Eris: An Online Auction for Scheduling Unbiased Distributed Learning Over Edge Networks

Jinlong Pang ⚫, Ziyi Han, Ruiting Zhou ⚫, Member, IEEE, Renli Zhang ⚫, John C.S. Lui ⚫, Fellow, IEEE, and Hao Chen

Abstract—The emergence of edge intelligence has made smart IoT services (e.g., video/audio surveillance, autonomous driving and smart city) a reality. To ensure the quality of service, edge service providers train unbiased models of distributed machine learning jobs over the local datasets collected by edge networks, and usually adopt the parameter server (PS) architecture. However, the training of unbiased distributed learning (UDL) depends on geo-distributed data and edge resources, bringing a new challenge for service providers: how to effectively schedule and price UDL jobs such that the long-term system utility (i.e., social welfare) can be maximized. In this paper, we propose an online auction-based scheduling algorithm Eris, which determines the data workload, the number and the placement of concurrent workers and PSs for each arriving UDL job, and dynamically prices limited edge resources based on current resource consumption. Eris applies a primal-dual framework which calls an efficient dual subroutine to schedule UDL jobs, achieving a good competitive ratio and pseudo-polynomial time complexity. To evaluate the effectiveness of Eris, we implement both a testbed and a large-scaled simulator. The results demonstrate that Eris outperforms and achieves up to 44% more social welfare compared to state-of-the-art algorithms in today’s cloud system.

Index Terms—Distributed machine learning, auction, online scheduling.

I. INTRODUCTION

The combination of edge computing and artificial intelligence (AI) introduced a new promising paradigm, namely “edge intelligence” [1]. Instead of relying on the cloud entirely, edge intelligence utilizes massive data collected in IoT devices and edge servers to provide real-time services for smart applications, e.g., intelligent video/audio surveillance, smart city and autonomous driving. To ensure IoT service’s quality, it is critical to train an unbiased model1 with the data collected by edge sites [3], [4], where an edge site can be an edge server attached to an access point (e.g., WiFi hotspots or base station), an edge cloudlet, or even a network gateway. For example, to deploy smart traffic management system, real-time road traffic data collected by installed cameras needs to be uploaded to nearby edge cloudlets for processing [5], [6], [7]. The collected data from different edge cloudlets are used to train an unbiased model which can work in all roads to manage the traffic. In this work, we define this type of learning jobs as the unbiased distributed learning (UDL) jobs. Unlike traditional distributed learning, UDL jobs source their training data from edge sites, which offer both computational resources and the necessary training data. Note that one common approach to training UDL jobs is to adopt data parallelism with the “parameter server (PS)” framework at the edge [3]. In each training iteration, the workers deployed for one UDL job (either implemented on virtual machines (VM) or cloud containers) compute gradients based on input datasets, and then send gradients to PSs. The PSs update model parameters by averaging gradients and then push parameters to all workers.

For the service provider, who owns the training data and edge resources, there exists two unique concerns need to be addressed when training UDL jobs: technical and economic. First, on the technical side, given heterogeneous resources (especially scarce and expensive bandwidth) and data volumes in different edge sites, it is challenging to decide the training data workload in each site, and deploy the matched number of workers, such that the average job completion time is minimized and total bandwidth consumption is as small as possible. Furthermore, due to the frequent communication between workers and PSs, the placement of PSs also affects the inter-site bandwidth consumption and the job completion time if workers and PSs are not deployed on the same site. Therefore, the training of UDL jobs needs to jointly optimize bandwidth consumption and minimize the average job completion time. Second, on the economic side, it is non-trivial to price limited edge resources upon the arrival of each UDL job without any future information such that the system’s utility in the long run (i.e., social welfare) can be maximized. One tailored pricing scheme is needed to charge UDL jobs, considering both specific features of the edge (i.e., resource scarcity) and UDL jobs (i.e., geo-distributed data).

1An unbiased model represents a model with high applicability that can be used in all sites to provide effective inference [2].
There have been some efforts to address the above two concerns. For the technical side, commonly adopted schedulers in cloud platforms [8], [9] follow intuitive strategies, e.g., First In First Out (FIFO) or Dominant Resource Fairness Scheduling [10]. However, for these cloud schedulers, providing the fixed configuration, i.e., the number of workers/PSs and the training time, is mandatory, which leads to inflexibility of resource scheduling [11]. For the edge network, recent work mainly focuses on task offloading problems [12], [13], [14]. For the economic side, fixed pricing, where charging a fixed unit price for the utilized resources based on jobs’ training time, is commonly adopted [8], [15]. However, fixed pricing fails to capture the changing supply-demand relationship in the market. Consequently, the case of overpricing or underpricing routinely happens, which will jeopardize the service operator’s profit as well as the utility of the entire system. Therefore, fixed pricing is not suitable for UDL jobs due to the uncertainty of job’s training time, which depends on the job placement. In this regard, auction is a natural pricing approach which can automatically set the right price and further allocate resources to users who value them most. Auction can further enable the users to bid for any combination of resources or configurations. Besides, auction-based mechanisms can effectively guarantee truthfulness and individual rationality. A number of auction-based mechanisms in different perspectives [16], [17], [18], [19], [20], [21] have been proposed, which focus both on pricing mechanism and resource allocation. However, these auction-based pricing mechanisms do not compatible with ML jobs with uncertain training time. More details are presented in Section II.

Herein, we develop an auction-based framework Eris for UDL jobs to tackle both the technical and economic challenges. Eris dynamically schedules each UDL job upon its arrival under the bandwidth consumption constraint to maximize social welfare, which is equivalent to minimizing the average job completion time (because a UDL job’s utility is non-increasing with its job completion time). On the other hand, Eris applies auction mechanisms to automatically price limited edge resources. Eris mainly consists of two modules: 1) an online scheduling algorithm that decides the amount of training data, the execution time window, the number and the placement of concurrent workers and PSs at each time slot upon the arrival of each UDL job; and 2) one well-designed pricing scheme that adjusts resource price (including bandwidth) over time based on the current resource usage to prevent users from submitting jobs and enable the operator to avoid resource starvation. To the best of our knowledge, this work is the first formal study of dynamic scheduling and pricing for UDL jobs at edge networks.

Specifically, our contributions are:

- Through capturing UDL jobs’ distinct features, we model the social welfare maximization problem into a NP-hard mixed integer nonlinear program (MINLP). To tackle this problem, we first reformulate this problem into a typical knapsack-type problem by leveraging exponential number of variables.
- An effective auction-based scheduling algorithm Eris is presented. First, Eris estimates the resources cost for each job by using the dual variables which can be interpreted as unit resource prices. The price function reflects the market rules and is computed based on the resource consumption. Given the resource prices, Eris then calculates the optimal schedule for each accepted job via a dynamic programming algorithm.

