Online Scheduling of Distributed Machine Learning Jobs for Incentivizing Sharing in Multi-Tenant Systems

Ne Wang, Ruiting Zhou, *Member, IEEE*, Ling Han, Hao Chen, *Member, IEEE*, and Zongpeng Li, *Senior Member, IEEE*

Abstract—To save cost, companies usually train machine learning (ML) models on a shared multi-tenant system. In this cooperative environment, one of the fundamental challenges is how to distribute resources fairly among tenants such that each tenant is satisfied. A satisfactory allocation policy needs to meet the following properties. First, the performance of each tenant in the shared cluster is at least the same as that in its exclusive cluster partition. Second, no tenant can get more benefits by lying about its demands. Third, tenants cannot use the idle resources of others for free. Moreover, the resource allocation for ML workloads should avoid costly migration overhead. To this end, we propose a three-layer scheduling framework *Astraea*: i) a batch scheduling framework groups unprocessed ML jobs into multiple batches; ii) a round-by-round algorithm enables tenants to reserve their share of resources and schedule ML jobs in a non-preemptive manner; iii) one-round algorithm based on the primal-dual approach and posted pricing framework, which encourages tenants to report truthful demands and computes a schedule with maximal revenue for each ML job. Besides, our algorithm also allows tenants to trade unused resources on a paid basis. *Astraea* is proven to achieve performance guarantee, polynomial time complexity and some desirable properties of resource sharing, including batch sharing incentive, strategy-proofness, Pareto efficiency and gain-as-you-contribute fairness. Extensive trace-driven simulations show that our algorithm advances in both fairness and cluster efficiency by at least 20% compared to three state-of-the-art baselines.

Index Terms—Distributed machine learning, fairness, multi-tenant system, online scheduling, parameter server architecture

1 Introduction

Recent years have witnessed a booming of machine learning (ML) applications, ranging from language translation [1] and object detection [2] to video classification [3]. The training of ML models is typically costly. For instance, it will spend more than $172 to train a GoogLeNet [4] model over ImageNet-1k [2] on a Titan server with 32 NVIDIA K20 GPUs [5]. Therefore, to save cost, tenants hope to maximize the utilization of the expensive computing resources. However, due to the imbalance in resource demands, the practical resource utilization is not as high as expected. As shown in Fig. 1, we extract usage logs for virtual clusters (i.e., tenants) vcHvQ and vcvG2 from the Helios trace [6] and plot the corresponding GPU utilization. We observe that the GPU utilization of vcHvQ reaches 100% for about 14 hours while vcvG2's GPUs are idle or underutilized at this time. A common approach for alleviating such imbalance is resource sharing, which has been widely adopted in real product and service markets [7], [8], [9]. Cloud providers such as Amazon [10] and Rackspace [11], provide volume discount policies to encourage groups rather than individuals to purchase resources [12], [13]. Incentivized by such mechanisms, ML tenants cooperate to purchase resources in bulk to form a multi-tenant system, and train their ML jobs in a cooperative manner.

Multi-tenant systems face challenges when implementing fair scheduling on ML workload. *First*, a non-preemptive scheduler is desired since migrating or preempting ML jobs is costly. Unlike cloud jobs or big-data jobs, ML jobs need to be *gang-aware* [14], [15], [16], i.e., all resources allocated to an ML job need to be scheduled in an all-or-nothing manner. The reason is that the resources of ML jobs are usually assembled into multiple workers and parameter servers (PSs), which need to cooperate with each other to train a job. As a result, preempting an ML job needs to migrate its model and dataset to new servers, which consumes much scarce bandwidth due to the volume of data. Moreover, it usually takes tens of seconds for a server to suspend and

- *Ne Wang is with the School of Computer Science, Wuhan University, Wuhan 430072, China. E-mail: ne.wang@whu.edu.cn.*
- *Ruiting Zhou is with the School of Computer Science, Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China. E-mail: ruitingzhou@whu.edu.cn.*
- *Ling Han is with the Department of Computer Science, Duke University, Durham, NC 27708 USA, and also with Wuhan University, Wuhan 430072, China. E-mail: hanl@whu.edu.cn.*
- *Hao Chen is with Huawei, Nanjing, Jiangsu 211100, China. E-mail: philips.chenhao@huawei.com.*
- *Zongpeng Li is with the Insititue for Network Sciences and Cyberspace, Tsinghua University, Beijing 100190, China. E-mail: zongpeng@whu.edu.cn.*

Manuscript received 18 Sept. 2021; revised 14 Apr. 2022; accepted 24 Apr. 2022. Date of publication 12 May 2022; date of current version 10 Feb. 2023.
This work was supported in part by NSFC under Grants 62072344 and U20A20177, in part by Hubei Science Foundation under Grant 2020CFB195, in part by the Compact Exponential Algorithm Project of Huawei under Grant YBN2020035131, and in part by Huawei Project under Grant FA2019071041. (Corresponding author: Ruiting Zhou.)
Recommended for acceptance by A. R. Alameldeen.
Digital Object Identifier no. 10.1109/TC.2022.3174566

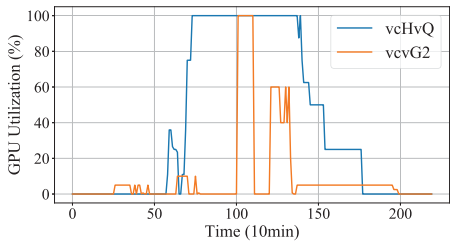


Fig. 1. GPU utilization of two virtual clusters vcHvQ and vcvG2 from April 14,2020 (Time = 0) to April 16, 2020.

wake up an ML job [15], which cannot be ignored. *Second*, without preemption, ML tenants struggle to determine whether and when to share or recall their own resources in online settings, due to the long runtime of ML jobs. Once an ML job is scheduled, it will take up resources for a long time span. Even if a tenant knows current resource usage, it can have difficulty deciding whether to share its spare resources with others or reserve them for future work. *Third*, from an economic perspective, no one is willing to selflessly contribute its spare shares of resources without any compensation. Therefore, to incentivize tenant cooperation, a fair scheduler should balance each tenant's contribution and return.

Unfortunately, popular fair algorithms such as max-min fairness [17] and dominant resource fairness (DRF) [18], widely adopted in cloud computing community [19], [20], [21], [22], big-data jobs [23], [24] and ML workload [14], [16], [25], cannot address all the above challenges. On the one hand, they all achieve fairness by real-time resource reallocation, which guarantees that tenants can preferentially use their own share of resources even in online settings. However, real-time reallocation leads to frequent migration or preemption of ML jobs, and the incurred costly overhead violates the original intention of resource sharing. On the other hand, these mechanisms usually assume that all tenants selflessly contribute their unused resources to improve system efficiency. Consequently, tenants with fewer resources may benefit from long-term free-riding on ones with rich computing capacities, thereby inhibiting tenant participation. More details can be found in Section 2.

To this end, we design an online scheduling algorithm *Astraea* to allocate resources among tenants fairly and schedule their ML jobs in a non-preemptive manner. Given the above challenges, *Astraea* develops a new metric that satisfies the following properties. *First*, the most fundamental property that encourages tenants to participate in the shared system is *sharing incentive*, i.e., the performance of each tenant in the shared cluster is at least the same as that in its exclusive cluster partition [18]. *Second*, resource allocation should be *strategy-proof*, in which tenants cannot make more profits by lying about their resource demands [26]. *Third*, tenants are not allowed to use the idle resources of others for free, thereby preventing tenants who have fewer resources than their demands from free-riding on tenants with rich resources. Moreover, for the collective revenue generated by multiple tenants, the gain distributed to each tenant should be commensurate with its contribution.

However, realizing the above properties is non-trivial under online settings. *First*, to achieve sharing incentive, *Astraea* introduces the idea of batch scheduling to partition the entire time span into multiple intervals and schedule jobs in batches. Consequently, online resource sharing is converted into multiple offline batch scheduling problems, where tenants can flexibly use and trade resources based on the knowledge of their workload and resource usage. *Second*, we allow any unscheduled job to use the unused resources of all tenants, but the job owner is required to pay the corresponding resource cost to the resource owners. The intuition is to transfer inter-tenant resource trading to monetary compensation. The payment acting as the collective revenue is distributed among resource owners using our gain-as-you-contribute division policy, which quantifies the contribution of each resource owner based on a profit-sharing scheme [27]. So far, we have coped with the issue of free-riding and realize gain-as-you-contribute fairness in dividing collective revenue. Finally, the resource cost of a job is determined by the price and amount of resources consumed. We adopt the compact-exponential technique [28] to design price functions and compute a strategy-proof schedule for each job. Our major contributions are highlighted as follows.

- We model sharing incentive as fairness constraints and formulate the problem of maximizing the total revenue of all tenants as a mixed integer nonlinear program (MINLP). To tackle the sharing uncertainty incurred by online settings, we adopt the batch processing techniques in [29], [30] to decompose the original problem into multiple offline batch scheduling problems. The goal of each batch scheduling problem is to maximize the ideal revenue of jobs in a batch. The batch scheduling problem is still a MINLP with non-conventional constraints. To tackle the difficulties, we adopt compact-exponential technique [28] to reformulate it into one with a packing structure.
- We develop *Astraea* to fairly allocate resources to tenants and schedule their ML jobs. Specifically, we design A_{round} to achieve sharing incentive in each interval and performance guarantee through multi-round resource allocation. In each round, A_{round} calls $A_{maxIrev}$ to schedule unprocessed jobs with given resources. For each job, $A_{maxIrev}$ invokes functions MINCOSTD and MINCOSTC to compute a feasible schedule with minimal resource cost, respectively. Among all possible schedules, the one with maximal ideal revenue is picked for the job.
- We prove that *Astraea* achieves a bounded competitive ratio, polynomial time complexity and some desirable properties of resource sharing, including sharing incentive in each interval, strategy-proofness, Pareto efficiency and gain-as-you-contribute fairness.
- We conduct extensive trace-driven simulations to evaluate the performance of *Astraea*. The results illustrate that: (i) *Astraea* improves the collective revenue by up to 58%, 22% and 35% compared to a benchmark for elastic resource allocation [30] and two fair baselines [14] [25], respectively; (ii) the practical competitive ratio (> 0.6) of *Astraea* is much better than the theoretical bound; (iii) each tenant's

revenue computed by *Astraea* is more commensurate with its initial share of resources, which implies *Astraea* is more fair compared to the two fair schedulers.

Roadmap. Section 2 gives an overview of related work. An enhanced sharing incentive mechanism is designed and the corresponding scheduling problem is modeled in Section 3. We then show how to achieve fairness among tenants and schedule distributed ML jobs in the shared system in Section 4. The performance of our proposed algorithm is evaluated in Section 5. Section 6 discusses the generalization of our algorithm to multiple resource types. Finally, Section 7 concludes this paper and directs our future work.

2 Related Work

Fair Schedulers for Cloud Jobs. (Weighted) max-min fairness [17] is one of the most popular fair schedulers in datacenters. A vast amount of work applies it to varies single-type resources, including machines [19] and millimetre wave spectrum [20]. Subsequently, Ghodsi *et al.* [18] propose DRF to fairly allocate multiple types of resources, which is an extension of max-min fairness. A line of work [21], [22], [23], [24] applies or modifies DRF to suit their scenarios. You [21] develops two mechanisms to support the fair scheduling of hierarchical flows while providing share guarantees. Furthermore, the above work achieves some or all of the four commonly used fairness properties, sharing incentive, strategy-proofness, envy-freeness and Pareto efficiency.[1] Additionally, they reallocate resources among tenants in real-time for fairness, since cloud jobs are small in scale and process switching cost is minor. However, these algorithms incur significant preemption overhead when applied to ML multi-tenant systems. Moreover, these schedulers are memoryless and only guarantee fair usage among users at the current time. They do not compensate these users whose resource usage is less than their fair share, lacking the ability to prevent free-riding. F2C [9] realizes similar fairness to our work, but it is a real-time scheduler for cloud jobs.

