]}\sum_{i\in[I]}\sum_{r'\in[R]}\max_{r\in[R]}d_i^{rr'}P_i\bar{y}_i^{r'}(t) \qquad (15a)$$

$$\leq(1+\lambda)\sum_{t\in[T]}\sum_{r'\in[R]}\sum_{i\in I_t}\max_{r\in[R]}d_i^{rr'}P_i\widetilde{y_i^{r'}}(t) \qquad (15b)$$

$$\leq\varrho_1\sum_{t\in[T]}\sum_{r'\in[R]}\sum_{i\in I_t}P_i\widetilde{y_i^{r'}}(t) \qquad (15c)$$

where $\varrho_1 = \max_{r\in[R],r'\in[R],i\in[I]}(1+\lambda)d_i^{rr'}$. (15a) holds due to constraint (8a) and (15b) follows from (14).

$$\begin{aligned}
&\mathbb{E}(C_2)\\
=&\mathbb{E}[\sum_{t\in[T]}\sum_{r\in[R]}\sum_{k\in[K]}F_k^r(\sum_{i\in I_t}(n_{i,k}\bar{y}_i^r(t) + m_{i,k}\bar{s}_i^r(t)))] \qquad (16)
\end{aligned}$$

$$= \sum_{t\in[T]}\sum_{r\in[R]}\sum_{k\in[K]} F_k^r\Big(\sum_{i\in I_t}(\mathbb{E}[n_{i,k}\bar{y}_i^r(t)] + m_{i,k}\mathbb{E}[\bar{s}_i^r(t)])\Big) \quad (16a)$$

$$\le \sum_{t\in[T]}\sum_{r\in[R]}\sum_{k\in[K]} F_k^r\Big(\sum_{i\in I_t}\big((1+\lambda)n_{i,k}\widetilde{y}_i^r(t) + m_{i,k}\widetilde{s}_i^r(t)\big)\Big) \quad (16b)$$

$$\le \min\{\sum_{t\in[T]}\sum_{r\in[R]}\sum_{k\in[K]}\sum_{i\in I_t}\frac{a_k^r((1+\lambda)n_{i,k}+\frac{m_{i,k}}{w_i})}{P_i}P_i\widetilde{y}_i^r(t),$$
$$\sum_{t\in[T]}\sum_{r\in[R]}\sum_{k\in[K]}\sum_{i\in I_t}\frac{(b_k^r+e_k^r)\big((1+\lambda)n_{i,k}+\frac{m_{i,k}}{w_i}\big)}{P_i}P_i\widetilde{y}_i^r(t)\} \quad (16c)$$

$$\le \varrho_2\widetilde{P}(mlBroker) \quad (16d)$$

where $\varrho_2 = \min\{\max_{t\in[T],r\in[R],k\in[K],i\in I_t}\frac{a_k^r((1+\lambda)n_{i,k}+\frac{m_{i,k}}{w_i})}{P_i}$ $\max_{t\in[T],r\in[R],k\in[K],i\in I_t}\frac{(b_k^r+e_k^r)((1+\lambda)n_{i,k}+\frac{m_{i,k}}{w_i})}{P_i}\}$ and $w_i = \lceil\frac{\sum_{r\in[R]}D_i^r(t)}{P_i}\rceil$. (16a) follows from the property of expectation. (16b) follows from (14) and the definition of the probability of $\bar{s}_i^r(t)$. (16c) holds since $F_k^r\big(\sum_{i\in I_t}\big(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)\big)\big)=\min\{a_k^r\sum_{i\in I_t}(n_{i,k}y_i^r(t) + m_{i,k}s_i^r(t)),\quad b_k^r\sum_{i\in I_t}\big(n_{i,k}y_i^r(t)+m_{i,k}s_i^r(t)\big) + e_k^r\}$ and $s_i^r(t) \le \frac{y_i^r(t)}{\lceil\frac{\sum_{r\in[R]}D_i^r(t)}{P_i}\rceil}$. We assume that for each time slot t, there always exits a job placing its worker or PS in the rth data center, which is feasible if the number of jobs is very large per time slot, thus $e_k^r \le e_k^r(\sum_{i\in I_t}n_{i,k}y_i^r(t) + m_{i,k}s_i^r(t))$. The reason for (16d) is the same for (15c).