Ⅲ. RELATED WORK

Scheduling and placing in Edge/Cloud: In the energy-constrained scenario, to minimize the latency, Saleem et al. [13] investigate the mobility-aware task scheduling problem. Alameddine et al. [14] attempt to maximize the number of admitted latency-sensitive tasks by optimizing task offloading and scheduling. However, the aforementioned work usually allocate resources for tasks by leveraging virtual machine (VM) or cloud container rather than workers/PSs using the typical “parameter server (PS)” architecture. Therefore, these work are not applicable for smart IoT applications. In cloud networks, how to make optimal decisions for job scheduling and placing has been explored over the last decades. Several respective cloud platforms are already in our sights, e.g., Microsoft Azure ML [9], Amazon AWS ML [15] and Google Cloud AI [8]. Nevertheless, these platforms only apply intuitive schedulers, for example, FIFO or Dominant Resource Fairness Scheduling (DRF) [10], which can not maximize resource utilization. Other recent proposed schedulers [24], [25], [26], [27], [28] focus on maximizing resource utilization, but usually do not take the data transmission consumption into consideration. However, in the edge network, this type of consumption can not be overlooked due to the bandwidth is scarce and expensive. Our previous work [29] identifies UDL jobs' distinct characteristics (training data is geo-distributed over sites), and discusses the dynamic deployment of workers and PSs for UDL jobs to minimize the overall training time. Nonetheless, in this paper, we investigate a totally different problem focusing on overall utility maximization (social welfare). Besides, to avoid overpricing or underpricing, tailored pricing functions based on market rules are further presented.

Auction Mechanism in Edge/Cloud: Wu et al. [16] propose a specific auction to maximize social welfare for a mobile
edge computing (MEC) system. Mashhadi et al. [18] design a truthful auction for MEC edge servers that can maximize overall profit without compromising mobile devices’ energy consumption and delay requirements. Lin et al. [17] present an auction framework to effectively allocate resources to maximize edge servers’ utilities meanwhile ensuring the latency constraint for computing tasks of end-devices. In [19], an online auction is developed to maximize cloud provider’s revenue in a cloud computing system. Zhang et al. [20] first discuss the heterogeneity of user demands within an auction scheme. Li et al. [21] focus on resource pricing and design an auction to achieve trade-off the utilities between users and cloud service provider. Zhang et al. [22] investigate the time-varying cloud resource allocation problem. Overall, the above work and other non-mentioned auction mechanisms [31], [32], [33] cannot be applied to price UDL jobs as the job training time is uncertain and highly depends on the job placement.

### III. SYSTEM MODEL

#### A. System Overview

**Auction Overview:** Fig. 1 depicts an auction scenario which involves two types of entities: the service operator (auctioneer) and the users (bidders). The operator owns the edge network and the training data, and receives bids from users to train unbiased distributed learning (UDL) jobs in its edge system. Once it received the bids submitted by users, the operator conducts a schedule policy for jobs, and calculates the required payment. Then, the operator allocates resources for the accepted job, and charges the user accordingly. Assume that \( I \) jobs arrive and request for training within \( T \) time slots. Note that \( X \) indicates the integer set \([1, 2, \ldots, X]\). Table I lists some important notations.

**Service Operator:** The service operator serves as the owner and manager of resources in the edge distributed learning system. The system involves \( R \) geo-distributed “sites”, e.g., an edge cloudlet or edge server [29]. Each site collects and provides training data for UDL jobs, which are targeted at different smart applications, such as, intelligent video surveillance and smart city. In addition, each site \( r \) can offer \( C_r^k \) units of type-\( k \) resource and connects with other sites by edge networks. Let \( K \) denote the number of resource types, e.g., GPU, CPU and disk storage. Since radio resources in the edge network are scarce and expensive, as a result, the operator imposes a limit of the total data rate for transmission, denoted by \( B \).

When a new job arrives, the service provider has to make multiple decisions, such as: i) whether accept this job; ii) where the PS of this job should be placed; iii) the allocated number of workers for this job; iv) how to transmit the training data among edge sites for this job.

**Users:** Users submit their bids for training jobs once they arrive. For ease of analysis, suppose one user submit one job at a time. Thus, we do not distinguish a user and a job below. User \( i \) submits bid information, which is a tuple \( \Pi_i \)

\[
\Pi_i = \{a_i, f_i(\cdot), \{D_r^i\}_{r \in R}, E_i, \{w_r^i, s_r^i\}_{r \in K}\},
\]

where \( a_i \) indicates the arrival time of job \( i \), \( f(\cdot) \) denotes the specific utility function of job \( i \), which is non-negative and non-increasing with its execution duration \((t_j - a_i)\) [34], [35].

Generally, one UDL job’s utility is customized by its user according to the latency-sensitive requirement (i.e., completion time) and the characteristics of its workload. To clarify, assuming that data chunk is set to be the unit of data. For job \( i \), \( D_r^i \) represents the required data size (the number of data chunks) in site \( r \). For good performance, all data needs to be trained for \( E_i \) epochs. \( w_r^i (s_r^i) \) represents the amount of type-\( k \)-resource occupied by one worker (PS) of job \( i \), \( \forall k \in K \). Let \( P_i \) indicate the processing capacity of job \( i \)’s worker, i.e., the number of data chunks job \( i \)’s worker can process in one time slot.

**Parameter Server Architecture:** In order to reduce bandwidth consumption, we adopt the data parallel training and the PS architecture [36], rather than requiring all training data across