Schedulers for Distributed Machine Learning. There are recent proposals to achieve efficiency for ML workloads from various perspectives. Peng *et al.* [31] predict the remaining training epochs of deep learning (DL) jobs by building their performance models, in order to flexibly tune the resource allocation of each job on the fly. Xiao *et al.* [32] reduce the fragmentation of resources by packing jobs on as few machines as possible, to improve cluster utilization. Gu *et al.* [15] consider the communication time when training ML model under PS architecture, then design a two-dimensional preemptive scheduler based on priority queue. Zhang *et al.* [30] and Bao *et al.* [33] investigate the demand elasticity to cluster efficiency. Narayanan *et al.* [34] capture the performance heterogeneity of training DL models on different accelerators (such as GPU, TPU and FPGA), and propose a round-based heterogeneity-aware scheduler. These studies focus on maximizing cluster efficiency, whereas our work gives priority to achieve fairness among multiple tenants.

Fair Scheduling of Machine Learning Jobs. Although user fairness have been extensively studied in big-data schedulers [19], [23], [24], [35], they cannot be directly applied to distributed ML due to its distinct characteristics (e.g., placement sensitivity, gang-aware scheduling and long runtime). A few researchers have investigated the fair scheduling of distributed ML jobs. Mahajan *et al.* [16] design finish-time fairness metric for ML applications, while there is no a notion of user. Subsequently, Shubham *et al.* [14] first develop a cluster-wide fair scheduler for multiple users by fairly allocating GPUs to users based on predefined shares and job migration. Then Zhao *et al.* [25] observe that the affinity of GPUs is not considered in the fair sharing based on quota of GPUs, and further improve the level of performance isolation by a virtual private cluster with different levels of affinity. However, none of them investigate the sharing problem without migration or preemption, which has significant impact on cluster efficiency. Besides, they are based on voluntary contribution of resources. However, without effective incentives, tenants may be annoyed by long-term free-riding and depart from the shared system.

Comparison With Related Work. Our algorithm achieves two goals, fair sharing and high cluster efficiency. For the former, our work is significantly different from existing researches, which has been discussed in Section 1. As for the later, [30] is the most similar work to our paper, while several key differences should be addressed. First, the studies in [30] focus on minimizing the total weighted completion time, whereas we aim to maximize the accumulated job revenue. Our objective is a linear or convex function with job completion time. We are the first to extent the performance guarantee from the objective in [29], [30] to the generalized linear or convex forms in our work. Besides, considering the discrete characteristic of resources (usually workers and PSs), we price the resources according to resource usage of the entire cluster. This prevents sudden changes in prices due to smaller resource consumption. Last but not the least, without affecting the revenue of jobs, we give priority to allocate fragmented resources to tenants, which is an efficient policy to improve the cluster efficiency [15], [32].

3 System Model

3.1 System Overview

Multi-Tenant System and ML Jobs. We consider a large distributed ML system shared across U independent tenants (users/groups). These users share their infrastructures to form a coalition, which hosts a large number of workers and PSs. We assume there is only one type of workers and one type of PSs, and discuss the scenario with multiple resource types in Section 6. Let $[X]$ denote the integer set $\{1, 2, \ldots, X\}$. Before joining this coalition, each user $u \in [U]$ bought q_w^u workers and q_p^u PSs in advance for private use. These quotas of resources are recorded as user u' fair share. Besides, the coalition consists of S servers, and each server $s \in [S]$ is equipped with C_{sw} workers and C_{sp} PSs, which reserve b_w and b_p bandwidth, respectively. Naturally, $\sum_u q_w^u = \sum_s C_{sw}$ and $\sum_u q_p^u = \sum_s C_{sp}$. In the shared system,

1. Envy-freeness means that no tenant prefers another tenant's resource allocation. Pareto efficiency means that no tenant can gain more without reducing the income of others.

each user u submits their own J^u ML jobs online for model training. Each job $j^u \in [J^u]$ arrives at time r_{j^u} with large input data sets. The data sets are usually partitioned into D_{j^u} equal-sized data chunks, which is further divided into M_{j^u} equal-sized mini-batches. After traversing the entire data sets E_{j^u} epochs, job j^u is accomplished.

Parameter Server Architecture. The parameter server (PS) architecture along with data parallelism is widely employed by most distributed ML frameworks [30], [33], [36]. With such combination, one iteration of training a job consists of four stages. First, each worker trains its data sets through local model and derives gradients. Second, the gradients are pushed to all PSs by all workers. Third, PSs update the global parameters via the received gradients. Finally, workers pull the newest parameters from PSs. In the third stage, if synchronous stochastic descent gradient method [37] is adopted, PSs are not allowed to update parameters until they receive all the gradients. Because such synchronization contributes to the convergence of models, this work adopts synchronous method to train models.

Processing Capacity. The processing capacity of job j^u, i.e., the number of mini-batches that can be trained in one time slot, is denoted by n_{j^u} and computed as below. We first calculate the time to train a mini-batch. The time of the first stage is dependent with the types of workers and the size of mini-batch, which can be indicated as m_{j^u}. The gradients and parameters transmitted between workers and PSs are the same in size, denoted by ϵ_{j^u}. Then the communication time is calculated by $\frac{2\epsilon_{j^u}}{b_w}$. Subsequently, the time to update parameters is denoted by G_{j^u} in analogy to the notion of m_{j^u}. So far the computation is completed. Specially, if all workers and PSs are placed in the same server, i.e., centralized placement, then the communication time can be eliminated. We introduce an indicator variable $\zeta_{j^u} = 1$ to reflect this case. Correspondingly, $\zeta_{j^u} = 0$ implies that a job employs at least two servers to deploy its workers and PSs, called *distributed placement*. Thus, we have:

$$n_{j^u} = \begin{cases} 1/(m_{j^u} + G_{j^u} + \frac{2\epsilon_{j^u}}{b_w}), & \text{if} \quad \zeta_{j^u} = 0 \\ 1/(m_{j^u} + G_{j^u}), & \text{if} \quad \zeta_{j^u} = 1 \end{cases} \tag{1}$$

Enhanced Sharing Incentive. To enhance sharing incentive, we state: (i) a user can utilize its quota. (ii) if a user cannot make full use of its fair share, it can share its unused resources to help other users train models for a fee. (iii) in turn, if a user employs others' unused resources for training, it is required to pay for the use of resources beyond its fair share. Undoubtedly, in the first case, the job revenue belongs to its owner. For the last two cases, suppose user u' employs the unused resources of others to train job $j^{u'}$, which generates two-fold resource cost: worker cost denoted by $c_{j^{u'}w}$ and PS cost indicated as $c_{j^{u'}p}$. The resource cost is paid by the job owner as the collective income of resource owners. Given the collective payoff, the next question is how to fairly distribute the payoff among resource owners so that each owner's gain is commensurate with its contribution, referred to as gain-as-you-contribute fairness. A common approach, called Egalitarian, is one of profit-sharing schemes [27]. In Egalitarian, any unit of utility produced by a coalition is divided equally among the resource owners who help produce it. Based on this idea, we regard the ratio of user u' unused resources to the total unused resources as its contribution. As a result, u's income derived from job u' in time slot t, denoted as $\rho^u_{j^{u'}}(t)$, is computed as:

$$\rho^u_{j^{u'}}(t) = \frac{q^u_w - \sum_{j^u}\sum_s y_{j^u s}(t)}{a_w(t)} c_{j^{u'}w} + \frac{q^u_p - \sum_{j^u}\sum_s z_{j^u s}(t)}{a_p(t)} c_{j^{u'}p} \tag{2}$$

where $a_w(t) = \sum_u (q^u_w - \sum_{j^u}\sum_s y_{j^u s}(t))$ $(a_p(t) = \sum_u (q^u_p - \sum_{j^u}\sum_s z_{j^u s}(t)))$ is the number of total unused workers (PSs) of all users at time t. Such division is proportionally fair, i.e., a resource owner's share of income is proportional to its resource contribution. Moreover, one user can pay itself when it is one of resource owners.

Decision Variable. Upon receiving user u' job j^u, the cluster manager computes its resource allocation and schedule: (i) $y_{j^u s}(t)$, the number of workers in server s at time t allocated to job j^u; (ii) $z_{j^u s}(t)$, the number of PSs in server s at time t allocated to job j^u; (iii) $c_{j^u w}, c_{j^u p} \geq 0$, the expense if job j^u is scheduled. Important notations are listed in Table 1.

TABLE 1
Notations

Inputs	Descriptions
$[X]$	integer set $\{1, 2, \ldots, X\}$
u, j^u, s, t	indexes for user, job, server and time slot
U, J^u, S, T	# of users, jobs of u, servers and time slots
r_{j^u}	arrival time of j^u
$E_{j^u}, D_{j^u}, M_{j^u}$	# of training epochs, data chunks, mini-batches in one data chunk of job j^u
q^u_w, q^u_p	user u' fair share of workers and PSs
C_{sw}, C_{sp}	# of workers and PSs on server s
$f_{j^u}(\cdot)$	$j^{u\prime}$ utility function of completion time
Decisions	**Descriptions**
$y_{j^u s}(t), z_{j^u s}(t)$	# of workers and PSs in s allocated to j^u at t
$t^-_{j^u}, t^+_{j^u}$	start and completion time of j^u
$\rho^u_{j^{u'}}(t)$	income of u by helping train job $j^{u'}$ at t
n_{j^u}	# of mini-batches trained by one worker and one PS of job j^u at a slot
ζ_{j^u}	if j^u's workers and PSs are placed on one server
$c_{j^u w}, c_{j^u p}$	worker and PS cost that j^u needs to pay
$a_w(t), a_p(t)$	# of total unused workers (PSs) of all users at t
o_{j^u}	if j^u is scheduled
$o^\iota_{j^u}$	if j^u is scheduled with schedule ι
$y^\iota_{j^u s}(t), z^\iota_{j^u s}(t)$	$y_{j^u s}(t)$ and $z_{j^u s}(t)$ with specified schedule ι

3.2 Problem Formulation

Revenue Maximization. Let $t^+_{j^u}$ be the completion time of job j^u of user u. Then the revenue of user u includes the

following three parts: (i) The utility is resulted by training user u' own jobs within its fair share. In this case, users do not have to pay for resource cost. We denote these jobs as set $[J_1^u]$, then the utility $f_{j_1^u}(t^+_{j_1^u})$ of job $j_1^u \in [J_1^u]$ is fully accumulated into user u' revenue. (ii) By sharing the unused resources to help other users train their models, user u can obtain revenue $\sum_{t=1}^{T}\sum_{u' \in [U]\setminus\{u\}}\sum_{j^{u'}} \rho^u_{j^{u'}}(t)$. (iii) User u employs the unused resources of other users, in consequence, its revenue is the utility of these jobs minus the resulting resource cost. That is $\sum_{j_2^u \in [J^u]\setminus[J_1^u]}(f_{j_2^u}(t^+_{j_2^u}) - c_{j_2^u w} - c_{j_2^u p})$. For a shared system, if fairness among users has been ensured, all users are required to cooperate with other to maximize the collective efficiency. In this paper, we have formulated a common objective for all users, i.e., maximize the total revenue of them. Observing (ii) and (iii), we can know one user's resource cost in (iii) is other users' income in (ii). By summing up the revenue in (ii) and (iii) of all users, the value obtained is exactly the total utility of jobs trained by cooperating, i.e., $\sum_{j_2^u \in [J^u]\setminus[J_1^u]} f_{j_2^u}(t^+_{j_2^u})$. Furthermore, the collective income is $\sum_u \sum_{j^u} f_{j^u}(t^+_{j^u})$.