$$\mathbb{E}[C_3] = \mathbb{E}\Big[\sum_{t\in[T]}\sum_{r\in[R]}\sum_{i\in I_t}c_i\bar{z}_i^r(t)\Big] \quad (17)$$

$$=\mathbb{E}\big[\sum_{t\in[T]}\sum_{r\in[R]}\sum_{i\in I_t}c_i(\bar{v}_i^r(t)-\bar{v}_i^r(t-1))\big] \quad (17a)$$

$$\le\mathbb{E}\big[\sum_{t\in[T]}\sum_{r\in[R]}\sum_{i\in I_t}c_i\bar{v}_i^r(t)\big] \quad (17b)$$

$$=\mathbb{E}\big[\sum_{t\in[T]}\sum_{r\in[R]}\sum_{i\in I_t}(c_i\bar{y}_i^r(t)+c_i\bar{s}_i^r(t))\big] \quad (17c)$$

$$\le \sum_{t\in[T]}\sum_{r\in[R]}\sum_{i\in I_t}\frac{(2+\lambda)c_i}{P_iw_i}P_i\widetilde{y}_i^r(t) \quad (17d)$$

$$\le\varrho_3\widetilde{P}(mlBroker) \quad (17e)$$

where $\varrho_3 = \max_{t\in[T],i\in[I]}\frac{(2+\lambda)c_i}{P_iw_i}$ and $w_i = \lceil\frac{\sum_{r\in[R]}D_i^r(t)}{P_i}\rceil$. (17a), (17b) and (17c) holds due to the definition of $z_i^r(t)$ and $v_i^r(t)$. The reasons for (17d) and (17e) are similar with (16c) and (16d).

$$\mathbb{E}[C_4] = \mathbb{E}\Big[\sum_{t\in[T]}\sum_{i\in[I]}\sum_{r\in[R]}\sum_{r'\in[R]}d_i^{rr'}B_i\bar{g}_i^{rr'}(t)\Big] \quad (18)$$

$$\le\mathbb{E}\Big[\sum_{t\in[T]}\sum_{i\in[I]}\sum_{r\in[R]}\frac{B_i}{P_i}P_i\bar{y}_i^r(t)\Big] \quad (18a)$$

$$\le\varrho_4\mathbb{E}[C_1] \quad (18b)$$

$$\le\varrho_4\widetilde{P}(mlBroker) \quad (18c)$$

where $\varrho_4 = (1+\lambda)\max_{i\in[I]}\frac{B_i}{P_i}$. (18a) holds due to the constraint (8i) and the reasons for (18b) and (18c) are similar to the former ones. □

5.3 Analysis for Our Complete Algorithm

We now show the competitive ratio of our complete online algorithm, referred to as $mlBroker$, which carries out $A_{fractional}$ and then A_{round} in each time slot t. The competitive raio is an upper-bound ratio of the objective value of (8) achieved by solutions produced by the online algorithm, to the offline optimum of (8).

Lemma 2. *The running time of $Round1\_y$ and $Round2\_y$ is $O(I_tR)$.*

Proof. Within each *For* loop in the outermost layer: Line 2 takes constant step to initialize a set $Y_{\mathbb{R}}^r(t)$. Lines 3-7 take I_t steps to finish the assignment tasks. The *While* loop (lines 8-22) take at most $O(\frac{|Y_{\mathbb{R}}^r(t)|(|Y_{\mathbb{R}}^r(t)|-1)}{2})$ steps to terminate. Lines 23-25 can be done in constant steps. In summary, the running time of $Round1\_y$ and $Round2\_y$ is $O(I_tR)$. □

Theorem 5. *Our complete online algorithm $mlBroker$ runs in polynomial time in each time slot, and achieves the competitive ratio*

$$r = r_1\cdot r_2 = \big(2+(1+\epsilon)\ln(1+\tfrac{1}{\epsilon})\big)\cdot\big(\varrho_1+\varrho_2+\varrho_3+\varrho_4\big).$$

Proof. The main body in $A_{fractional}$ uses the interior point method [35] to solve $\widetilde{P}_{ft}$ in (10), which runs in polynomial time. According to Lemma 2, the running time of $Round1\_y$ and $Round2\_y$ is $O(I_tR)$. In A_{round}, lines 1-5 can be finished in $O((K+1)I_tR)$ steps to set $\bar{\boldsymbol{y}}$. Lines 6-9 take $O(I_t)$ steps to choose where to place parameter servers. Line 10 involves solving a linear program, which can be solved by interior point method in polynomial time [35]. Hence the algorithm runs in polynomial time in each t.

We have $\frac{\text{objective value of (8) mlBroker achieves}}{\text{offline optimum of (8)}} \le r_1\times r_2 = \frac{\text{objective value of } P_f \text{ computed by } A_{fractional}\text{'s solution}}{\text{offline optimum of relaxed } P_f}\times \frac{\text{objective value of } P_f \text{ computed by } A_{round}\text{'s solution}}{\text{objective value of } P_f \text{ computed by } A_{fractional}\text{'s solution}}$, since the objective functions in P_f and in (8) achieve the same values under the same set of solutions, and the offline optimum of (8) is no smaller than the offline optimum of relaxed P_f. Hence, $r_1\times r_2$ is an upper bound of the ratio of the objective value of (8) achieved by $mlBroker$ to the offline optimum of (8), and thus a competitive ratio of $mlBroker$. By Theorem 3 and Theorem 4, the competitive ratio r holds. □

6 Performance Evaluation

In this section, we evaluate the performance of our online algorithms through simulation studies (in Sec. 6.1- Sec. 6.3) and real experiments (in Sec. 6.4).