---

**TABLE I LIST OF NOTATIONS**

<table>
<thead>
<tr>
<th>( I )</th>
<th># of jobs</th>
<th>( T )</th>
<th># of time slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>job ( i )’s arrival time</td>
<td>( t_i )</td>
<td>completion time of job ( i )</td>
</tr>
<tr>
<td>( x_i )</td>
<td>accept job ( i ) or not</td>
<td>( f_i(\cdot) )</td>
<td>job ( i )’s utility function</td>
</tr>
<tr>
<td>( P_i )</td>
<td>processing capacity of job ( i )’s worker</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( D_r^i )</td>
<td>the size of the training data for job ( i ) in site ( r )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( B )</td>
<td>overall volume (data rate) of the amount of data transmission per time slot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_i(\cdot) )</td>
<td>whether job ( i )’s PS is placed in site ( r ) at ( t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g_i^r(\cdot) )</td>
<td># of job ( i )’s workers in site ( r ) at ( t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_r^k )</td>
<td>the capacity of type-( k )-resource in site ( r )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_r^i(\cdot) )</td>
<td>the size (MB) of the parameter exchanged between a worker and the PS of job ( i ) per time slot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_r^{i(t)} )</td>
<td>the size of the training data of job ( i ) transmitted from site ( r ) to site ( r' ) at ( t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \theta )</td>
<td>the size (MB) of one data chunk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( w_r^i(\cdot) )</td>
<td>the amount of type-( k )-resource occupied by one worker (PS) of job ( i )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \psi_r^i(\cdot) )</td>
<td>the amount of data transmitted between site ( r ) and ( r' ) at ( t ) in job ( i )’s schedule</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \nu_r^i(\cdot) )</td>
<td>the size of the training data transmitted at time ( t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_r^i(\cdot) )</td>
<td>the amount of allocated type-( k )-resource of site ( r ) at ( t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_r^i(\cdot) )</td>
<td>unit price for data transmission (MB) at time ( t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \gamma_k(\cdot) )</td>
<td>the base of the data transmission (resources) price functions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
geo-distributed sites to train centrally [37]. Workers and PSs are configured as virtual machines (VMs) or containers. Assuming that each UDL job is allocated with one PS, which in practice can represent various PS instances within one site. We use synchronous training to guarantee the model convergence [38] due to the fact that synchronous training can obtain a model with comparatively higher level of accuracy compared to asynchronous training [39]. Typically, one time slot will be longer than a training epoch in length, so one can perform training at each time slot. One time slot, for instance, will be for 30 minutes or more. To meet the demand for training data, at beginning of each time slot, one needs to redistribute training data (workload) among sites for processing at the current time slot. For adopting Mini-batch Gradient Descent [40], each data chunk will be split up into some equal-sized mini-batches. Iteratively, workers calculate and transmit model gradients (directions of changes) to the PS. The PS updates the model after received all workers' model parameters, and then sends back to all workers. Once all mini-batches have been processed for the predefined number of epochs, workers will further compute gradients using the subsequent mini-batch. Let \( \lambda_i \) denote the total size (MB) of the parameter transmitted between a worker and the PS per slot.

### Auction Preliminary

Here, to make everyone happy in the long run, we focus on the whole system’s overall utilities, that is, aiming to achieve social welfare maximization. To achieve this, eliciting truthful information from users is extremely necessary. In practice, due to the egocentric characteristic of users, they will attempt to maximize their own payoffs. More specifically, to obtain a higher payoff, users may try to lie about the real valuation, i.e., for all

\[
\text{Definition 1(Truthful Auction):} \quad \text{An auction is truthful if and only if every user’s payoff is maximized by reporting its true valuation, i.e., for all } \lambda_i \neq v_i(\cdot), u_i(v_i(\cdot)) \geq u_i(\lambda_i(\cdot)).
\]

### Social Welfare

- **Definition 2(Social Welfare):** Note that the payoff of job \( i \) is

\[
\text{Definition 2(Social Welfare):} \quad u_i(f_i(\cdot)) = v_i(\cdot) - p_i, \text{ if its bid is accepted, and } 0 \text{ otherwise.}
\]

The operator’s revenue is the total payment of all accepted jobs, i.e., \( \sum_{i \in \mathcal{I}} p_i x_i \). Here, the social welfare of the auction is defined as the aggregate utilities of the operator and users, and equals \( \sum_{i \in \mathcal{I}} (v_i(\cdot) - p_i) x_i + \sum_{i \in \mathcal{I}} p_i x_i \), i.e., \( \sum_{i \in \mathcal{I}} v_i(\cdot) x_i \).

### Problem Formulation

To clarify, we define the decisions made by the operator as:

- i) \( x_i \), a binary variable which means whether job \( i \)'s bid is accepted \( (x_i = 1) \) or not \( (x_i = 0) \); ii) \( z_i(t) \), a binary variable which indicates whether job \( i \)'s PS is placed in site \( r \) at time slot \( t \) or not; iii) \( g_i^r(t) \in \mathbb{N}^+ \), the allocated number of workers for job \( i \) in site \( r \) at time \( t \), \( \forall r \in \mathcal{R} \); iv) \( h_i^{r'}(t) \in \mathbb{N}^+ \), the number of data chunks for job \( i \) transmitted from site \( r \) to \( r' \) at time \( t \); v) \( p_i \geq 0 \), the required payment for user \( i \) once the operator accepts his bid \( (x_i = 1) \).

The social welfare maximization problem can be expressed as follows under truthful bidding \( (v_i(\cdot) = f_i(\cdot)) \):

\[
\begin{align*}
\text{maximize} & \quad \sum_{i \in \mathcal{I}} x_i f_i(\hat{t}_i - a_i), \\
\text{subject to} & \quad \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} \sum_{r' \in \mathcal{R}} h_i^{r'}(t) = \sum_{r \in \mathcal{R}} D_i x_i, \quad \forall i \in \mathcal{I}, \\
& \quad P_i g_i^r(t) \geq E_i \sum_{r' \in \mathcal{R}} h_i^{r'}(t), \quad \forall i \in \mathcal{I}, r \in \mathcal{R}, \forall t \in \mathcal{T}, \\
& \quad \sum_{r \in \mathcal{R}} z_i^r(t) = x_i, \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T}, \\
& \quad \sum_{v \in \mathcal{V}} (\theta h_i^{r'}(t) + \lambda_i g_i^r(t) [z_i^r(t) = 0]) \leq B, \forall t \in \mathcal{T}, \\
& \quad \sum_{v \in \mathcal{V}} (w_k^b g_i^r(t) + s_k^b z_i^r(t)) \leq C_k^b, \quad \forall k \in \mathcal{K}, r \in \mathcal{R}, \forall t \in \mathcal{T}, \\
& \quad \hat{t}_i = \arg \max_{t \in \mathcal{T}} \left( \sum_{r \in \mathcal{R}} g_i^r(t) > 0 \right), \quad \forall i \in \mathcal{I}, \\
& \quad g_i^r(t) = h_i^{r'}(t) = z_i^r(t) = 0, \quad \forall t : t < a_i, \\
& \quad g_i^r(t), h_i^{r'}(t) \in \{0, 1, 2, \ldots\}, \quad \forall i \in \mathcal{I}, r \in \mathcal{R}, t \in \mathcal{T}, \\
& \quad x_i, z_i^r(t) \in \{0, 1\}, \quad \forall i \in \mathcal{I}, r \in \mathcal{R}, \forall t \in \mathcal{T}.
\end{align*}
\]

Constraint (2a) ensures that the amount of transmitted data equals the amount of job \( i \)'s training data. Specially, \( h_i^{r'}(t) \) denotes the local training data in site \( r \) at time \( t \), which means it does not need to be transmitted at all. Constraint (2b) guarantees that job \( i \)'s workers deployed in site \( r' \) at time \( t \) is sufficient for handling \( E_i \) epochs of data in site \( r' \). Constraint (2c) indicates that each job has one PS if accepted. Constraint (2d) illustrates that the amount of training data transmitted and the amount of parameter transmitted between sites limited by the data rate \( B \). Given the edge network’s inherent dynamic conditions like interference and fading, our constraint (2e) focuses on data volume, providing a more stable metric than uncontrollable traditional bandwidth. Utilizing data volume allows us to indirectly constrain on associated physical resources, such as bandwidth or transmission power, ensuring that the system operates within feasible and realistic limits. Here, we consider that there is no

\^Specially, decision variable \( p_i \) does not appear in problem (2) because of the definition of social welfare (Definition 2).
need to transmit parameters if the PS and workers are deployed in the same site. We specify this case by using \( I(z_i^t(t) = 0) \), where \( I(\cdot) \) is an indicator function. The resource constraint of sites for deploying workers and PSs is demonstrated by (2e).