Problem Formulation. We formulate the problem of maximizing total revenue as below.

$$\text{maximize} \quad \sum_{u\in[U]}\sum_{j^u\in[J^u]} f_{j^u}(t^+_{j^u}) \tag{3}$$

subject to:

$$\sum_s\sum_{j^u} y_{j^u s}(t) \le q_w^u + \sum_{u'\neq u}(q_w^{u'} - \sum_{j^{u'}}\sum_s y_{j^{u'}s}(t)), \forall u, \forall t \tag{3a}$$

$$\sum_s\sum_{j^u} z_{j^u s}(t) \le q_p^u + \sum_{u'\neq u}(q_p^{u'} - \sum_{j^{u'}}\sum_s z_{j^{u'}s}(t)), \forall u, \forall t \tag{3b}$$

$$\sum_u\sum_j y_{j^u s}(t) \le C_{sw}, \forall s, \forall t \tag{3c}$$

$$\sum_u\sum_j z_{j^u s}(t) \le C_{sp}, \forall s, \forall t \tag{3d}$$

$$\sum_s y_{j^u s}(t) \le D_{j^u}, \forall u, \forall j^u, \forall t \tag{3e}$$

$$y_{j^u s}(t) = y_{j^u s}(t+1), \forall u, \forall j^u, \forall s, \forall t \in [t^-_{j^u}, t^+_{j^u}] \tag{3f}$$

$$\sum_s z_{j^u s}(t) \ge 1, \forall u, \forall j^u, \forall t : \sum_s y_{j^u s}(t) > 0 \tag{3g}$$

$$z_{j^u s}(t) = z_{j^u s}(t+1), \forall u, \forall j^u, \forall s, \forall t \in [t^-_{j^u}, t^+_{j^u}] \tag{3h}$$

$$\sum_{s'\in[S]\setminus\{s\}} y_{j^u s'}(t) b_w \le z_{j^u s}(t) b_p, \forall u, \forall j^u, \forall s, \forall t \tag{3i}$$

$$\zeta_{j^u} = 1, \forall u, \forall j^u : s = s,' \forall y_{j^u s}(t) > 0, \forall z_{j^u s'}(t) > 0 \tag{3j}$$

$$\sum_s y_{j^u s}(t) n_{j^u} \ge E_{j^u} D_{j^u} M_{j^u}, \forall u, \forall j^u \tag{3k}$$

$$y_{j^u s}(t) = z_{j^u s}(t) = 0, \forall u, \forall j^u, \forall s, \forall t < r_{j^u} \vee t \notin [t^-_{j^u}, t^+_{j^u}] \tag{3l}$$

$$t^+_{j^u} = \arg\max_{t\in[T]} \Big\{ \sum_s y_{j^u s}(t) > 0 \Big\}, \forall u, \forall j^u \tag{3m}$$

$$t^-_{j^u} = \arg\min_{t\in[T]} \Big\{ \sum_s y_{j^u s}(t) > 0 \Big\}, \forall u, \forall j^u \tag{3n}$$

$$y_{j^u s}(t), z_{j^u s}(t) \in \{0,1,2,\ldots\}, \zeta_{j^u} \in \{0,1\}, t^-_{j^u}, t^+_{j^u} \in [T], \forall u, \forall j^u, \forall s, \forall t. \tag{3o}$$

Constraints (3 a) and (3 b) represent that a job can employ both its fair share and others' unused resources at any time slot. Constraints (3 c) and (3 d) are the resource capacity constraints of servers. Constraints (3 e)–(3 n) are needed to train one ML job. Constraint (3 e) limits the number of allocated workers to be at most the number of data chunks, D_{j^u}. Constraint (3 g) assures that there is at least one PS allocated to each job. Constraints (3 f) and (3 h) indicate that the workers and PSs allocated to a job cannot be preempted until its training is completed. Constraint (3 i) ensures that each PS should cover the bandwidth of all workers deployed on other servers. Constraint (3 j) reveals that only if all workers and PSs of job j^u are deployed together can it ignore the communication time between workers and PSs. Constraint (3 k) guarantees that job j^u is accomplished by sufficient workers. $E_{j^u} D_{j^u} B_{j^u}$ is the total number of mini-batches that need to be trained. Constraint (3 l) implies that only when a job arrives or it is running, can the job consume workers and PSs. Constraints (3 m) and (3 n) compute the job completion time and start time to train, respectively.

Challenges. The maximization problem in (3) is a mixed integer nonlinear program (MINLP), even if its offline version is NP-hard [38]. Moreover, $y_{j^u s}(t)$, $z_{j^u s}(t)$ and n_{j^u} are interdependent, which makes the job completion time $t^+_{j^u}$ and job utility $f_{j^u}(t^+_{j^u})$ unpredictable. Besides, due to the online arrival and long runtime of ML jobs, it is rather tough to ensure fairness among users. One solution is to conduct a realtime scheduling at each time slot, which is not practical due to expensive overhead of preemption or migration. However, non-realtime scheduling cannot capture the intensity of users' willingness to share and the unpredictability of future. In online setting, users have no knowledge of their future jobs, so users cannot determine whether and when to share their unused resources. If a user shares its unused resources to train other users' jobs, these jobs will occupy the shared resources for a long time. As a result, if new jobs of this user comes during renting, there may be not enough resources to serve its new jobs.

4 Online Scheduling Framework for Incentivizing Sharing

4.1 Algorithm Idea

To tackle the uncertainty in sharing incurred by online arrival of jobs, we introduce the idea of batch scheduling to group jobs into multiple batches. As a consequence, by scheduling jobs batch by batch, resource sharing among users is bounded within a batch, which is independent with future knowledge. In a batch, the jobs of each user are first scheduled within its fair share to achieve sharing incentive, we call it *batch sharing incentive*. After traversing all the users and their pending jobs, the total unused resources in this batch are shared by the remaining unprocessed jobs. Besides, each job can be scheduled only once, i.e., the migration and preemption of jobs are evaded. We design an online scheduling framework *Astraea* along with an offline approximation algorithm A_{round} to arrange jobs in one batch. The main idea of our algorithm is shown in Fig. 2.

- In Section 4.2, we first partition the time horizon into multiple intervals at geometrically increasing points. Consequently, the online problem is converted into multiple offline batch scheduling problems, which

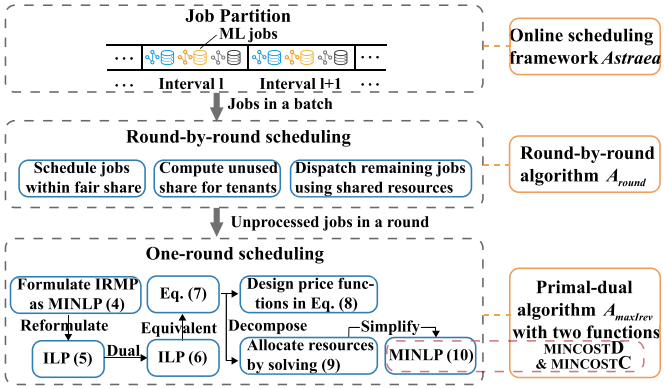


Fig. 2. Main idea of our online algorithm.

are solved in different time interval. In each batch, to gain provable performance guarantee, we run the total *ideal* revenue maximization problem in Definition 1 by multiple rounds, i.e., maximize the ideal revenue of given jobs under given resources.

- In Section 4.3 and Section 4.4, we design algorithms A_{round} and $A_{maxIrev}$ to solve the constructed offline batch scheduling problem. We argue that the produced schedules still satisfy two properties mentioned in Section 4.3. For the batch scheduling problem, two sub-problems need to be solved: (i) whether to process a job in this batch; (ii) if a job is admitted, the amount and occupy time of resources should be determined. To solve the two sub-problems, we adopt the compact-exponential technique [28] to reformulate the offline batch scheduling problem into one with a packing structure. Following this technique, we formulate the dual problem of the reformulated one. Then by well interpreting the dual problem and combining complementary slackness, we can get a solution to the first sub-problem, i.e., find an optimal schedule that maximizes the job' revenue. To this end, for each job, we design two subroutines MINCOSTD and MINCOSTC to compute a feasible schedule with minimal resource cost under distributed and centralized placement, respectively. Among all the possible schedule, the one with maximal revenue is picked. If the maximal revenue is greater than 0, then the job is admitted and is processed with the picked schedule. So far, the second sub-problem is also solved.

4.2 Online Scheduling Framework

As shown in Fig. 3, the time horizon is partitioned at geometrically increasing points, $\tau_0 = 1, \tau_1 = 2, \ldots, \tau_l = 2^{l-1}, \ldots, \tau_L$, where L is the interval that all jobs are completed by time τ_L. To eliminate the uncertainty of online settings, we collect unprocessed and newly arrived jobs in previous intervals, and schedule them in the next interval (weighted by λ, which is discussed in Section 4.3). In this way, our online scheduling is converted into multiple offline batch scheduling problems. To this end, we develop an online scheduling framework *Astraea* in Alg. 1. For each interval l, we wait until time τ_l and collect unprocessed jobs into set J_l (line 4). Subsequently, we invoke A_{round} to schedule jobs in

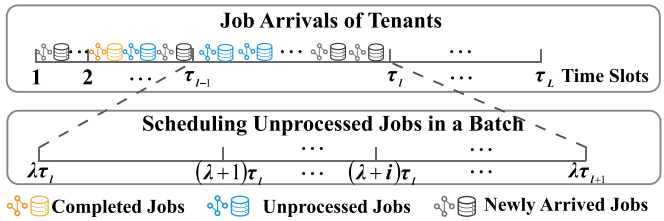


Fig. 3. The workflow of online scheduling framework.

J_l (line 5). Then the admitted jobs ($\widetilde{J}_l$) are run within timespan $(\lambda\tau_l, \lambda\tau_{l+1}]$ (line 6), where λ is a parameter that play an important role in achieving performance guarantee and is computed in Section 4.3. Eventually, unprocessed jobs in this interval ($J_l \setminus \widetilde{J}_l$) are postponed to the next interval for scheduling (line 7).

Algorithm 1. Online Scheduling Framework for Incentivizing Sharing: *Astraea*

Input: $T, C_{sw}, C_{sp}, q^u_w, q^u_p, \forall s \in [S]$

Output: $y_{j^u s}(t), z_{j^u s}(t), c_{j^u w}, c_{j^u p}, \rho^u_{j^{u'}}(t), \quad \forall u, u' \in [U], \forall j^u \in [J^u], \forall s \in [S], \forall t \in [T]$

1: Initialize $y_{j^u s}(t) = 0, z_{j^u s}(t) = 0, c_{j^u w} = c_{j^u p} = 0, \rho^u_{j^{u'}}(t) = 0, \forall u, u,' \forall j^u, \forall s, \forall t$
2: **for** $l = 1, 2, \ldots, L$ **do**
3: Initiate $J_l = \emptyset, \widetilde{J}_l = \emptyset$
4: Set $J_l = J_{l-1} \cup \{j^u | r_{j^u} \geq \tau_{l-1} \wedge r_{j^u} < \tau_l\}$
5: Set $\{\{t^*_{j^u}, \mu_{j^u}, c_{j^u w}, c_{j^u p}, \rho^u_{j^{u'}}(t)\}_{\forall j^u}, \widetilde{J}_l\} = A_{round}(J_l, l)$
6: Schedule each job in $\widetilde{J}$ in $(\lambda\tau_l, \lambda\tau_{l+1}]$ according to $\{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$
7: Set $J_l = J_l \setminus \widetilde{J}_l$
8: **end for**

Example. Here, we use a small example to describe how our algorithms work. We assume that there are two users and four servers in our system, i.e., $U = 2, S = 4$. For ease of computation and description, we assume servers are homogeneous, each with two workers and one PS, $C_{sw} = 2, C_{sp} = 1, \forall s \in \{1, 2, 3, 4\}$. The users have the same share fair of resources, i.e., $q^1_w = q^2_w = 4, q^1_p = q^2_p = 2$. User u_1 owns two jobs indexed by j^{11}, j^{12}, and user u_2 has two jobs, j^{21}, j^{22}. For each job $j^u \in \{j^{11}, j^{12}, j^{21}, j^{22}\}$, its configurations are as follows:$n_{j^u} = 1$ if $\zeta_{j^u} = 1$, otherwise $n_{j^u} = 0.5$; $r_{j^u} = 3, E_{j^u} = 2, M_{j^u} = 1, f_{j^u}(t^+_{j^u}) = -2 * (t^+_{j^u} - r_{j^u}) + 100, b_w = b_p = 1$. In particular, $D_{j^{11}} = D_{j^{21}} = 2, D_{j^{12}} = 3, D_{j^{22}} = 30$. According to line 4 in Alg. 1, the arrival time of each job is within $[2^1, 2^2)$, so $J_2 = \{j^{11}, j^{12}, j^{21}, j^{22}\}$ by assuming that no job arrives before $t = 3$. Let $\lambda = 2$. Jobs in J_2 are scheduled within (8,16] by incorporating Alg. 2-Alg. 5, designed in Section 4.3 and 4.4. Here we directly give the final scheduling results, and explain how these values are calculated by these algorithms later. All the resources of server s_1 are allocated to job j^{11} during (8,10], server s_2 is assigned to job j^{21} during (10,12]. The above allocation is within the fair share of each user. Besides, the two users cooperate to schedule job j^{12}, i.e., two workers and one PS of server s_3 and one worker of s_4 are allocated to j^{12} during (8,12]. As a result, user u_2 gains 6 from user u_1 by sharing its unused resources. Job j^{22} is too big to be scheduled in this interval. We save it by set J_2 and try to schedule it in the third interval.