6.1 Simulation Settings

We simulate a geo-distributed ML system running for $T\in[50,100]$ time slots ($T=50$ by default); each time slot is half day long, which can be set to $2-3$ hours according to realistic demands. The default number of data centers in this system is 15. Following similar settings in [4], [44] and [3], we set the resource demand of each worker as follows: $0-4$ GPUs, $1-10$ vCPUs, $2-32$ GB memory and $5-10$ GB storage; we also set the resource demand of each parameter server as $1-10$ vCPUs, $2-32$ GB memory and $5-10$ GB storage. We

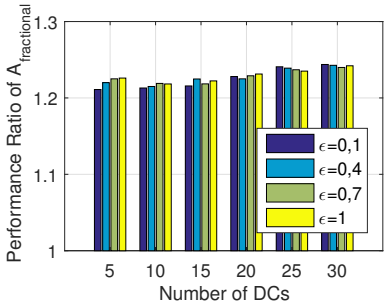


Fig. 5: Performance ratio of $A_{fractional}$ under different numbers of DCs and ϵ.

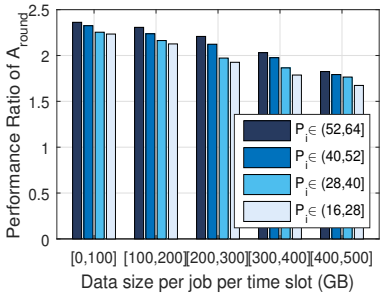


Fig. 6: Performance ratio of A_{round} under different P_i and data sizes.

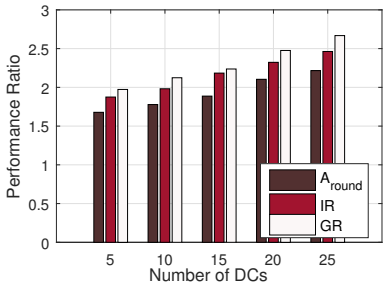


Fig. 7: Comparison of A_{round}, IR and GR.

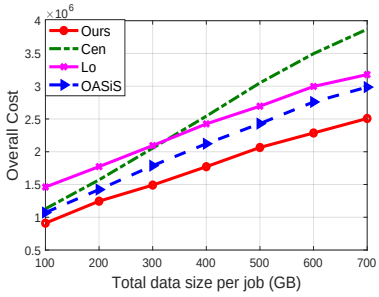


Fig. 8: Overall cost comparison under different data sizes per job per time slot.

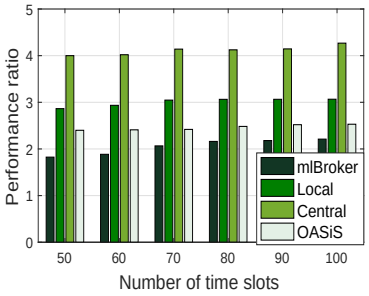


Fig. 9: Performance ratio under different time slots.

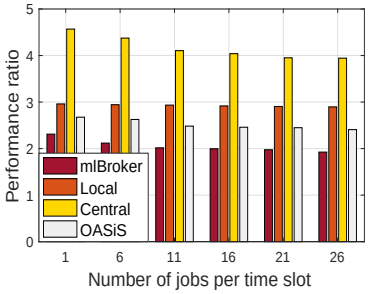


Fig. 10: Performance ratio with different numbers of jobs per slot.

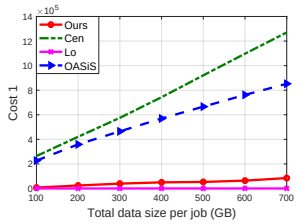


Fig. 11: Cost1 under different data sizes per job per time slot.

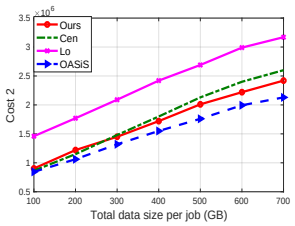


Fig. 12: Cost2 under different data sizes per job per time slot.

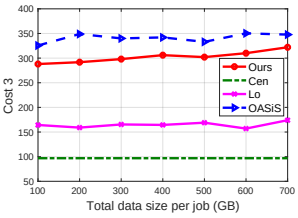


Fig. 13: Cost3 under different data sizes per job per time slot.