The completion time of one job is defined in constraint (2f), that is, defining the completion time of a job as the largest time slot that this job still need workers to train.

**Challenges:** Problem (2) is a mixed integer nonlinear programming (MINLP), which is NP-hard [41, 42]. This statement can be easily supported by the following reformulated problem (3), which can be transformed from a classical Knapsack problem in a polynomial time. Furthermore, the problem (2) involves integer variables which are dependent on each other, and a non-conventional constraint (2d). In this regard, there is a need to make online decisions for deploying workers and PSs, and transmitting data, without any future information.

IV. Algorithm Design

In this section, we develop an auction-based scheduling framework, Eris, to schedule UDL jobs. Eris aims to achieve social welfare maximization. The following is the main idea.

**i.** In Section IV-A, we reformulate problem (2) into an integer linear program (ILP) by introducing exponential number of variables. Then, we construct its dual problem to handle the primal variables. By applying the theory of complementary slackness, we convert our goal to compute each job’s payoff, which means jobs with positive payoff will be accepted, otherwise rejected. The computation of jobs’ payoff is based on resource pricing and its corresponding assigned resources.

**ii.** In Section IV-B, we analyze the traditional pricing mechanisms in the market. Then we design various tailored price functions for resources based on market rules.

**iii.** In Section IV-C, we discuss how to deploy workers and the PS for one job to reduce resource consumption. Eris further decomposes problem (7) to a series of one-shot problems (8). A dynamic programming (DP) algorithm is applied to divide the total training data into time slots within the range \([a_i, t_i]\). For each one-shot problem, Eris conducts data transmission according to a tailored metric. Then Eris deploys workers in a greedy fashion according to the resource cost of one worker. Finally, the PS is placed to the site which has the maximum number of workers to reduce the communication cost.

A. Problem Reformulation and Auction Framework

Recall that searching for a feasible schedule for each job is the ultimate goal of the service operator. Therefore, to address problem (2), we consider reformulating it into the following 0-1 integer linear program (ILP) by encoding three types of integer variables \( z_i^t(t) \), \( g_i^t(t) \) and \( h_i^r(t) \) into one schedule variable \( t \).

\[
\text{maximize } \sum_{i \in I} \sum_{t \in L_i} x_{i,t} f_i(t - a_i),
\]

subject to

\[
\sum_{i \in I} \sum_{t \in L_i} \psi_i(t) x_{i,t} \leq B, \quad \forall t \in T,
\]

\[
\sum_{i \in I} \sum_{t \in L_i} \varphi_i^k(t) x_{i,t} \leq c_i^k, \quad \forall k \in K, \forall r \in R, \forall t \in T,
\]

\[
\sum_{i \in I} x_{i,t} \leq 1, \quad \forall i \in I,
\]

\[
x_{i,t} \in \{0, 1\}, \quad \forall i \in L_i, \forall i \in I.
\]

In ILP (3), schedule \( \ell \) represents one deployment result for job \( i \), which consists of integer variables \( z_i^t(t) \), \( g_i^t(t) \) and \( h_i^r(t) \). \( L_i \) denotes the set containing job \( i \)'s all feasible schedules that satisfies constraint (2a)–(2c) and (2f). For consistency, binary variable \( x_{i,t} \) is reshaped to \( x_{i,t} \), which denotes whether select job \( i \)'s schedule \( \ell \) or not. And \( \hat{t}_i \) represents the completion time of job \( i \) according to schedule \( \ell \). Likewise, \( \psi_i(t) \) represents the amount of data transmitted between site \( r \) and \( r' \) at time \( t \) in job \( i \)'s schedule \( \ell \), i.e., \( \psi_i(t) = \sum_{r,r' \in R} h_i^r(t) + h_i^{r'}(t) (I(z_i^r(t) = 0)) \). \( \varphi_i^k(t) \) indicates the amount of type-\( k \) resource in site \( r \) currently occupied by job \( i \) according to its schedule \( \ell \), i.e., \( \varphi_i^k(t) = \sum_{r \in R} g_i^r(t) + \sum_{r' \in R} h_i^{r'}(t) \). In addition, variables \( z_i^r(t) \), \( g_i^r(t) \) and \( h_i^r(t) \) represent the corresponding value within schedule \( \ell \), respectively. Constraint (3a) and (3b) are equivalent to constraint (2d) and (2e), respectively. Then constraint (3c) and (3d) are corresponding to constraints (2a)–(2c). Thus, ILP (3) is equivalent to problem (2).

**Dual Problem:** Obviously, the number of feasible schedules of each job is potentially exponential because of combinatorial nature of these variables. To tackle the exponential scale of introduced variables, we construct its dual problem (4) by relaxing the binary variable \( x_{i,t} \) to \( x_{i,t} \geq 0 \). Dual variables \( \alpha(t), \beta_i^k(t) \) and \( \mu_i \) are associated with constraint (3a), (3b) and (3c), respectively.

\[
\text{minimize } \sum_{i \in I} \mu_i + \sum_{t \in T} B \alpha(t) + \sum_{t \in T} \sum_{r \in R} \sum_{k \in K} C_r^i \beta_i^k(t),
\]

subject to

\[
\mu_i \geq f_i(\hat{t}_i - a_i) - \sum_{t \in T} \psi_i(t) \alpha(t) - \sum_{t \in T} \sum_{r \in R} \sum_{k \in K} \varphi_i^k(t) \beta_i^k(t),
\]

\[
\forall i \in I, \forall t \in L_i, \quad \forall i \in I, \forall r \in R, \forall t \in T.
\]

**Design Rationale:** By interpreting the dual variable \( \alpha(t) \) as the unit price for data transmission (MB) at time \( t \), \( \sum_{i \in T} \psi_i(t) \alpha(t) \) indicates the data transmission cost of job \( i \). Similarly, \( \sum_{t \in T} \sum_{r \in R} \sum_{k \in K} \varphi_i^k(t) \beta_i^k(t) \) represents the total cost of resources occupied by job \( i \) with schedule \( \ell \), where \( \beta_i^k(t) \) recognizes as the unit price of the type-\( k \) resource of site \( r \) at time \( t \). So the right hand side (RHS) of (4a) means job \( i \)'s utility minus the overall cost. In this regard, \( \mu_i \) represents the received payoff of job \( i \) if it is accepted. To minimize the objective of the dual problem, we set the dual variable \( \mu_i \) to be the maximum value between 0 and the RHS of (4a) based on the optimal
Algorithm 1: Auction Framework Eris.