4.3 Round-By-Round Algorithm for One Batch

Next, we describe a technique that produces an algorithm A_{round} with a bounded approximation ratio for NP-hard batch scheduling problem. This technique is quite dependent on the existence of a bounded algorithm $A_{maxIrev}$ for the problem defined in Definition 1. We focus on the design of A_{round} here, and discuss $A_{maxIrev}$ in Section 4.4.

The sketch of A_{round} is presented in Alg. 2. We argue that A_{round} satisfies the following two properties in Lemma 1: (i) the length of any produced schedule is at most $D = \tau_l$; (ii) the total revenue generated by A_{round} is at least the optimal revenue by optimally solving the problem in Definition 1 with deadline $D = \tau_l$. In Alg. 2, the input jobs in J_l are scheduled for λ rounds in $(\lambda\tau_l, \lambda\tau_{l+1}]$ (line 2), where λ is computed in Lemma 1. In each round i, we first schedule each user's jobs within its fair share, i.e., invoke $A_{maxIrev}$ with $a_w(t) = 0, a_p(t) = 0$ (line 3). Then the values of $a_w(t), a_p(t)$ are updated according to set J_l^s of scheduled jobs (line 4). In line 5, the remaining jobs ($J_l \setminus J_l^s$) of all users are processed using all the unused resources, i.e., call $A_{maxIrev}$ with updated $a_w(t), a_p(t)$. Finally, for each accepted job j^u in $J_l^{s'}$, the corresponding expense that user u should pay to others is computed according to Eq. (2).

Definition 1. (Ideal Revenue Maximization Problem.) *In a ML cluster, given a deadline D, a set of jobs available at time 0, and a utility function with completion time for each job, construct a feasible schedule that maximizes the total utility of jobs completed by time D.*

Lemma 1. *The schedules computed by algorithm A_{round} satisfy the two properties highlighted in this section. Besides, $\lambda = \lfloor \frac{\log f(\bigcup_{u=1}^{U}[J^u]) - \log f_{min}}{\log\sigma - \log(\sigma-1)} \rfloor + 1$, where σ is the approximation ratio of $A_{maxIrev}$.*

Proof. Please see Appendix A., which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2022.3174566 □

Example. We follow the system parameters in the example of Section 4.2. Moreover, we use the same parameters in subsequent sections unless otherwise stated. At the beginning of each round, such as $i = 1$, there are no unused workers and PSs, i.e., $a_w(t) = 0, a_p(t) = 0, \forall t \in (8, 12]$. In line 3 of Alg. 2, we resort to $A_{maxIrev}$ to allocate two workers and one PS in server s_1 to centrally schedule job j^{11} during (8,10] and allocate all the resources in server s_2 to schedule job j^{21} during (10,12]. The two jobs are scheduled within their corresponding job owners' fair share. For each user, its remaining fair share of resources is not adequate to complete j^{12} or j^{22}. Next, all the unused resources (i.e., servers s_3 and s_4) are shared among the two users. As a result, $a_w(t) = 6, a_p(t) = 3, \forall t \in (8, 12]$ by assuming the time to make scheduling decisions is negligible. Subsequently, during time span (8,12], two workers and one PS of server s_3 and one worker of server s_4 are allocated to job j^{12} by line 5 in Algorithm 2. For $t \in (8, 12]$, the worker cost is $3 * 1 = 3$, and the PS cost is $1 * 1 = 1$. According to line 6, user u_2 gains $(4-0)/6 * 3 * 2 + (2-0)/3 * 1 * 2 + (4-2)/6 * 3 * 2 \quad +(2-1)/3 * 1 * 2 = 8$ by sharing its unused resources. Then we attempt to schedule the unprocessed job j^{22} in next round 2 ($t \in (12, 16]$), but it is too big to schedule. As a result, j^{22} is postponed to the next interval for scheduling.

Algorithm 2. Round-by-Round Approximation Algorithm of Job Scheduling in One Batch: A_{round}

Input: J_l, l
Output: $\widetilde{J}_l, \{\iota^*_{j^u}, \mu_{j^u}, c_{j^u w}, c_{j^u p}, \rho^u_{j^{u'}}(t)\}_{\forall j^u \in \widetilde{J}_l}$

1: Initiate $y_{j^u s}(t) = z_{j^u s}(t) = 0, a_w(t) = a_p(t) = 0, \forall u, \ \forall j^u, \forall s, \forall t \in (\lambda\tau_l, \lambda\tau_{l+1}], \widetilde{J}_l = \emptyset$
2: **for** $i = 1$ to λ **do**
3: Set $\{\{\iota^*_{j^u}, \mu_{j^u}\}_{\forall j^u}, J_l^s\} = A_{maxIrev}(l, i, a_w(t), a_p(t), J_l)$
4: Set $a_w(t) = \sum_u (q^u_w - \sum_{j^u \in J_l^s} \sum_s y_{j^u s}(t)), a_p(t) = \sum_u (q^u_p - \sum_{j^u \in J_l^s} \sum_s z_{j^u s}(t)), \forall t \in (\lambda + i - 1)\tau_l, (\lambda + i)\tau_l]$
5: Set $\{\{\iota^*_{j^u}, \mu_{j^u}\}_{\forall j^u}, J_l^{s'}\} = A_{maxIrev}(l, i, a_w(t), a_p(t), J_l \setminus J_l^s)$
6: Compute $c_{j^u w}, c_{j^u p}, \rho^u_{j^{u'}}(t)$ for each job in $J_l^{s'}$ according to Eq. (2)
7: Set $\widetilde{J}_l = \widetilde{J}_l \cup \{J_l^s \cup J_l^{s'}\}, J_l = J_l \setminus \widetilde{J}_l$
8: **end for**
9: **Return** $\widetilde{J}_l, \{\iota^*_{j^u}, \mu_{j^u}, c_{j^u w}, c_{j^u p}, \rho^u_{j^{u'}}(t)\}_{\forall j^u \in \widetilde{J}_l}$

4.4 Primal-Dual Algorithm for One-Round Scheduling

Problem Reformulation. In each round, we solve the total ideal revenue maximization problem (IRMP) in Definition 1 with different jobs as input, as shown in (4). Since not all jobs in J_l can be completed at current interval l, we introduce variable o_{j^u} to denote whether it is scheduled ($o_{j^u} = 1$) or not ($o_{j^u} = 0$).

$$\text{maximize} \quad \sum_{u \in [U]} \sum_{j^u \in J_l} o_{j^u} f_{j^u}(t^+_{j^u}) \tag{4}$$

subject to:

$$o_{j^u} \in \{0, 1\}, \forall u, \forall j^u \in J_l$$
$$(3a) - (3o), \text{ where } j^u \in J_l, \forall t \in (0, \tau_l] \tag{4a}$$

This problem is non-trivial due to non-conventional constraints (3 m) and (3 n). Even without these two constraints, the problem is still an offline integer linear program (ILP), which is also NP-hard. We next design an approximation algorithm $A_{maxIrev}$ to solve this problem in polynomial time. We adopt the compact-exponential technique [28] to reformulate IRMP into the following ILP (5). Here $\mathcal{L}_{j^u}$ denotes the set of feasible schedules for job j^u. A feasible schedule is composed of $y_{j^u s}(t), z_{j^u s}(t)$ that satisfy all the constraints except capacity constraints (3 c) and (3 d). Furthermore, by abusing the aforementioned notions, we use $o^\iota_{j^u}$ to indicate whether to admit job j^u with schedule $\iota \in \mathcal{L}_{j^u}$. Likely, $y^\iota_{j^u s}(t)$ and $z^\iota_{j^u s}(t)$ represent $y_{j^u s}(t)$ and $z_{j^u s}(t)$ with specified schedule ι, respectively. Similarly, $t^{+\iota}_{j^u}$ is the completion time with schedule ι.

$$\text{maximize} \quad \sum_{u \in [U]} \sum_{j^u \in [J^u]} \sum_{\iota \in \mathcal{L}_{j^u}} o^\iota_{j^u} f_{j^u}(t^{+\iota}_{j^u}) \tag{5}$$

subject to:

$$\sum_u \sum_{j^u} \sum_{\iota} y^{\iota}_{j^u s}(t) o^{\iota}_{j^u} \le C_{sw}, \forall s, \forall t \in (0, \tau_l] \tag{5a}$$

$$\sum_u \sum_{j^u} \sum_{\iota} z^{\iota}_{j^u s}(t) o^{\iota}_{j^u} \le C_{sp}, \forall s, \forall t \in (0, \tau_l] \tag{5b}$$

$$\sum_{\iota \in \mathcal{L}_{j^u}} o^{\iota}_{j^u} \le 1, \forall u, \forall j^u \tag{5c}$$

$$o^{\iota}_{j^u} \in \{0,1\}, \forall u, \forall j^u, \forall \iota \in \mathcal{L}_{j^u}. \tag{5d}$$

Problem (5) is equivalent to (4). Consequently, if we can compute a feasible solution to problem (5), problem (4) is solved as well.

Dual Problem. Note that there are potential exponential number of $o^{\iota}_{j^u}$ corresponding to feasible schedules in (4). We formulate the dual of (4) by relaxing integer variables $o^{\iota}_{j^u} \in \{0,1\}$ to $o^{\iota}_{j^u} \ge 0$ and introducing dual variables $\eta_{sw}(t)$, $\eta_{sp}(t)$ and μ_{j^u} to constraints (5 a), (5 b) and (5 c), respectively.

$$\min \quad \sum_u \sum_{j \in J_l} \mu_{j^u} + \sum_s \sum_t \eta_{sw}(t) C_{sw} + \sum_s \sum_t \eta_{sp}(t) C_{sp} \tag{6}$$

subject to:

$$\mu_{j^u} \ge f_{j^u}(t^{+\iota}_{j^u}) - \sum_s \sum_t \eta_{sw}(t) y^{\iota}_{j^u s}(t) - \sum_s \sum_t \eta_{sp}(t) z^{\iota}_{j^u s}(t), \quad \forall u, \forall j^u \in J_l, \forall \iota \in \mathcal{L}_{j^u} \tag{6a}$$

$$\mu_{j^u}, \eta_{sw}(t), \eta_{sp}(t) \ge 0, \forall u, \forall j^u \in J_l, \forall s, \forall t \in (0, \tau_l] \tag{6b}$$

The dual variables $\eta_{sw}(t)$ and $\eta_{sp}(t)$ can be interpreted as the unit prices of workers and PSs on server s at time t, respectively. Then $\sum_s \sum_t \eta_{sw}(t) y^{\iota}_{j^u s}(t) + \sum_s \sum_t \eta_{sp}(t) z^{\iota}_{j^u s}(t)$ is the total resource cost of job j^u with schedule ι. As a result, the right hand side (RHS) of (6), i.e., ideal job utility minus total resource cost, can be interpreted as ideal income of each user. According to complementary slackness, if job j^u is admitted, i.e., $o^{\iota}_{j^u} = 1$, the corresponding constraint (6 a) is tight. Combining $\mu_{j^u} \ge 0$ in constraint (6 b), we have:

$$\mu_{j^u} = \max\{0, \max_{\iota \in \mathcal{L}_{j^u}} \text{RHS of (6a)}\}. \tag{7}$$

Therefore, for each job j^u, we first compute its optimal schedule ι^* and corresponding μ_{j^u}, $\iota^* = \arg\max_{\iota \in \mathcal{L}_{j^u}} \text{RHS of (6a)}$. If $\mu_{j^u} > 0$, then the cluster manager schedules job j^u, otherwise rejects it.