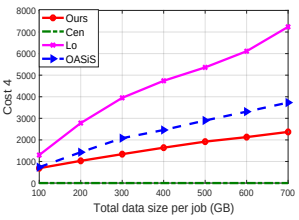


Fig. 14: Cost4 under different data sizes per job per time slot.

set job arrival patterns according to the Google cluster data [45]. For each job, the processing capacity of a worker ranges in 16–66 GB (according to the training speed for ResNet-152 shown in the performance benchmarks of TensorFlow [46]). The size of the total training data accumulated during each time slot in a job is in $[100, 700]$ GB (500 GB by default) (according to size of pictures uploaded at Instagram per minute [47]). We set the transmission data size between each pair of worker and PS in $[30, 575]$ MB [48], thus the overall transmission size per time slot, B_i, is set in $[4.32, 82.8]$ GB. The unit data transmission cost is in $[0.01, 0.02]$ USD per GB [49]. The deployment cost for each job is set in $[0.05, 0.1]$ USD per GB. The unit prices of resources in different data centers before volume discount, a_k^r, are set in $[1.2, 9.6]$, $[0.13, 0.24]$, $[0.01, 0.1]$, $[0.01, 0.1]$ USD per day, respectively [49]. The discounted unit prices, b_k^r, are within $[70\%, 80\%]$ of the respective price before discount [8]. The thresholds, l_k^r, are set within $[500, 600]$, $[800, 1000]$, $[1000, 1050]$, and $[1000, 1050]$ for the respective resource types [8].

6.2 Performance of $A_{fractional}$ and A_{round}

We first study the performance ratio of $A_{fractional}$, which is the ratio of the overall cost of P_f generated by $A_{fractional}$ to the cost produced by the optimal solution of P_f (we do not refer to it as the competitive ratio, as the later is derived under the worst case). Fig. 5 illustrates that $A_{fractional}$ performs well with a small performance ratio (< 1.25). The results confirm that there is only a small loss in the performance when we decompose the online problem into a series of one-shot problems. Furthermore, the number of DCs and ϵ have little impact on the performance. Although Theorem 3 indicates that the worst case ratio is larger with smaller ϵ. The average-case value, as we observed in our simulation, shows that its impact is not obvious.

We next examine the performance ratio of our one-shot rounding algorithm A_{round}, which is the ratio of the objective value of P_f computed by solutions of A_{round} to that computed by solutions of $A_{fractional}$ (we do not refer to it as approximation ratio as the latter is evaluated under worst case). Fig 6 shows that the ratio decreases with the increase of data size and the decrease of worker's processing capacity. This can be explained as follows: as proved in Theorem 3, P_i appears in the numerator and $D_i^r(t)$ appears

in the denominator of parameter λ. Therefore, a low ratio comes with a smaller P_i and a larger $D_i^r(t)$.

We further compare our algorithm with two other rounding algorithms: i) *Independent Rounding (IR)*, which rounds each $y_i^r(t)$ to the nearest integer (*e.g.*, 2.6 to 3, 2.1 to 2) and rounds $s_i^r(t)$ in the same way as in A_{round}; ii) *Greedy Rounding (GR)*, which rounds up $y_i^r(t)$ (*e.g.*, 2.2 to 3) and assigns $s_i^{r^*}(t) = 1, r^* = \arg\max_{r\in[R]} s_i^r(t)$. Fig.7 demonstrates that our algorithm consistently outperforms the two algorithms under different numbers of data centers.

6.3 Performance of our complete online algorithm

Algorithms for comparison. We compare our complete online algorithm $mlBroker$ with four job placement algorithms. i) *OASiS (OA)* [4]: given unit resource prices, each job selects the best placement scheme which minimizes its payment cost. ii) *Local (Lo)*: dataset is processed locally in the data center where it is collected, with workers deployed in the data center, and the PS is randomly placed in one of the data centers. Therefore, $C_1(t) = 0$. iii) *Central (Cen)*: Each job's PS and workers are placed into one data center which incurs the lowest cost, and all data are aggregated into that data center for processing. iv) *Optimum (Opt)*, which is the minimum cost obtained by solving P_f.

Fig. 8 shows the total cost generated by different algorithms, when we vary the size of each job's per-time-slot training data. We can observe that when the dataset is large, it is better to train it locally to avoid high data transfer costs; on the other hand, centralized training is preferred with small datasets. Nevertheless, our algorithm always generates a lower cost, compared to other algorithms. Fig. 11, Fig. 12, Fig. 13 and Fig. 14 show each component of the overall cost under different algorithms. As can be seen in Fig. 11, with the increment data size per time slot, data transfer cost increases, especially for *Central* and *OASiS*. This shows the weakness of centralization when meeting geo-distributed training datasets. Our algorithm *mlBroker* selectively transfers datasets, leading to a better performance. Resource cost accounts for a large part of the overall cost. In Fig. 12, we can see that when the data size is small, training in a centralized method leads to a lower payment. However, when the data size gets larger, our method performs better than *Local* and *Central* since we balance between these two methods. *OASiS* shows the best performance with the lowest overall cost. As shown in Fig. 13, the deployment cost keeps stable for each algorithm even when the data size varies. The difference between them is small compared to the total cost. In Fig. 14, our algorithm also shows a good performance with a lower communication cost compared with *Local* and *OASiS*. This is due to our efficient worker/PS placement decisions.