Input: $T, C^h_i, B, \forall k \in K, \forall r' \in R$

Output: $x_i, g_i(t), z^*_i(t), h_i^r(t), \forall i \in I, \forall r, r' \in R, \forall t \in T$

1: Initialize $g_i^0(t) = 0, z^*_i(t) = 0, h_i^r(t) = 0, \rho(t) = 0, \eta^h_i(t) = 0, \alpha(t) = \alpha(0), \beta^h_i(t) = \beta^h_i(0), \forall i, \forall r, r', \forall k, \forall t$

2: for $i = 1, 2, \ldots, I$ do //new jobs
3: \quad $\text{Set } (\mu_i, \mu_r) = \text{SA}(a_i, E_i, \{D^l_i\}_{r}, \rho(t), t, \eta^h_i(t))$
4: \quad if $\mu_i > 0$ then
5: \quad \quad Update $\rho(t) = \psi(t), \eta^h_i(t) = \eta^h_i(t) + \phi^h_i(t), \alpha(t) = \alpha(t) + \beta^h_i(t)$
6: \quad \quad $\forall i, \forall r, \forall k, \forall t$
7: \quad \quad Set $x_i = 1$ and deploy job $i$ according to schedule $l^*$
8: \quad \quad else
9: \quad \quad Set $x_i = 0$ and reject job $i$
10: \quad end if
11: end for

schedule $l^*$. $\mu = \max\{0, \max_{i \in I} \text{RHS of (4a)}\}$. (5)

If $\mu_i > 0$, the operator accepts and allocates resources for job $i$ according to its schedule $l^*$ ($x_i = 1$); otherwise, the operator rejects ($x_i = 0$, $\forall i \in \mathcal{I}$). The explanation is that jobs' value can cover the overall cost, which ensures the jobs' owners are willing to train their jobs. Besides, the operator attempts to accept those jobs with larger value and less resource consumption.

Auction Framework: The main workflow of Eris is presented in Algorithm 1. The subroutine Algorithm 2 computes the optimal schedule $l^*$ for each newly arrived job (Line 3). If the value of $\mu$ is larger than $0$, the operator accepts the job ($x_i = 1$) and updates the current price of resources or data by using well-designed price functions (lines 4–7), discussed in Section IV-B.

B. Price Function Design

Before finding the optimal schedule for jobs, let us first discuss the marginal price of data and resources, i.e., variables $\alpha(t)$ and $\beta^h_i(t)$. In market economics, resource price always increases with the amount of resources occupied. To be specific, when there are enough resources available, resource prices should be decreased to encourage consumption. When demand for a resource exceeds supply, service providers will inevitably raise the price in order to relieve the strain on supply. In this paper, we adopt an exponential function as the basic expression to depict this market rule. Here, let $\rho(t)$ denote the amount of data transmitted at time $t$, i.e., $\sum_{i \in I_t} \sum_{r, r' \in R} \theta_i^r r^{r'}(t) + \lambda_i g_i^r(t) \{z^*_i(t) = 0\}$. And $I_t$ indicates a set of accepted jobs before job $i$ arrived. Additionally, $\eta^h_i(t)$ represents the amount of allocated type-$k$ resource of site $r$ at time $t$, i.e., $\sum_{i \in I_t} \sum_{r \in R} w_i^r g_i^r(t) + s_i^k z^*_i(t)$. Therefore, the price functions of data transmission/resources can be formulated as

$$\alpha(\rho(t)) = (\gamma_d \frac{\rho(t)}{\rho_i} - 1, \beta^h_i(\eta^h_i(t)) = (\gamma_k \frac{\eta^h_i(t)}{\eta_i} - 1). \quad (6)$$

Here, let us discuss several characteristics of our tailored price functions. If there are no resources used (i.e., $\rho(t) = 0$) or no transmitted data (i.e., $\eta^h_i(t) = 0$), then $\alpha(t) = \beta^h_i(t) = 0, \forall r, \forall k, \forall t$. In this case, all jobs would be accepted, to incentive more people to submit their jobs in this auction, and increase the possibility of accepting jobs with larger utility for the operator. If there are no available resources or no data budget left, $\alpha(t) = \gamma_d - 1, \beta^h_i(t) = \gamma_k - 1, \forall r, \forall k, \forall t$. This means that these two values should be large enough to avoid any jobs from being accepted by the operator, i.e., $f_i(t_i - a_i) = \sum_{t \in \mathcal{T}} (\gamma_d - 1) \psi(t_i) = \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{K}} (\gamma_d - 1) \varphi_i^{r,k}(t_i) < 0$. A feasible solution is to set $\sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{K}} \varphi_i^{r,k}(t_i) \leq \gamma_d - 1$, by assuming that $\sum_{i \in \mathcal{K}} \sum_{t \in \mathcal{T}} \varphi_i^{r,k}(t_i) \geq 1, \forall i$. It is reasonable because there are no sufficient resources to handle more jobs. Moreover, such assumptions can prevent users from submitting jobs and enable the operator to escape peak load quickly.

C. Sub-problem of Finding Schedule $l$

The sub-problem of finding an optimal schedule $l^*$ for each job $i$ can be expressed as follows.

$$\max_{i, t, g, h} f_i(t_i - a_i) - \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} \sum_{k \in \mathcal{K}} (\lambda_i g_i^r(t) + s_i^k z^*_i(t)) \beta^h_i(t)$$

$$- \sum_{t \in \mathcal{T}} \sum_{r, r' \in R} (\theta_i^r r^{r'}(t) + \lambda_i g_i^r(t) \{z^*_i(t) = 0\}) \alpha(t)$$

subject to

$$\psi(t_i) \leq B - \rho(t), \forall t \in \mathcal{T}, \quad (7a)$$

$$\varphi_i^{r,k}(t) \leq C^h_i - \eta^h_i(t), \forall r \in \mathcal{R}, \forall k \in \mathcal{K}, \forall t \in \mathcal{T}, \quad (7b)$$

Constraint (7a) and constraint (7b) are equivalent to constraint (3a) and (3b), respectively. To solve problem (7), we consider fixing and enumerating the completion time $t_i$ to simplify problem (7). Then we further apply DP to divide the total workload $D_i$ (i.e., $\sum_{r \in \mathcal{R}} D_i^r$) into $D_i(t)$, $\forall t \in [a_i, t_i]$. Consequently, problem (7) can be simplified as a series of following one-shot problems.

$$\min_{i, t, g, h} \text{cost}(t, D_i(t)) = \sum_{r, r' \in \mathcal{R}} (\theta_i^r r^{r'}(t) + \lambda_i g_i^r(t) \{z^*_i(t) = 0\}) \alpha(t)$$

$$+ \sum_{r \in \mathcal{R}} \sum_{k \in \mathcal{K}} (w_i^r g_i^r(t) + s_i^k z^*_i(t)) \beta^h_i(t), \quad (8)$$

subject to

$$(7a) - (7b), (2b) - (2c), (2g) - (2i), \text{where } x_i = 1. \quad (7b)$$

Algorithm Details: The details of simplifying and computing optimal schedule $l^*$ is given in Algorithm 2. In lines 2–7,
Algorithm 2: Scheduling Algorithm SA.