Price Function. To compute resource cost, we need to determine resource prices $(\eta_{sw}(t), \eta_{sp}(t))$ and resource consumption $(y^{\iota}_{j^u s}(t), z^{\iota}_{j^u s}(t))$. At one time slot, setting the same price for workers with identical type is in line with practical cluster market. To this end, we introduce variables $\eta_w(t), \eta_p(t)$ to denote the prices of workers and PSs at time t, where $\eta_w(t) = \eta_{sw}(t), \eta_p(t) = \eta_{sp}(t), \forall s, \forall t$. In addition, since exponential functions can capture market rules, we design exponential price functions of workers and PSs. Let $h_{sw}(t), h_{sp}(t)$ be the number of allocated workers and PSs in server s at time t, respectively. Then the total number of allocated workers and PSs are denoted by $h_w(t), h_p(t)$, and $h_w(t) = \sum_s h_{sw}(t), h_p(t) = \sum_s h_{sp}(t)$. The formal expressions of price functions are given in Eq. (8).

$$\eta_w(h_w(t)) = (\theta_w)^{\frac{h_w(t)}{\sum_s C_{sw}}} - 1, \eta_p(h_p(t)) = (\theta_p)^{\frac{h_{sp}(t)}{\sum_s C_{sp}}} - 1 \tag{8}$$

When resources are exhausted, the price of workers (PSs) is $\theta_w - 1$ ($\theta_p - 1$). Thus, $\theta_w - 1$ ($\theta_p - 1$) should be large enough to prevent jobs from being scheduled, i.e., $f_{j^u}(t^{+\iota}_{j^u}) - \sum_s \sum_t (\theta_w - 1) y^{\iota}_{j^u s}(t) - \sum_s \sum_t (\theta_p - 1) z^{\iota}_{j^u s}(t) < 0, \forall u, \forall j^u$. A feasible solution is to set $\frac{f_{j^u}(t^+_{j^u})}{\sum_s \sum_t y^{\iota}_{j^u s}(t)} \le \theta_w - 1, \frac{f_{j^u}(t^+_{j^u})}{\sum_s \sum_t z^{\iota}_{j^u s}(t)} \le \theta_p - 1, \forall u, j^u$ by assuming $\theta_w - 1 \ge 1, \theta_p - 1 \ge 1$, i.e., $\frac{f_{j^u}(t^+_{j^u})}{\sum_s \sum_t y^{\iota}_{j^u s}(t)} \ge 1, \frac{f_{j^u}(t^+_{j^u})}{\sum_s \sum_t z^{\iota}_{j^u s}(t)} \ge 1, \forall u, j^u$.

Resource Allocation. We next investigate how to produce an optimal schedule ι^* for job j^u and then determine whether to accept it in current round. This optimization problem is formulated as follows:

$$\max_{t^-_{j^u}, t^+_{j^u}, \zeta^u_j, \mathbf{y}, \mathbf{z}} f_{j^u}(t^+_{j^u}) - \sum_s \sum_t \eta_w(t) y_{j^u s}(t) - \sum_s \sum_t \eta_p(t) z_{j^u s}(t) \tag{9}$$

subject to:

$$y_{j^u s}(t) \le C_{sw} - h_{sw}(t), \forall s, \forall t \in (0, \tau_l] \tag{9a}$$

$$z_{j^u s}(t) \le C_{sp} - h_{sp}(t), \forall s, \forall t \in (0, \tau_l]$$

$$\text{Constraints } (3a) - (3b)(3e) - (3o), \text{ where } o_{j^u} = 1, \forall t \in (0, \tau_l] \tag{9b}$$

In problem (9), the number of workers and PSs $(y_{j^u s}(t), z_{j^u s}(t))$ is hard to determine. Because both of revenue and resource cost are non-decreasing with the quantities of resources in general. To this end, we continue to fix start time $t^-_{j^u}$, and enumerate the possible number of workers from 1 to D_{j^u}. As a result, the completion time $t^+_{j^u}$ is computed. With fixed completion time, the objective in problem (9) is equivalent to minimizing resource cost, as shown in ILP (10). Consequently, the remaining issues include determining the number of PSs and the location of resources, which are closely related to placement policies. Finally, the primary steps to solve problem (9) in Algorithm 3 are summarized as follows: enumerate start time $t^-_{j^u}$ from 1 to τ_l and the possible number D_w of allocated workers from 1 to D_{j^u}; given above information, invoke Algorithm 4 and Algorithm 5 designed for problem (10) under two placement strategies, to produce allocations with minimal resource cost; compute the objective value and update current optimal schedule ι^* as well as corresponding revenue.

$$\min_{\zeta_j, \mathbf{y}, \mathbf{z}} \sum_s \sum_t \eta_w(t) y_{j^u s}(t) + \sum_s \sum_t \eta_p(t) z_{j^u s}(t) \tag{10}$$

subject to:

$$\text{Constraints } (3a) - (3b)(3e) - (3o)(9a)(9b), \quad \text{where } o_{j^u} = 1, \text{ specific } t^-_{j^u}, t^+_{j^u}$$

Algorithm Design. The sketch of one-round algorithm $A_{maxIrev}$ is in Algorithm 3. As mentioned above, the enumeration is

conducted in line 3. For each tuple of parameters, we get two schedules by invoking functions MINCOSTD() and MINCOSTC() (lines 4-5). Subsequently, we compare these results with the current optimal schedule $\iota^*_{j^u}$ and update it if necessary (lines 6-12).

Algorithm 3. One-Round Scheduling to Maximize Total Ideal Revenue: $A_{maxIrev}$

Input: $h_{sw}(t), h_{sp}(t), q^u_w(t), q^u_p(t), a_w(t), a_p(t), C_{sw}, C_{sp}, \forall s \in [S], \forall t \in (\lambda(i-1)\tau_l, \lambda i \tau_l]$
Output: $\iota^*_{j^u}, \mu_{j^u}$
1: Initialize $\mu_{j^u} = 0, \iota^*_{j^u} = \emptyset, x_{j^u w} = 0, x_{j^u p} = 0, y_{j^u s}(t) = 0, z_{j^u s}(t) = 0, \forall u, \forall j^u, \forall s, \forall t$
2: **for** $j^u \in J_l$ **do**
3: **for** each tuple $\{t^-_{j^u}, D_w\}, \forall t^-_{j^u} \in [1, \tau_l], D_w \in [1, D_{j^u}]$ **do**
4: Set $(cost, \iota) = \text{MINCOSTD}(\text{J}^\text{U}, \text{T}^-_{\text{J}^\text{U}}, \text{D}_\text{W}, \text{L}, \text{I})$
5: Set $(cost,' \iota') = \text{MINCOSTC}(\text{J}^\text{U}, \text{T}^-_{\text{J}^\text{U}}, \text{D}_\text{W}, \text{L}, \text{I})$
6: **if** $f_{j^u}(t^{+\iota'}_{j^u} - (\lambda + i - 1)\tau_l) - cost' \geq f_{j^u}(t^{+\iota}_{j^u} - (\lambda + i - 1)\tau_l) - cost$ **then**
7: Set $(cost, \iota) = (cost,' \iota')$
8: **end if**
9: Compute $\mu^\iota_{j^u} = f_{j^u}(t^{+\iota}_{j^u} - (\lambda + i - 1)\tau_l) - cost$
10: **if** $\mu^\iota_{j^u} > 0 \wedge \mu^\iota_{j^u} > \mu_{j^u}$ **then**
11: $\iota^*_{j^u} \leftarrow \iota, \mu_{j^u} = \mu^\iota_{j^u}$
12: **end if**
13: **end for**
14: **end for**
15: **Return** $\{\iota^*_{j^u}, \mu_{j^u}\}_{\forall j^u}$

Example. At the beginning, no jobs are scheduled and no resources are consumed, i.e., $h_w(t) = h_p(t) = 0, \forall t \in (8, 12]$. At this time, the prices of workers and PSs are 0 according to Eq. (8). For job j^{11}, its start time $t^-_{j^{11}}$ is within (8,12], and the possible number of workers allocated D_w is within $\{1, 2\}$. For each parameter combination $\{t^-_{j^{11}}, D_w\}$, we try to compute an optimal schedule under distributed and centralized placement, respectively. For example, when $\{t^-_{j^{11}} = 9, D_w = 1\}$, the training duration is $E_{j^{11}} D_{j^{11}} M_{j^{11}} / (n_{j^{11}} D_w) = 8$ under distributed placement, which exceeds the length of scheduling interval (8,12]. This implies that $\{t^-_{j^{11}} = 9, D_w = 1\}$ cannot generate a feasible distributed schedule. Under distributed placement, when $\{t^-_{j^{11}} = 9, D_w = 2\}$, the resource cost and job utility of job j^{11}'s optimal schedule are 0 and 92, respectively. Moreover, for the centralized placement, the corresponding utility of the optimal schedule is $\mu_{j^{11}} = 96$. Due to $96 > 92$, we actually schedule and allocate resources to job j^{11} according to the optimal schedule under centralized placement, i.e., all the resources of server s_1 are allocated to j^{11} during (8,10]. At this time, the prices of workers and PSs are updated as $h_w(t) = 2, h_p(t) = 1, \eta_w(2) = 16^{(2/8)} - 1 = 1, \eta_p(1) = 16^{(1/4)} - 1 = 1, \forall t \in (8, 10]$. Other jobs are computed in a similar way. Similarly, server s_2 is actually allocated to job j^{21} during (10,12], whose resource cost and utility are 0 and $\mu_{j^{21}} = -2 * 4 + 100 = 92$, respectively. The resource prices are updated as $h_w(t) = 2, h_p(t) = 1, \eta_w(2) = 16^{(2/8)} - 1 = 1, \quad \eta_p(1) = 16^{(1/4)} - 1 = 1, \forall t \in (10, 12]$. Jobs j^{12}, j^{22} are not scheduled due to the insufficient fair share. After the two users share the unused resources with each other, two workers and one PSs of server s_3 and one worker of server s_4 are allocated to job j^{12} during (8,12]. The resulting cost and job utility are $4 * 3 * 1 + 4 * 1 * 1 = 16$ and $-2 * 4 + 100 - 16 = 76$, respectively. The remaining unused resources are not enough to complete job j^{22}, which is postponed to the next round for scheduling.

Functions MINCOSTD() and MINCOSTC() are used to compute an allocation with minimal resource cost under distributed and centralized placement strategies, respectively. In function MINCOSTD(), we first compute the training duration d_{j^u}, actual start and completion time $(t^-_{j^u}, t^+_{j^u})$ in line 3. If $t^+_{j^u}$ exceeds the scheduling timespan of current round i or the remaining workers that are available cannot meet job demand, then the allocation will be terminated (lines 4-6). Otherwise, based on the above idea, we pick up the server that can employ minimal PSs to satisfy bandwidth constraint (3 i) to place the needed PSs (lines 10,14). Besides, the allocation of PSs cannot violate capacity constraint (3 d) and sharing constraint (3 b) (lines 7-9). For the remaining workers, we place them starting from the server with fewest resources (lines 17-20). In this way, servers with sufficient resources are reserved, as a result, the probabilities of centralized placement increase.