Fig. 9 and Fig. 10 show the performance ratio of $mlBroker$ with large size of training datasets, which is the ratio of the overall cost in (8) generated by $mlBroker$ to the cost produced by optimal solution of (8). In Fig. 9, we can observe that the ratios become slightly large when the number of time slots increases. This is mainly due to the performance loss incurred when we decouple decisions over the time slots and make decisions in each slot, and the more time slots there is, the more loss results. Fig. 10 shows that the ratio of $mlBroker$ remains stable when the number of jobs per slot grows. In addition, we still observe the better performance of our online algorithm. However, our algorithm performs the best in all cases.

6.4 Real Experiment

In this section, we evaluate the performance of our online algorithms on geo-distributed GPU clusters which are managed by kubernets 1.7. We choose parameter severs in MXNet as the distributed training framework. We created three clusters in three different availability zones on the Amazon cloud. Each cluster is interconnected with high-bandwidth, low-latency networking, dedicated metro fiber. The unit data transmission cost between the cluster is about 0.02 USD per GB. We use p3.2xlarge instances for each cluster. Each node is equipped with 8-core vCPUs, 61GB RAM, 40GB SSD storage space and a Nvidia Tesla V100 GPU with 16GB RAM. The network bandwidth across nodes is 10Gbps. The instance price before the discount is 3.06 USD per hour. Due to resource and economic constraints, we only conducted small-scale verification experiments and it is difficult for us to reach the amount that triggers the Amazon discount price, so we set the discounted prices and thresholds by ourselves. The final cost is calculated based on the running time of each instance and data transmission volume.

TABLE 3: Deep learning jobs for experiments

Model	Params	Dataset	Batch size
AlexNet	61.1M	ImageNet-12	128
VGG-16	138M	ImageNet-12	64
VGG-19	143M	ImageNet-12	64
Inception-V3	27M	ImageNet-12	128
ResNet-152	60.2M	ImageNet-12	128

Workload. We set total time slots to 10 and job arrives randomly at each time slot. Upon an arrival event, we randomly choose the job among the example in Table 3 and set the duration of the job to $3-8$ time slots. We use ImageNet as the dataset for all jobs, which contains about 1.28 million training images. During the duration of the job, we randomly send part of the ImageNet dataset to three clusters at every time slot. The number of workers and PSs required by the job is determined by the amount of data generated and the job type at current time slot. The scheduling algorithm determines data transfer and deployment decisions of workers and PSs.

TABLE 4: Detailed costs under different algorithms

Algorithm	Cost1	Cost2	Cost3	Cost4	Overall Cost
mlBroker	2.8	1018.3	14.5	34.5	1070.1
Local	0	1253.8	13.2	119.1	1386.1
Central	3.9	1083.5	12.4	0	1099.8
OASiS	3.2	1129.3	17.8	50.7	1201.0

Table 4 shows each component of the overall cost under different algorithms in our real experiments. As we can see, resource cost accounts for the largest part of the overall cost

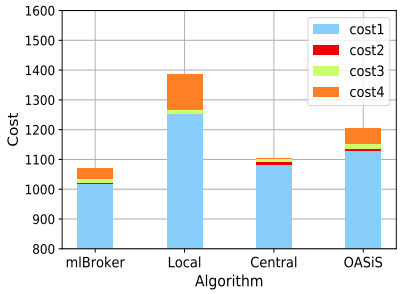


Fig. 15: Cost for different Algorithm.

and transfer cost is the smallest cost. The main reason is that due to the resource and economic constraints, the size of each job's total training data and the number of DCs in our verification experiments is very small, compared to the simulation settings. This leads to a small amount of data transfer between data centers. As shown in Fig. 15, the performance of our algorithm is better than others, and this proves the effectiveness of our algorithm, even with small scale of input.

TABLE 5: Components of Resource Cost (Cost2).

Algorithm	mlBroker	Local	Central	OASiS
Running Time	392.2	417.7	388.6	396.7
Discount Rate	0.85	0.98	0.91	0.93
Resource Cost (= time × price)	1018.3	1253.8	1083.5	1129.3

Table. 5 shows the total running time of all instances and the total discount rate under different algorithms, which are related to resource costs (resource cost = total running time × price with discount). As can be seen from Table. 5, Fig. 16 and Fig. 17, $Central$ has the minimal running time while our algorithm $mlBroker$ triggers the largest overall discount. The reason why $Central$ has the minimal running time is that all the instances are centralized, thus the time spent on waiting for communications between workers and PSs is minimized. Moreover, we compute the ratio of total resource payment with discount to total payment without discount. Our algorithm shows the best performance by taking running time and discounts together into conditions, which leads to the smallest resource cost under these small-scale verification experiments. When the size of datasets increases, $mlBroker$ always obtains minimal total cost (see Fig. 8).