Input: $a_i, E_i, D_i, r_i, \eta_i(t), \forall k \in K, \forall r \in R, \forall t \in T$
Output: $l^*, \mu_i$
1: Initialize $l^* = \emptyset$, $\mu_i = 0$
2: for $l^* = a_i$ to $T$ do
3: \hspace{1em} $(\text{cost}^*, l) = \text{DPC}(l^*, \sum_{r \in R} D_i^r)$
4: \hspace{1em} if $\mu_i < f_i(l^* - a_i) - \text{cost}^*$ then
5: \hspace{2em} $\mu_i = f_i(l^* - a_i) - \text{cost}^*$, $l^* = l$
6: \hspace{1em} end if
7: end for
8: Return $l^*, \mu_i$
9: function $\text{DPC}_{T_i}, D_i$
10: \hspace{1em} $\text{cost}_d = \infty$, $l = \emptyset$
11: for $d = 0$ to $D_i$ do
12: \hspace{1em} $(\text{cost}_d, \{g^i_r(T_i)\})_r \triangleright \{z^i_r(T_i)\}_r, \{h^{rr}_i(T_i)\}_r = \text{DPC}_{T_i}(l, d)$
13: \hspace{1em} $(\text{cost}_d, l) = \text{DPC}(T_i - 1, D_i - d)$
14: \hspace{1em} if $\text{cost}_d > \text{cost}_l + \text{cost}$ then
15: \hspace{2em} $\text{cost}_l = \text{cost}_d$, $l = l \cup \{\{g^i_r(T_i)\}_r, \{z^i_r(T_i)\}_r, \{h^{rr}_i(T_i)\}_r\}$
16: \hspace{1em} end if
17: end for
18: Return cost, $l$
19: end Function

for new arrived job $i$. Algorithm 2 enumerates its $l^*$, within the range $[a_i, T]$. In line 5, Algorithm 2 obtains the optimal schedule for each job with the fixed $l^*$ by function DPC. Through comparing $\text{cost}_l$, at different $l^*$, achieved by DPC, the optimal schedule $l^*$ can be returned (lines 4–6). Lines 9–18 achieve a DP function: $\text{DPC}(l^*, D_i) = \min_{d \in [0, D_i]} \text{DPC}_{T_i}(l^*, d) + \text{DPC}(l^* - 1, D_i - d)$. The workload to be finished within time slot $l^*$ will be enumerated from 0 to $D_i$ to narrow the scheduling scale (lines 11–12). The remaining workload $D_i - l^* - 1$ will be handled within $[a_i, l^* - 1]$ (line 13). Lines 14–16 attempt to identify the corresponding schedule with the smallest cost. Similarly, obtaining the corresponding optimal schedule at a narrower time window $[a_i, l^* - 1]$ can be addressed. For time slot $t$ within $[a_i, l^* - 1]$, we will enumerate the workload and further narrow the scheduling scale recursively until reach the arriving time $a_i$. The deployment algorithm is discussed in Section IV-D.

### D. Deployment Algorithm

The key to solve problem (8) is to decide where to deploy workers and the PS, and to determine data transmission to minimize the overall cost. This can be addressed optimally by a simple greedy algorithm, which is discussed in Algorithm 3. And its optimality will be shown in Theorem 4.

Here, we consider redistributing the training data across sites to accommodate workers and the PS’s deployment. Algorithm 3 makes decisions for deploying workers and PSs at $t$ to fulfill workload $D_i(t)$, and calculates the required data transmission. The data redistribution is based on a tailored metric, which is defined as below.

$$\Omega_{rr}(t) = \theta \alpha(t) - \frac{E_i}{\beta_k} \sum_{k \in K} w^k_i (\beta_k^r(t) - \beta^k_r(t)), \quad (9)$$

where the first term denotes the cost of transmitting one data chunk, and the second term represents the reduced resource cost of one data chunk if we transmit it from site $r$ to $r'$. The rationale is that we aim to transmit data to sites with lower resource cost to train if the reduced resource cost can cover the cost introduced by data transmission. Note that we will stop transmitting data if the value of $\Omega_{rr}(t)$ is smaller than 0, i.e., the cost of transmitting one data chunk is larger than the corresponding reduced resource cost. Therefore, we first sort pairs of sites (i.e., $(r, r')$) in non-decreasing order according to the metric $\Omega_{rr}(t)$ in line 2. Then we enumerate each pair to determine whether the operator needs to transmit data between sites or not (lines 2–11). In particular, line 5 calculates $h^{rr}_i(t)$ by using the following equation.

$$h^{rr}_i(t) = \min \{M^{rr}_i(t) - \sum_{r' \in R} h^{rr}_i(t), \left[\frac{B - \rho(t)}{\theta}\right]\},$$

$$D_i(t) = \sum_{j=1}^{j-1} (M^{rr}_i(t) + \sum_{r \in R} h^{rr}_i(t)),$$

\[\begin{align*}
\frac{P_i}{E_i} \min_{k \in K} & \left[\frac{C^k_v - h^k_i(t)}{w^k_i}\right] - M^{rr}_i(t) - \sum_{r \in R} h^{rr}_i(t),
\end{align*}\]

where the first term represents the remaining amount of job $i$’s data in site $r_j$. The second term guarantees constraint (7a). The third term indicates the remaining workload needed to be processed currently. In addition, the last term represents that the number of data chunks that site $r_j$ can still handle currently except its current training data volumes. In lines 6–8, we decide whether transmitting data from site $r_j$ to $r_j'$ is feasible through the value of $h^{rr}_i(t)$. We reset $h^{rr}_i(t)$ to 0 and exclude all pairs includes site $r_j$ if transmitting data is unnecessary, i.e., $h^{rr}_i(t) < 0$.