Algorithm 4. Function for Scheduling Job j^u Under Distributed Placement: MINCOSTD

1: **function** MINCOSTD $(j^u, t^-_{j^u}, D_w, l, i)$
2: Initialize $\iota = \emptyset, cost = 0, y_{j^u s}(t) = 0, z_{j^u s}(t) = 0, \forall s$
3: Compute $d_{j^u} = \lceil E_{j^u} D_{j^u} M_{j^u} / (n_{j^u} D_w) \rceil, t^-_{j^u} = (\lambda + i - 1)\tau_l + t^-_{j^u}, t^+_{j^u} = t^-_{j^u} + d_{j^u}$
4: **if** $t^+_{j^u} > (\lambda + i)\tau_l \parallel \min_{t \in [t^-_{j^u}, t^+_{j^u}]} \{\max\{q^u_w - \sum_{j^u} \sum_s y_{j^u s}(t), a_w(t)\}\} < D_w$ **then**
5: **Return** ∞, ι
6: **end if**
7: **for** $s = 1$ to S **do**
8: Set $D_{sw} = \min_{t \in [t^-_{j^u}, t^+_{j^u}]} \{D_w - 1, C_{sw} - h_{sw}(t)\}, D_{sp} = \min_{t \in [t^-_{j^u}, t^+_{j^u}]} \{\lceil (D_w - D_{sw}) b_w / b_p \rceil, C_{sp} - h_{sp}(t), \max\{q^u_w - \sum_{j^u} \sum_s y_{j^u s}(t), a_w(t)\}\}$
9: **end for**
10: Sort servers in $[H] = \{s | s \in [S] \wedge D_{sp} \geq 1\}$ according to D_{sp} in non-decreasing order into $s_1, s_2, \ldots, s_H$
11: **if** $H = 0$ **then** ▷ no enough PSs
12: **Return** ∞, ι
13: **end if**
14: Set $z_{j^u s_1}(t) = D_{s_1 p}, y_{j^u s_1}(t) = D_{s_1\, w}, \forall t \in [t^-_{j^u}, t^+_{j^u}]$
15: Set $D'_w = D_w - D_{s_1\, w}$
16: Sort servers $[H'] = [S] \setminus \{s_1\}$ according to $\min_{t \in [t^-_{j^u}, t^+_{j^u}]} \{C_{sw} - h_{sw}(t)\}$ in non-decreasing order into $s_{1'}, s_{2'}, \ldots, s_{H'}$
17: **for** $i = 1$ to H' **do** ▷ deploy the remaining workers
18: Set $y_{j^u s_i}(t) = \min\{\min_{t \in [t^-_{j^u}, t^+_{j^u}]} \{C_{s_i w} - h_{s_i w}(t)\}, D'_w\}, \forall t \in [t^-_{j^u}, t^+_{j^u}]$
19: $D'_w \leftarrow D'_w - y_{j^u s_i}(t^-_{j^u})$
20: **end for**
21: **Return** ∞, ι **if** $D'_w > 0$ ▷ no enough workers
22: Set $cost = \sum_{t \in [t^-_{j^u}, t^+_{j^u}]} \{\sum_s \eta_w(t) y_{j^u s}(t) + \eta_p(t) z_{j^u s_1}(t)\}$
23: **Return** $cost, \iota$
24: **end Function**

Example. When $\{t^-_{j^{11}} = 9, D_w = 2\}$, the training duration is 4, and the completion time $t^+_{j^{11}}$ is 12, which can be satisfied. Next, recalling constraints (3 f) and (3 h), we can know our algorithm does not allow preemption during the training course $[t^-_{j^{11}}, t^+_{j^{11}}]$ ([9,12]). Distributed placement implies

that at least one worker is not placed with the PSs. In line 8, we aim to find the target server that deploys the PSs. We first compute the maximum available workers $D_{sw} = \min_{t\in[t^-_{j11},t^+_{j11}]}\{2-1, 2-0\} = 1$ for each server s, and the needed PSs $D_{sp} = \min_{t\in[t^-_{j11},t^+_{j11}]}\{(2-1)*1/1, 1-0, \max\{2-0-0\}\} = 1$. Clearly, all the four servers are qualified, i.e., $[H] = \{s_1, s_2, s_3, s_4\}$. According to line 14, one worker and one PS in server s_1 are allocated to job j^{11}. The remaining one worker is selected from other servers s_2, s_3, s_4, which are ordered according to the number of available workers in non-decreasing order. Specifically, we select one worker of s_2 to job j^{11}. The corresponding cost is zero because the resource prices are zero. The ideal revenue is $f_{j^{11}}(t^+_{j11}) = -2*(t^+_{j11} - (\lambda+i-1)\tau_l) + 100 = -2*(12-8)+100 = 92$ and the job utility $\mu_{j^{11}} = 92 > 0$ in Eq. (7). After arranging job j^{11}, the number of workers consumed is $h_w(t) = 2, \forall t \in (8, 12]$ and the price of workers is updated as $\eta_w(2) = 16^{(2/8)} - 1 = 1, \forall t \in (8, 12]$ by assuming $\theta_w = 16$. Similarly, $h_p(t) = 1, \forall t \in (8, 12]$ and $\eta_p(1) = 16^{(1/4)} - 1 = 1, \forall t \in (8, 12]$ by assuming $\theta_p = 16$. Note that the above schedule is only a simulated, and the actual optimal schedule is determined by Algorithm 3. Assuming that the above schedule is actually adopted by Algorithm 3, then by repeating the above process, we can allocate one worker and one PS of server s_3 and one worker of s_4 to job j^{21}. The resulting cost is $4*2*1+4*1*1 = 12$ and the ideal revenue is $f_{j^{21}} = -2*(12-8)+100 = 92$. Substituting them into Eq. (7), the job utility $\mu_{j^{21}} = 92 - 12 > 0$, then it is also a feasible solution. In a similar way, the schedule of job j^{12} is computed with $a_w(t) = 4, a_p(t) = 2, \forall t \in (8, 12]$.

For function MINCOSTC(), lines 3-6 are run for the same reason. Under centralized placement, the number of PSs is exactly 1. Any server that can accommodate all workers and one PS while satisfying capacity constraint (3 c),(3 d) and sharing constraint (3 b) is qualified (line 7). Thus, we can choose any such server (line 11).

Algorithm 5. Function for Scheduling Job j^u Under Centralized Placement: MINCOSTC

1: **function** MINCOSTC($j^u, t^-_{j^u}, D_w, l, i$)
2: Initialize $cost = \infty, \iota = \emptyset, y_{j^u s}(t) = 0, z_{j^u s}(t) = 0, \forall s$
3: Compute $d_{j^u} = \lceil E_{j^u} D_{j^u} M_{j^u} / (n_{j^u} D_w) \rceil, t^-_{j^u} = (\lambda+i-1)\tau_l + t^-_{j^u}, t^+_{j^u} = t^-_{j^u} + d_{j^u}$
4: **if** $t^+_{j^u} > (\lambda+i)\tau_l \parallel \min_{t\in[t^-_{j^u},t^+_{j^u}]}\{\max\{q^u_w - \sum_{j^u} \sum_s y_{j^u s}(t), a_w(t)\}\} < D_w$ **then**
5: **Return** ∞, ι
6: **end if**
7: Select servers in $[H] = \{s | \min_{t\in[t^-_{j^u},t^+_{j^u}]}\{C_{sw} - h_{sw}(t)\} \geq D_w \wedge \min_{t\in[t^-_{j^u},t^+_{j^u}]} \{C_{sp} - h_{sp}(t), \max\{q^u_w - \sum_{j^u}\sum_s y_{j^u s}(t), a_w(t)\}\} \geq 1\}$
8: **if** $H = 0$ **then** ▷ no enough workers or PSs
9: **Return** ∞, ι
10: **else**
11: Set $y_{j^u s_1}(t) = D_w, z_{j^u s_1}(t) = 1, \forall t \in [t^-_{j^u}, t^+_{j^u}]$
12: Set $cost = d_{j^u}(\eta_w(t^-_{j^u})D_w + \eta_p(t^-_{j^u})), \iota \leftarrow \{\mathbf{y}, \mathbf{z}\}$
13: **Return** $cost, \iota$
14: **end if**
15: **end Function**

Example. The schedule under centralized placement is relatively simple. We can select any one server whose workers and PSs are enough to complete a job. For example, when $\{t^-_{j11} = 9, D_w = 2\}$, we can allocate two workers and one PS of server s_1 to j^{11}. The training duration $t^+_{j11} = 2*2*1/(1*2) = 2$ and the job utility $\mu_{j^{11}} = -2*2+100 = 96 > 0$. Other jobs can be scheduled similarly, and we omit the details here.

4.5 Theoretical Analysis

We first prove the optimality of Algorithm 3 in Lemma 2, and compute its approximation ratio in Lemma 3. Next, according to Lemmas 1 and 3, we analyze the performance of *Astraea* in terms of performance guarantee, correctness, time complexity, strategy-proofness and Pareto efficiency.

Lemma 2. *(Optimality). Algorithm 3 along with functions* MINCOSTC() *and* MINCOSTD() *produces an optimal solution to problem (9).*

Proof. Please see Appendix B., available in the online supplemental material □

Lemma 3. *(Approximation Ratio). For problem (4), $A_{maxIrev}$ is $2(\log\theta_w + \log\theta_p)$-approximation.*

Proof. Please see Appendix C., available in the online supplemental material □

Theorem 1. *(Performance Guarantee). We analyze the performance guarantee of Astraea with the following two generalized forms of utility functions.*

(i) Linear function: if the utility function of job j^u is linear of the form $k_{j^u} t^+_{j^u} + g_{j^u}$, where $k_{j^u} < 0, g_{j^u} > 0$, then the difference of objectives computed by the optimal algorithm and Astraea is bounded by a constant, $(\sum_u J^u)((2^L-1)k_{max} - (2^{L+1}-1)(2\lambda-1)k_{min})$, where $k_{min} = \min_{j\in\bigcup_u [J^u]}\{k_j\}$, $k_{max} = \max_{j\in\bigcup_u [J^u]}\{k_j\}$.

(ii) Convex function: if the utility function of job j^u is convex of the form $(\frac{e_{j^u}}{t^+_{j^u}})^{\kappa_{j^u}}$, where $e_{j^u}, \kappa_{j^u} > 0$, then the competition ratio of Astraea is $(\frac{1}{4\lambda})^{\kappa_{max}} (\frac{1}{2^{L-2}})^{\kappa_{max}-\kappa_{min}} \frac{e_{min}{}^{\kappa_{max}}}{e_{max}{}^{\kappa_{min}}}$, where $\kappa_{min} = \min_{j\in\bigcup_u [J^u]}\{\kappa_j\}$, $\kappa_{max} = \max_{j\in\bigcup_u [J^u]}\{\kappa_j\}, e_{min} = \min_{j\in\bigcup_u [J^u]}\{e_j\}$, $e_{max} = \max_{j\in\bigcup_u [J^u]}\{e_j\}$. The competition ratio *is the lower bound of the ratio of the objective value returned by our online algorithm of problem (3) to that computed by the optimal offline algorithm.*

Proof. Please see Appendix D., available in the online supplemental material □

Theorem 2. *(Correctness). Astraea generates a feasible solution to the original problem (3).*

Proof. Please see Appendix E., available in the online supplemental material □

Theorem 3. *(Polynomial Time). The time complexity of Astraea is $\lambda L \sum_u J^u T D_{j^u}(S \log S)$.*