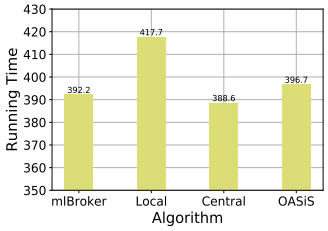


Fig. 16: Total running time of each algorithm.

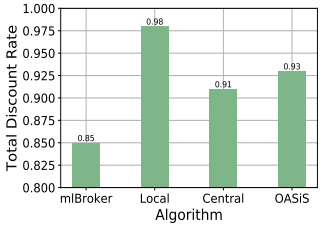


Fig. 17: Discount rate of each algorithm.

7 Concluding Remarks

This paper proposes a machine learning broker service which strategically rents resources from different data centers to serve users' requests for geo-distributed machine learning. We propose an efficient online algorithm for the broker to deploy new jobs and adjust existing jobs' placement over time, to maximally exploit volume based discounts for overall cost minimization. The online algorithm $mlBroker$ consists of two main components: (i) an online regularization algorithm that coverts the online deployment problem into a sequence of one-shot optimization problems, and each can be solved to a set of fractional solutions directly; (ii) a dependent rounding algorithm which rounds the fractional solutions to feasible integer solutions. Through extensive theoretical analysis and simulation studies, we verify our online algorithm's good performance as compared to both the offline optimum and representative alternatives.

Note that our algorithmic framework can be readily adapted to handle additional constraints in geo-distributed job placement, *e.g.*, locally generated data cannot be transferred out of a region due to security constraints [14]. In addition, we focus on broker scheduling strategy in this paper, which can readily work with any detailed design of broker pricing strategy to charge the users [50][51][52]. Especially, our cost minimization exploiting volume discounts easily guarantees that the prices the broker charges the users are no higher than what the users need to pay, if they directly rent resources from the cloud providers. We can assume that the broker charges each user just at the middle of the discounted price and the non-discounted price, which can cover all the costs and is profitable for the ML broker. There are already many pricing models proposed for brokers, which has been well studied in [50], [51] and [52]. We will continue to study this topic in our future work.

References

[1] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*. Springer, 2007, pp. 325–341.
[2] V. Potluru, J. Diaz-Montes, A. D. Sarwate, S. M. Plis, V. D. Calhoun, B. Pearlmutter, and M. Parashar, "Comet-cloudcare (c3): distributed machine learning platform-as-a-service with privacy preservation," in *Proc. of NIPS*, 2014.
[3] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. of OSDI*, 2014.
[4] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. of IEEE INFOCOM*, 2018.
[5] R. Polikar, J. Byorick, S. Krause, A. Marino, and M. Moreton, "Learn++: A classifier independent incremental learning algorithm for supervised neural networks," in *Proc. of IEEE IJCNN*, 2002.
[6] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental learning of object detectors without catastrophic forgetting," in *Proc. of IEEE ICCV*, 2017.
[7] S. S. Sarwar, A. Ankit, and K. Roy, "Incremental learning in deep convolutional neural networks using partial network sharing," *arXiv preprint arXiv:1712.02719*, 2017.
[8] *Rackspace*, http://alturl.com/bakec.
[9] *Amazon EC2*, http://alturl.com/y5vch.

[10] *Telecoms Cloud*, https://www.telecomscloud.com/volume-discounts/.

[11] N. Wang and J. Wu, "Optimal cloud instance acquisition via iaas cloud brokerage with volume discount," in *Proc. of IEEE/ACM IWQOS*, 2018.

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. of ICML*, 2010.

[13] I. Cano, M. Weimer, D. Mahajan, C. Curino, and G. M. Fumarola, "Towards geo-distributed machine learning," *arXiv preprint arXiv:1603.09035*, 2016.

[14] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds." in *Proc. of NSDI*, 2017.

[15] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proc. of {USENIX}*, 2015.

[16] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[17] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. of IEEE INFOCOM*, 2018.

[18] J. Chen, K. Li, K. Bilal, K. Li, S. Y. Philip *et al.*, "A bi-layered parallel training architecture for large-scale convolutional neural networks," *IEEE transactions on parallel and distributed systems*, vol. 30, no. 5, pp. 965–976, 2018.

[19] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 919–933, 2016.