After finishing the data re-distribution, we consider deploying workers via a greedy fashion according to the unit price of one worker of job $i$, i.e., $\sum_{k \in K} w^k_i (\beta_k^r(t))$. Lines 13–15 compute the corresponding number of workers for each site. Here, we reserve resources for the PS in each site to reduce the complexity. This is reasonable because the resources occupied by workers and PS are essentially different. Workers need GPU to train [11, however PSs aggregate parameters using CPU. So reserving resources for the PS would not affect the deployment of workers. To reduce the cost of parameter exchanged, we greedily deploy the PS on the site, which has the maximum number of workers (line 16). If not enough workers or violating constraint (7a), fulfilling workload $D_i(t)$ at time $t$ is infeasible (lines 17–19); otherwise, we return the overall cost and corresponding schedule.
Algorithm 3: Deployment Algorithm DPC_T.

**Input:** \( t, D_i(t) \);

**Output:** \( \text{cost}_t, g^k_i(t), z^k_i(t), h^\ast_{r'}(t), \forall r', r' \in R \)

1: Initialize \( g^k_i(t) = 0, z^k_i(t) = 0, h^\ast_{r'}(t) = 0, \forall r, r' \in R \)

2: Sort all pairs of sites according to \( \Omega_{r'}(t) \) in non-decreasing order into \( \{r, r'\}^1, \{r, r'\}^2, \ldots \).

3: for \( j = 1 \) to \( \frac{R(R-1)}{2} \) do

4: if \( \Omega_{r'}(t) < 0 \) then \# whether transmit data or not \#\$

5: Compute \( h^\ast_{r', r'}(t) \) using (10)

6: if \( h^\ast_{r', r'}(t) < 0 \) then \# exceeding resource limits \#\$

7: \# end if

8: Exclude all pairs \( \{r, r'\} \), that include site \( r' \)

9: end if

10: end for

11: end for

12: Sort in \( R \) according to \( \sum_{k \in K} w^k_i \beta^k_i(t) \) in non-decreasing order into \( r_1, r_2, \ldots, r_R \)

13: for site \( j = 1 \) to \( R \) do \# deploy workers \#\$

14: \( g^k_i(t) = \min \left( \left\{ \frac{g^k_i(t) + \sum_{r \in R} h^\ast_{r'}(t)}{w^k_i(t)} \right\} \right) \)

15: end for

16: \# end for

**V. THEORETICAL ANALYSIS**

In Section V, we verify several key properties of Eris including the correctness, individual rationality, truthfulness, competitive ratio and time complexity.

**Theorem 1:** Our proposed auction Eris produces a feasible solution to the original sub-problem (2).

**Proof:** We first show that a feasible solution of sub-problem (8) can be obtained by using Algorithm 3. For each pair, we calculate its corresponding transmitted data by using (10), which can satisfy constraint (7a), (2d), and (2h). Line 4 of Algorithm 3 computes \( g^k_i(t) \) according to constraint (7a), (7b), (2h), and (2e). And the placement of the PS also satisfies constraint (2e). So Algorithm 3 produces a feasible solution of sub-problem (8). Function DPC can guarantee constraint (2a) will not be violated. In lines 2–7 of Algorithm 2, the for loop enumerates \( i \), which can satisfy constraint (2f). Therefore, we complete the proof. □

**Theorem 2 (Individual rationality):** Eris can achieve individual rationality.

**Proof:** To prove this, we need to ensure that the utilities of users and the profit of the service provider are both non-negative. For users, this requirement can be easily satisfied because (5) ensures that if one user’ utility \( \mu_i > 0 \), the operator accepts and allocates resources for job \( i \) according to its schedule \( l' \) ( \( x_{il'} = 1 \)); otherwise, the operator rejects ( \( x_{il'} = 0, \forall l \in L_i \)). For the service provider, its profit equals to the total payment of all accepted jobs, i.e., \( \sum_{i \in j} p_i x_i \), which always be positive. Therefore, we finish this proof. □

**Theorem 3 (Truthfulness):** Eris is a truthful auction.

**Proof:** This theorem can be verified through analyzing the truthfulness of all elements in the bid tuple. □

**Theorem 4:** In the arrival time \( a_i \): To start with, we discuss that user \( i \) has no motivation to report an earlier arrival time of the job. In fact, user \( i \)'s job cannot arrive at the reported/earlier time. In this case, the service operator can directly discard this job when this happened. Besides, additional punishment mechanisms can be applied to avoid this. If user \( i \) bids with a later arrival time, it does not actually affect the service operator to schedule jobs as expected. Intuitively, users also have no motivation to do so because it would prolong the completion time of jobs. Therefore, we finish this part of proof.

**Truthfulness in utility functions \( f_i(\cdot) \):** Recall that Eris is based on the posted pricing mechanisms [43]. For each accepted job \( i \), its payment is computed as \( p_i = \sum_{t \in T} \sum_{r \in R} \alpha(t) + \sum_{t \in T} \sum_{r \in R} \lambda_i g_i^k(t) \alpha(t) + \sum_{t \in T} \sum_{r \in R} \sum_{k \in K} w^k_i g_i^k(t) + s^k_i z^k_i(t) \beta^k_i(t) \). We can see that the payment equation only reflects the total cost for training job \( i \), independent of its utility function \( f_i(\cdot) \). Moreover, according to Definition 1, the payoff equals its valuation minus the payment, which has no relationship with utility function \( f_i(\cdot) \). So jobs’ owners would not misreport their utility functions.

**Truthfulness in the required data size \( D_i \) and epochs \( E_i \):** Regardless of the required data sizes or epochs, as long as user \( i \) reports a larger one, it will cause its job to take longer time to compete. On the other hand, a smaller one will harm the interests of users because their ML models fail to achieve the predetermined performance, i.e., accuracy threshold. Thus, users have no motivation to misreport the required data size and epochs.

**Truthfulness in the resource configuration of one worker \( w^k_i \) and PS \( s^k_i \):** Similarly, users have no motivation to report the amount of any type of resources occupied by one worker or PS, since it will reduce the computation capacity of worker/PS so that prolong the completion time of jobs. Intuitively, if users can report the resource configuration of workers/PSs with more computing power, i.e., larger \( w^k_i \) or \( s^k_i \), their jobs can be finished more quickly. In fact, there is no need for users to do this because the more resources occupied also means that users have to pay more money to the operator. Thus, the truthfulness of the resource configuration of workers/PSs can be verified.

As a result, the proposed auction framework Eris can achieve truthfulness. □

**Theorem 4:** Algorithm 2 achieves an optimal solution of original problem (7), in which Algorithm 3 returns an optimal solution of problem (8).