Proof. Please see Appendix F., available in the online supplemental material □

Theorem 4. *(Strategy-proofness). Astraea is strategy-proof.*

Proof. Please see Appendix G., available in the online supplemental material □

Theorem 5. *(Pareto Efficiency). Astraea is Pareto efficient.*

Proof. Please see Appendix H., available in the online supplemental material □

5 Performance Evaluation

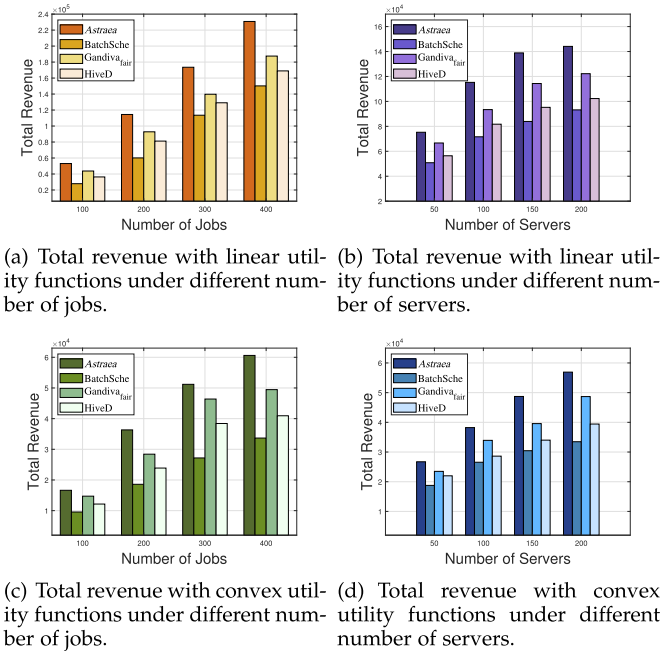
5.1 Experimental Setup

Settings. We conduct trace-driven simulation using the Microsoft Philly trace [39], [40], which contains the information of 117325 DNN jobs and around 550 machines over 14 virtual clusters. Based on these data, we simulate a distributed ML system with 14 users corresponding to 14 virtual clusters, and 100 servers. The two values are set by default and can be adjusted according to experimental needs. For each server, the number of workers is set to be that of GPUs in each machine. Since there is only one CPU in each machine, we set the number of PSs to 1. Other server information that is not given in the trace, is configured according to [5], [30], [33]. The bandwidth (b_w, b_p) of workers and PSs is set to be $[100, 5 * 1024]$ Mbps and [5,20] Gbps, respectively. Moreover, we extract the correspondence between virtual clusters and machines to count each user's fair share of resources. We generate deep learning (DL) jobs as the experimental workload. The contention between jobs are remained by intercepting the original real-world traces of a small continuous timespan. For the extracted trace, we map the submission time of jobs into arrival time (r_{j^u}) by setting one time slot to one hour long. And the number of GPUs requested by jobs is regarded as their demand for workers. Following the same way of configuring servers, the jobs' information not mentioned in the trace is set as follows: $E_{j^u} \in [50, 100], D_{j^u} \in [20, 50], M_{j^u} \in [10, 50], \quad m_{j^u} \in [0.001, 0.05]$ hour, $G_{j^u} \in [10, 100]$ milliseconds, $\epsilon_{j^u} \in [30, 575]$ MB. The parameters of utility functions are set by the authors as follows: $k_{j^u} \in [-2.0, -0.1], g_{j^u} \in [100, 1000], e_{j^u} \in [100, 1000], \kappa_{j^u} \in [1.0, 2.0]$.

Baselines. We compare *Astraea* with three state-of-the-art and representative ML job scheduling schemes - BatchSche [30], $Gandiva_{fair}$ [14] and HiveD [25]; BatchSche is the possible best benchmark for maximizing efficiency, whereas $Gandiva_{fair}$ and HiveD aim at achieving fairness. We implement these baselines in our simulator as described below:

- BatchSche: BatchSche provides a non-fair scheduling algorithm for ML jobs. The system setting in this study is similar to our work, but there is no the notion of users and sharing. We perform this algorithm for each user to schedule its jobs in its private cluster.
- $Gandiva_{fair}$: $Gandiva_{fair}$ is one of the state-of-the-art fair schedulers for ML jobs, which provides inter-user fairness and transparent heterogeneity support.
- HiveD: HiveD is the latest fair scheduler so far, which ensures sharing safety by binding each user to a virtual private cluster (VC). Then it improves cluster efficiency by dynamically binding of GPUs in a physical cluster to VCs.

Metrics. (i) Efficiency: We use the total revenue resulted from Algorithm 1 to evaluate how efficiently the cluster is utilized. (ii) Revenue Fairness: according to sharing incentive, if a resource allocation makes the ratio of tenant u' gain to collective revenue is at least the tenant's initial share $(\frac{q_w^u / \sum_u q_w^u + q_p^u / \sum_u q_p^u}{2})$, then such allocation is *fair* to the tenant, otherwise it is *unfair*. As analyzed in Section 3.2, a tenant' gain denoted as g_u is composed of three parts: $\sum_{j_1^u \in [J_1^u]} f_{j_1^u}(t_{j_1^u}^+)$, $\sum_{t=1}^{T} \sum_{u' \in [U] \setminus \{u\}} \rho_{j^{u'}}^u(t)$ and $\sum_{j_2^u \in [J^u] \setminus [J_1^u]} (f_{j_2^u}(t_{j_2^u}^+) - c_{j_2^u w} - c_{j_2^u p})$, where $[J_1^u]$ denotes the jobs that are processed using user u' share of resources. Then we define the revenue fairness ratio for each tenant u as $\alpha_u = \frac{g_u}{\sum_{u \in [U]} \sum_{j^u \in [J^u]} f_{j^u}(t_{j^u}^+)} \div \frac{q_w^u / \sum_u q_w^u + q_p^u / \sum_u q_p^u}{2}$, i.e., the ratio of the tenant's share of revenue to its initial share. $\alpha_u = 1$ means absolute revenue fairness. $\alpha_u > 1$ implies the corresponding tenant benefits from resource sharing while $\alpha_u < 1$ implies the tenant suffers economic losses. (iii) Performance Ratio: We define the performance ratio as the ratio of experimental total revenue computed by *Astraea* to that returned by the offline optimal algorithm, and observe the actual bound of our algorithm.



(a) Total revenue with linear utility functions under different number of jobs.
(b) Total revenue with linear utility functions under different number of servers.
(c) Total revenue with convex utility functions under different number of jobs.
(d) Total revenue with convex utility functions under different number of servers.

Fig. 4. Total revenue with different setting.

5.2 Result Analysis

Note that there are many random variables in our setting, so all the experiments results are the average of 5 trials.

Efficiency. Fig. 4 shows the comparisons of the total revenue of four methods under different number of jobs and servers and two forms of utility functions, respectively. First, irrespective of the forms of utility functions, the overall trends of *Astraea* and other baselines are consistent. As a result, we take the case of a linear function (Figs. 4a and 4b) as an example to analyze the experimental results as follows. The collective revenue grows with the increase of jobs in Fig. 4a, which is attributed to more serviced jobs. As for the similar trend in Fig. 4a, the reason is that the completion time of jobs are shortened with richer resources. Specifically, *Astraea* has the total revenue around 1.58X, 1.24X and 1.35X (1.63X, 1.22X and 1.46X) better than BatchSche,

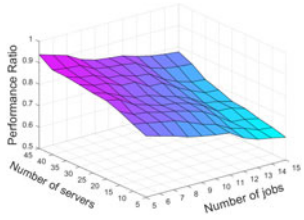


Fig. 5. Performance ratio of *Astraea* under different number of jobs and servers.

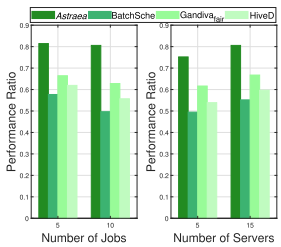


Fig. 6. Comparison of performance ratio across schedulers.

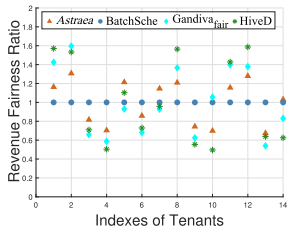


Fig. 7. Comparison of revenue fairness across schedulers.

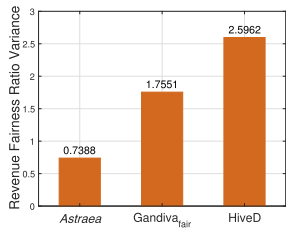


Fig. 8. Variance of users' revenue fairness ratios across all baselines.

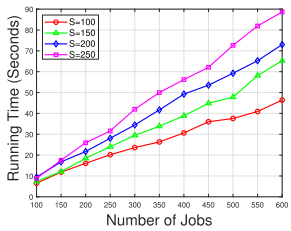


Fig. 9. Running time of *Astraea* with different number of jobs.

$Gandiva_{fair}$ and HiveD with different number of jobs (servers), respectively. It is because BatchSche does not share the unused resources among all users. $Gandiva_{fair}$ pursues the fair division of GPUs among users, while not considering the communication time of different placement of workers and PSs. HiveD reserves the GPU affinity for each user to guarantee sharing safety, which restricts resource sharing with others. Such sharing provides the highest level of performance isolation. But it may also hinder the full utilization of resources, and further sacrifices the system efficiencies to some extend.

Performance Ratio. Fig. 5 shows the performance ratio of *Astraea* changes with the number of jobs and servers. Due to the high time complexity of the offline optimal algorithm, we consider a reduced input with [5, 15] jobs of 2 users and [5, 45] servers. We observe that the performance ratio decreases with the increasing size of jobs and grows as the number of servers increases. Besides, the observed performance ratio remains at an acceptable level (> 0.6), which implies the availability of *Astraea* in practice. Besides, the performance ratio decreases with increasing job size, and the reasons may be as follows. When the workload increases and resources become scarce, different schedulers' effects on system efficiency get larger. The offline optimal algorithm has the foresight of all workload and thus always generates an optimal schedule. In contrast, our algorithm runs online and makes scheduling decisions irrevocably. As a result, the performance difference between our algorithm and the offline optimal one widens. We computed the difference in performance ratio when the number of jobs is 5 and 15, respectively, and the number of servers varies from 5 to 45. The maximum difference is not larger than 0.22, which is probably not a significant drop.

Fig. 6 shows the comparisons of the performance ratio of *Astraea* and other algorithms. In any settings, *Astraea* consistently outperforms others in their category. Besides, the performance ratio of *Astraea* decreases more gentle than other baselines.

Revenue Fairness. We evaluate the revenue fairness ratio of four algorithms for their 14 tenants, and present their comparison in Fig. 7. We observe the following facts. First, BatchSche achieves absolute revenue fairness since there are no resource sharing between tenants. However, BatchSche also results in the worst efficiency for all setting due to lack of resource sharing, as shown in Figs. 4a, 4b, 4c, and 4d and Fig. 6.

Second, compared to the remaining baselines, *Astraea* enables more tenants to achieve revenue fairness, since these baselines assumes that all tenants selflessly contribute their spare shares of resources without any compensation. Besides, the revenue fairness ratios of all tenants in *Astraea* fluctuates around 1, and its variance is much smaller than that of other baselines as shown in Fig. 8. This fact implies that the compensation in *Astraea* can decrease the loss of unfair tenants as much as possible, which contributes to healthy cooperation between tenants.

Running Time of Astraea. We run *Astraea* with different number of jobs and servers on a laptop (Intel Core i5-8250 U/8 GB RAM) and present the average running time in Figs. 9 and 10, respectively. We can observe the execution time of *Astraea* grows with the increasing number of jobs or servers. But the time to compute a schedule for a job is about 1.56 seconds under different setting. Compared to the long time to train a ML job, such decision time is acceptable.

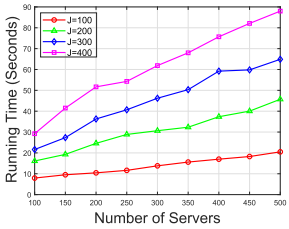


Fig. 10. Running time of *Astraea* with different number of servers.

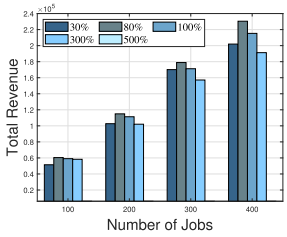


Fig. 11. Total revenue under inaccurate maximal price of workers and PSs.