[20] J. Chen, K. Li, Q. Deng, K. Li, and S. Y. Philip, "Distributed deep learning model for intelligent video surveillance systems with edge computing," *IEEE Transactions on Industrial Informatics*, 2019.

[21] Y. Chen, K. Li, W. Yang, G. Xiao, X. Xie, and T. Li, "Performance-aware model for sparse matrix-matrix multiplication on the sunway taihulight supercomputer," *IEEE transactions on parallel and distributed systems*, vol. 30, no. 4, pp. 923–938, 2018.

[22] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *Proc. of IEEE INFOCOM*, 2013.

[23] C. Xu, K. Wang, P. Li, R. Xia, S. Guo, and M. Guo, "Renewable energy-aware big data analytics in geo-distributed data centers with reinforcement learning," *IEEE Transactions on Network Science and Engineering*, 2018.

[24] C. Xu, K. Wang, and M. Guo, "Intelligent resource management in blockchain-based cloud datacenters," *IEEE Cloud Computing*, vol. 4, no. 6, pp. 50–59, 2017.

[25] K. Wang, Q. Zhou, S. Guo, and J. Luo, "Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3560–3580, 2018.

[26] Y. S. L. Lee, M. Weimer, Y. Yang, and G.-I. Yu, "Dolphin: Runtime optimization for distributed machine learning," in *Proc. of ICML ML Systems Workshop*, 2016.

[27] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proc. of ICML*, 2017.

[28] Y. Gao, L. Chen, and B. Li, "Spotlight: Optimizing device placement for training deep neural networks," in *Proc. of ICML*, 2018.

[29] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. of EuroSys*, 2018.

[30] L. Zheng, C. Joe-Wong, C. G. Brinton, C. W. Tan, S. Ha, and M. Chiang, "On the viability of a cloud virtual service provider," in *Proc. of ACM SIGMETRICS*, 2016.

[31] J. Wang, X. Xiao, J. Wang, K. Lu, X. Deng, and A. A. Gumaste, "When group-buying meets cloud computing," in *Proc. of IEEE INFOCOM*, 2016.

[32] X. Hu, A. Ludwig, A. Richa, and S. Schmid, "Competitive strategies for online cloud resource allocation with discounts: The 2-dimensional parking permit problem," in *Proc. of IEEE ICDCS*, 2015.

[33] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proc. of ACM SOSP*, 2013.

[34] S. Shastri and D. Irwin, "Hotspot: automated server hopping in cloud spot markets," in *Proc. of SOCC*, 2017.

[35] N. Buchbinder, S. Chen, and J. S. Naor, "Competitive Analysis Via Regularization," in *Proc. of ACM SODA*, 2014.

[36] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Online cost minimization for operating geo-distributed cloud cdns," in *Proc. of IEEE/ACM IWQOS*, 2015.

[37] L. Jiao, A. M. Tulino, J. Llorca, Y. Jin, and A. Sala, "Smoothed online resource allocation in multi-tier distributed cloud networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2556–2570, 2017.

[38] L. Jiao, L. Pu, L. Wang, X. Lin, and J. Li, "Multiple granularity online control of cloudlet networks for edge computing," in *Proc. of IEEE SECON*, 2018.

[39] Y. Jia, C. Wu, Z. Li, F. Le, and A. Liu, "Online scaling of nfv service chains across geo-distributed datacenters," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 2, pp. 699–710, 2018.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. of IEEE CVPR*, 2009.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of IEEE CPVR*, 2016.

[42] *Karp's 21 NP-complete problems*, https://tinyurl.com/y64mwkrm.

[43] G. Gens and E. Levner, "Complexity of Approximation Algorithms for Combinatorial Problems:A Survey," *ACM SIGACT News*, vol. 12, no. 3, pp. 52–65, 1980.

[44] P. Sun, Y. Wen, N. B. D. Ta, and S. Yan, "Towards distributed machine learning in shared clusters: A dynamically-partitioned approach," in *Proc. of IEEE SC*, 2017.

[45] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. of ACM SOCC*, 2012.

[46] *TensorFlow*, http://alturl.com/fyp3h.

[47] *Data created on the Internet*, http://alturl.com/jahpw.

[48] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "Firecaffe: Near-linear acceleration of deep neural network training on compute clusters," in *Proc. of IEEE CVPR*, 2016.

[49] *Amazon pricing for data transfer*, https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls.

[50] O. Rogers and D. Cliff, "A financial brokerage model for cloud computing," *Journal of Cloud Computing Advances Systems and Applications*, vol. 1, no. 1, p. 2, 2012.

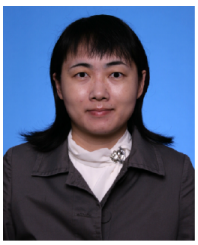
[51] C. Qiu, H. Shen, and L. Chen, "Towards green cloud computing: Demand allocation and pricing policies for cloud service brokerage," in *Proc. of IEEE ICBD*, 2015.