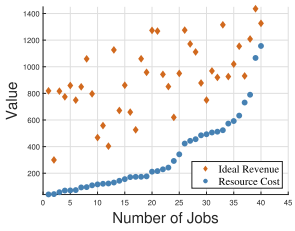


Fig. 12. Ideal Revenue versus resource cost of jobs.

Setting of Maximum Price. The maximal price (θ_w, θ_p) of workers and PSs are estimated based on past experiments. To study the effect of this estimation on our algorithm, we use θ_w and θ_p with different percentages of their actual values as the input of *Astraea*, and demonstrate the results in Fig. 11. We observe that when resources are scarce due to high load, an appropriate underestimation (80%) leads to higher total revenue compared to overestimation. Since it prevents abrupt price rise, which may filter out jobs that should be accepted.

Admission Control. Fig. 12 demonstrates the resource cost and ideal revenue of jobs. We can observe that the ideal revenue of each job is always larger than its resource cost, which is consistent with the admission control ($o_{j^u} = 1$) in each round.

Resource Utilization. Fig. 13 shows the average GPU utilization of all schedulers. The results show that the utilization of *Astraea* and $Gandiva_{fair}$ can reach 100% due to the full use of resources. However, BatchSche and HiveD can rarely reach 100% resource utilization due to disabling sharing and high-level performance isolation, respectively. Moreover, we observe that *Astraea* can get off-peak faster than $Gandiva_{fair}$ since *Astraea* computes an optimal schedule for each job while considering placement sensitivity, whereas $Gandiva_{fair}$ ignores these effects. Consequently, the makespan of *Astraea* is about $0.83\times, 0.67\times$ smaller than that of $Gandiva_{fair}$ and HiveD, respectively, and is much smaller than BatchSche.

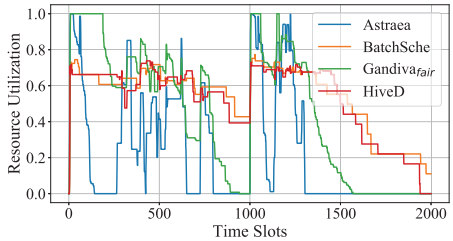


Fig. 13. Resource utilization of four algorithms.

TABLE 2
A Sensitivity Study on *Astraea*'s Efficiency and Revenue Fairness

# of Users	Revenue per Job	Revenue Fairness Ratio Variance
5	599.64	0.7053
8	571.55	0.78126
11	611.28	0.7979
14	581.06	0.7388

Algorithm Sensitivity. We next study how the efficiency and revenue fairness achieved by *Astraea* are affected by the number of users. To this end, we compute the revenue per job and revenue fairness ratio variance under the different number of users, whose results are presented in Table 2. The ratio between maximum revenue per job to minimum one is around 1.069, which is a small value. Similarly, the ratio of maximum variance to minimum one is about 1.131, which is also relatively stable. Based on the two observations, we can conclude that the efficiency and revenue fairness obtained by *Astraea* is not sensitive to the number of users.

6 Discussion

In this section, we discuss how to generalize *Astraea* to multiple resource types. To implement this generalization, two problems need to be tackled. The first issue is to achieve gain-as-you-contribute fairness in multiple resource types, i.e., reformulate Eq. (2). The other problem is to modify the resource allocation proposed in Section 4.4. Next, we briefly explain the solutions to the above two problems. We first introduce the number of worker types and PS types, denoted as W and P. Then w, p are redefined as a specified worker type and PS type by abusing the notions. Similarly, other notions containing w or p are correlated to resource types. For example, q_w^u are redefined as the number of type-w workers of user u. As for decision variables $y_{j^u s}(t)$ and $z_{j^u s}(t)$, they are remodeled as $y_{j^u sw}(t)$ and $z_{j^u sp}(t)$. Then by assuming job $j^{u'}$ employs type-w workers and type-p PSs, Eq. (2) is reformulated as follows:

$$\rho_{j^{u'}}^u(t) = \frac{q_w^u - \sum_{j^u}\sum_s y_{j^u sw}(t)}{a_w(t)} c_{j^{u'}w} + \frac{q_p^u - \sum_{j^u}\sum_s z_{j^u sp}(t)}{a_p(t)} c_{j^{u'}p} \quad (11)$$

So far the gain-as-you-contribute fairness of multiple resource types is achieved. As for the resource allocation of multiple types, we can reuse the algorithm idea described in Section 4.1 to reformulate the model with multiple resource types. By re-interpreting dual variables $\eta_{sw}(t)$ and $\eta_{sp}(t)$ as the unit prices of type-w workers and type-p PSs on server s at time t, we correlate price functions in Eq. (8) with resource types. In this way, the resource allocation can be resolved. Meanwhile, the properties in Section 4.5 can also remained in the setting of multiple resource types.

7 Conclusion

In this paper, we focus on the design of an online scheduling framework that incentivizes users to share. In the framework, we first propose a batch framework to resolve the uncertainty in sharing incurred by online setting, which further evades migration and preemption of jobs. Then for the offline batch scheduling problem, we design a round-by-round algorithm along with an one-round subroutine to solve it. Moreover, we extent the performance guarantee from the problem of minimizing the total weighted completion time of jobs to one of maximizing total revenue, where each item is a linear or convex function with completion time of the job. Finally, we conduct trace-driven simulations to demonstrate that the proposed algorithm performs better than three state-of-the-art baselines. A meaningful extension might be to improve the level of performance isolation by considering the GPU affinity, while minimizing the reduction in cluster efficiency.

References

[1] A. Vaswani *et al.*, "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[3] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 1725–1732.

[4] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[5] 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. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2592–2600.

[6] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, pp. 1–15.

[7] R. Chaiken *et al.*, "SCOPE: Easy and efficient parallel processing of massive data sets," *Proc. VLDB Endowment*, vol. 1, pp. 1265–1276, 2008.

[8] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1–17.

[9] H. Liu and B. He, "F2C: Enabling fair and fine-grained resource sharing in multi-tenant IaaS clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2589–2602, Sep. 2016.

[10] Amazon EC2 pricing, 2021. [Online]. Available: https://aws.amazon.com/cn/ec2/pricing/?nc2=type_a

[11] Rackspace, 2021. [Online]. Available: https://www.rackspace.com/cloud/public

[12] Groupon, 2021. [Online]. Available: http://www.groupon.com/

[13] Teambuy, 2021. [Online]. Available: http://www.teambuy.ca/

[14] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha, "Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning," in *Proc. ACM 15th Eur. Conf. Comput. Syst.*, 2020, pp. 1–16.

[15] J. Gu *et al.*, "Tiresias: A GPU cluster manager for distributed deep learning," in *Proc. 16th USENIX Conf. Netw. Syst. Des. Implementation*, 2019, pp. 485–500.

[16] K. Mahajan *et al.*, "THEMIS: Fair and efficient GPU cluster scheduling," in *Proc. 17th USENIX Conf. Netw. Syst. Des. Implementation*, 2020, pp. 289–304.

[17] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Reading, MA, USA: Addison-Wesley., 1997.

[18] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. 8th USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 323–336.

[19] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-Min fair sharing for datacenter jobs with constraints," in *Proc. 8th ACM Eur. Conf. Comput. Syst.*, 2013, pp. 365–378.

[20] R. Li and P. Patras, "Max-min fair resource allocation in millimetre-wave backhauls," *IEEE Trans. Mobile Comput.*, vol. 19, no. 8, pp. 1879–1895, Aug. 2020.

[21] C. You, "Hierarchical multi-resource fair queueing for network function virtualization," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 406–414.

[22] S. M. Zahedi and B. C. Lee, "REF: Resource elasticity fairness with sharing incentives for multiprocessors," in *Proc. ACM SIGPLAN Notices*, vol. 49, pp. 145–160, 2014.

[23] Hadoop: Fair Scheduler, 2017. [Online]. Available: https://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/FairScheduler.html

[24] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 455–466.

[25] H. Zhao *et al.*, "HiveD: Sharing a GPU cluster for deep learning with guarantees," in *Proc. 14th USENIX Conf. Operating Syst. Des. Implementation*, 2020, pp. 515–532.

[26] S. Barberà, D. Berga, and B. Moreno, "Individual versus group strategy-proofness: When do they coincide?," *J. Econ. Theory*, vol. 145, no. 5, pp. 1648–1674, 2010.

[27] G. Christodoulou, K. Mehlhorn, and E. Pyrga, "Improving the price of anarchy for selfish routing via coordination mechanisms," in *Proc. Eur. Symp. Algorithms*, 2011, pp. 119–130.

[28] R. Zhou, Z. Li, C. Wu, and Z. Huang, "An efficient cloud market mechanism for computing jobs with soft deadlines," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 793–805, Apr. 2017.

[29] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms," *Math. Oper. Res.*, vol. 22, no. 3, pp. 513–544, 1997.

[30] Q. Zhang, R. Zhou, C. Wu, L. Jiao, and Z. Li, "Online scheduling of heterogeneous distributed machine learning jobs," in *Proc. Int. Symp. Theory Algorithmic Found. Protoc. Des. Mobile Netw. Mobile Comput.*, 2020, pp. 111–120.

[31] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–14.

[32] W. Xiao *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. 13th USENIX Conf. Oper. Syst. Des. Implementation*, 2018, pp. 595–610.

[33] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 495–503.

[34] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *Proc. 14th USENIX Conf. Oper. Syst. Des. Implementation*, 2020, pp. 481–498.

[35] V. K. Vavilapalli *et al.*, "Apache hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, pp. 1–16.

[36] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. Int. Conf. Big Data Sci. Comput.*, 2014, Art. no. 1.

[37] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 1355–1364.

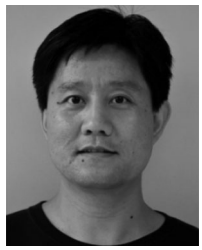
[38] G. Gens and E. Levner, "Complexity of approximation algorithms for combinatorial problems: A survey," *SIGACT News*, vol. 12, no. 3, pp. 52–65, 1980.
[39] Microsoft Philly Trace, 2019. [Online]. Available: https://github.com/msr-fiddle/philly-traces
[40] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 947–960.
[41] Z. Huang and A. Kim, "Welfare maximization with production costs: A primal dual approach," in *Proc. 26th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2015, pp. 59–72.

Ne Wang received the BE degree from theSchool of Computer Science and Technology, Wuhan University of Technology, China, in 2016. Since September, 2019, she is currently working toword the PhD degree with the School of Computer Science, Wuhan University, China. Her research interests include edge computing, distributed machine learning optimization and online scheduling.

Ruiting Zhou (Member, IEEE) received the PhD degree from the Department of Computer Science, University of Calgary, Canada, in 2018. She has been an Associate Professor with the School of Cyber Science and Engineering, Wuhan University since June 2018. 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 the IEEE INFOCOM, ACM MobiHoc, ICDCS, *IEEE/ACM Transactions on Networking*, *IEEE Journal on Selected Areas in Communications*, and *IEEE Transactions on Mobile Computing*. She also serves as a reviewer for journals and international conferences such as the *IEEE Journal on Selected Areas in Communications*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Wireless Communications*, and *IEEE/ACM IWQOS*.

Ling Han received the bachelor's and master's degrees from Wuhan University, in 2019 and 2022, respectively. He is currently working toward the PhD degree in computer sciencewith Duke University from 2022. His research interests include blockchain, machine learning, database and cybersecurity, in which areas he has achieved several results in forms of research papers and patents.

Hao Chen (Member, IEEE) received the PhD degree in electronic science and engineering from Southeast University, in 2012. Currently, he is a senior engineer with Huawei Nanjing Research Institute, Nanjing, China. His research interests include storage network, and resource management in cloud computing environment.



Zongpeng Li (Senior Member, IEEE) received the BE degree in computer science from Tsinghua University, in 1999, and the PhD degree from the University of Toronto, in 2005. He has been with the University of Calgary and then Wuhan University. His research interests include in computer networks and cloud computing. He 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. He co-authored papers that received best paper awards at the following conferences: PAM 2008, HotPOST 2012, and ACM e-Energy 2016. He 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.