[52] G. Saha and R. Pasumarthy, "Maximizing profit of cloud brokers under quantized billing cycles: a dynamic pricing strategy based on ski-rental problem," *Computer Science*, 2015.

Xiaotong Li received a B.E. degree in Aerospace Information Security and Trusted Computing from Wuhan University, China, in 2017. She is now a master student in School of Computer Science in Wuhan University. Her research interests is in the areas of mobile cloud computing, network optimization, and online scheduling.

Ruiting Zhou has been an Associate Professor in the School of Cyber Science and Engineering at Wuhan University since June 2018. She received a M.S. degree in telecommunications from Hong Kong University of Science and Technology, Hong Kong, in 2008, a M.S. degree in computer science from University of Calgary, Canada, in 2012 and her Ph.D. degree in 2018 from the Department of Computer Science, University of Calgary, Canada. Her research interests include cloud computing, machine learning and mobile network optimization. She has published research papers in top-tier computer science conferences and journals, including IEEE INFOCOM, IEEE/ACM TON, IEEE JSAC, IEEE TMC. She also serves as a reviewer for journals and international conferences such us IEEE JSAC, IEEE TMC, IEEE TCC, IEEE TWC, IEEE Transactions on Smart Grid and IEEE/ACM IWQOS. She held NSERC Canada Graduate Scholarship, Alberta Innovates Technology Futures (AITF) Doctoral Scholarship, and Queen Elizabeth II Graduate Scholarship from 2015 to 2018.

Lei Jiao is an assistant professor at the Department of Computer and Information Science, University of Oregon, USA. He received the Ph.D. degree in computer science from University of Göttingen, Germany in 2014. He worked as a member of technical staff at Alcatel-Lucent/Nokia Bell Labs in Dublin, Ireland from 2014 through 2016, and also as a researcher at IBM Research in Beijing, China in 2010. He is broadly interested in exploring optimization, control, learning, mechanism design, and game theory to manage and orchestrate large-scale distributed computing and communication infrastructures and services. His research has been published in journals such as TON, JSAC, and TMC, and in conferences such as SIGMETRICS, MOBIHOC, INFOCOM, ICNP, SECON, IPDPS, and ICDCS. He is a guest editor for IEEE JSAC Series on Network Softwarization and Enablers and an associate editor of IEEE Access. He has served as a TPC member of a number of conferences such as MOBIHOC, INFOCOM (Distinguished Member), ICDCS, IWQoS, and ICC. He is also a recipient of the Best Paper Awards of IEEE CNS 2019 and IEEE LANMAN 2013, and the 2016 Alcatel-Lucent Bell Labs UK and Ireland Recognition Award.



Chuan Wu received her B.Engr. and M.Engr. degrees in 2000 and 2002 from the 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. Since September 2008, Chuan Wu has been with the Department of Computer Science at the University of Hong Kong, where she is currently an Associate Professor and serves as an Associate Head on curriculum and development matters. Her current research is in the areas of cloud computing, distributed machine learning/big data analytics systems, network function virtualization, and data center networking. She is a senior member of IEEE, a member of ACM, and served as the Chair of the Interest Group on Multimedia services and applications over Emerging Networks (MEN) of the IEEE Multimedia Communication Technical Committee (MMTC) from 2012 to 2014. She is an associate editor of IEEE Transactions on Cloud Computing, IEEE Transactions on Multimedia and ACM Transactions on Modeling and Performance Evaluation of Computing Systems. She has also served as TPC members and reviewers for various international conferences and journals. She was the co-recipient of the best paper awards of HotPOST 2012 and ACM e-Energy 2016.

Yuhang Deng received a B.E. degree in College of Life Sciences from Wuhan University and a second B.E. degree in School of Computer Science & Technology from Huazhong University of Science and Technology, China, in 2018. He is now a master student in School of Computer Science in Wuhan University. His research interests include distributed machine learning and natural language processing.

Zongpeng Li received his B.E. degree in Computer Science from Tsinghua University in 1999, and his Ph.D. degree from University of Toronto in 2005. He has been with the University of Calgary and then Wuhan University. His research interests are in computer networks and cloud computing. Zongpeng was named an Edward S. Rogers Sr. Scholar in 2004, won the Alberta Ingenuity New Faculty Award in 2007, and was nominated for the Alfred P. Sloan Research Fellow in 2007. Zongpeng co-authored papers that received Best Paper Awards at the following conferences: PAM 2008, HotPOST 2012, and ACM e-Energy 2016. Zongpeng received the Department Excellence Award from the Department of Computer Science, University of Calgary, the Outstanding Young Computer Science Researcher Prize from the Canadian Association of Computer Science, and the Research Excellence Award from the Faculty of Science, University of Calgary.