**Proof:** Optimality of Algorithm 3: First, we try to analyze the optimality of Algorithm 3. Here we denote the schedule returned
by Algorithm 3 as $l_{i,d}$, which consists of data redistribution, and the deployment of workers and the PS. Note that $\sum_{r \in R} D_{r,i}(t)$ represents the minimum number of workers, which can handle workload $d$ at time $t$ for job $i$. Assume that there exists one another schedule $\hat{l}_{i,d}$, which receives a smaller cost than $l_{i,d}$. Likewise, the data transmission, and the deployment of workers and the PS are indicated by $\hat{h}_{i,r}^r(t), \forall r, r' \in R, \hat{g}_{i,i'}(t), \forall r \in R$ and $\hat{z}_{i,r}(t), \forall r \in R$, respectively. Note that the total cost consists of data transmission cost, the resource cost of workers and the PS, and parameter exchanged cost.

We first discuss the data transmission based on a metric $\Omega_{i,r}^r(t)$. Recall that the third term and the last term can ensure the amount of transmitted data is minimum from the analysis of (10). There are two cases of this data re-distribution if necessary.

1) If $\sum_{r, r' \in R} \hat{h}_{i,r}^r(t) = \sum_{r, r' \in R} h_{i,r}^{r'}(t)$, then in $\hat{l}_{i,d}$, there exists some $\hat{h}_{i,r}^r(t)$, which not equals $h_{i,r}^{r'}(t)$. Because the amount of transmitted data remains the same, $l_{i,d}$ can be converted into $\hat{l}_{i,d}$ by transmitting data from other sites. Based on the metric $\Omega_{i,r}^r(t)$, we greedily transmit data from the site with the highest resource cost to the one with the lowest cost. So transmitting data from other sites in $\hat{l}_{i,d}$ can only increase the data transmission cost.

2) If $\sum_{r, r' \in R} \hat{h}_{i,r}^r(t) > \sum_{r, r' \in R} h_{i,r}^{r'}(t)$, we need to transmit more data in the edge-network, which would cause to more data transmission cost. For the first case, we can know that moving data from other sites can increase the cost even if the amount of transmitted data is the same. So in this case, the transmission cost would definitely increase when meeting more transmitted data.

Similarly, there are also two cases for the deployment of workers in schedule $l_{i,d}$. Here, we omit some details since the optimality of this part can be proved because of the similar reason. As for the deployment of the PS, on the site with maximum number of workers can minimize the parameter exchanged cost. Consequently, we can conclude that $DPC_{t,T}$ returns an optimal schedule $l_{i,d}, \forall i \in [a_i, t_i], 0 \leq d \leq D_t$.

**Optimality of Function DPC:** For every completion time, we try to optimally dispatch the total workload into time slots within range $[a_i, t_i]$ by using dynamic programming. The optimality of $DPC$ can be proved by the induction on $t = a_i, \ldots, T$. At time $a_i$, we have $DPC(a_i, d) = DPC_{a_i}(a_i, d), \forall 0 \leq d \leq D_t$, which means $DPC(a_i, d), 0 \leq d \leq D_t$ can return the smallest cost when handling workload $d$. The induction hypothesis illustrates that $DPC(t, d)$ fulfills workload $d \in [0, D_t]$ with the minimum cost, for any time $t$. Furthermore, based on the DP function, we can get $DPC(t+1, d) = \min_{d' \in [0, d]} DPC_{t+1}(t+1, d'), DPC(t, d-1'), \forall 0 \leq d \leq D_t$. Therefore, $DPC(t+1, 1)$ also returns the minimum cost when fulfilling workload $d$ at $t+1$. Consequently, we can claim that $DPC(t, D_t)$ can obtain the schedule with the smallest cost at time $t_i \in [a_i, T]$, for any jobs.

**Optimality of Algorithm 2:** Recall that $\hat{t}_i$ is the testing deadline rather than the final completion time. The optimal $\hat{t}_i$ is tight, i.e., $\sum_{t \in T} [\hat{t}_i(t) = 0]$, after enumerating $\hat{f}_i$ from $a_i$ to $T$. Assume that job $i$’s final completion time is $\hat{t}_i$, corresponding to the optimal schedule $l^*$. So we can obtain the optimal schedule when $\hat{t}_i = \hat{t}_i$. In Algorithm 2, we enumerate $\hat{f}_i$ from $a_i$ to $T$ and update the optimal schedule until receiving one new schedule with smaller payoff. Recall that the utility function of jobs is non-increase with $\hat{t}_i - a_i$. In this case, Algorithm 2 would not update the optimal schedule even if there exists another schedule with the same cost and a larger deadline. So we can record the optimal schedule if we get it. Therefore, we finish this part of proof.

**Theorem 5:** The competitive ratio is the upper-bound ratio of the optimal objective value to the objective value of the solution found by Eris. For problem (2), Eris is $c$-competitive, where $c = 2 \log \left( \gamma_d \gamma_k \right) + 1 = \max_k \{ \gamma_k \}$.

**Proof:** Let OPT indicates problem (3)’s optimal objective value, which is equivalent to problem (2), $c$, and $\beta^*$ represent the objective value of problem (3) and (4) returned by Algorithm 1 for scheduling job $i$, respectively. In particular, $\Gamma_0$ and $\Delta_0$ denote their original objective values when no job arrives. So the final objective values of the two problems are indicated by $\Gamma_t$ and $\Lambda_t$. For clarity, let us introduce one lemma.

**Lemma 1:** When a constant $c$ satisfies $\Gamma_1 - \Gamma_{i-1} \geq \frac{1}{2} (\Lambda_1 - \Lambda_{i-1}), \forall i \in T$, and if $\Gamma_0 = \Lambda_0 = 0$, then Algorithm 1 is $c$-competitive in total social welfare.

**Proof:** Given $\Gamma_t$, we can get $\Gamma_t - \Gamma_0 = \sum_{i \in T} (\Gamma_i - \Gamma_{i-1})$. Likewise, we have $\Gamma_t - \Gamma_0 = \sum_{i \in T} (\Gamma_i - \Gamma_{i-1}) \geq \frac{1}{2} \sum_{i \in T} (\Lambda_i - \Lambda_{i-1}) = \frac{1}{2} (\Lambda_t - \Lambda_0)$. Moreover, we have $\beta_t \geq OPT \geq \Gamma_t$ according to the weak duality. So we have $\Gamma_t \geq \frac{1}{2} \Lambda_t \geq \frac{1}{2} OPT$ combining $\Gamma_0 = \Lambda_0 = 0$. Therefore, we finish this proof.

Next we discuss how to design a specific $c$ to make Algorithm 1 satisfy $\Gamma_1 - \Gamma_{i-1} \geq \frac{1}{2} (\Lambda_1 - \Lambda_{i-1}), \forall i$. There are two cases according to whether the job is accepted or not. The inequality holds since $\Gamma_1 - \Gamma_{i-1} = \Lambda_1 - \Lambda_{i-1} = 0$ when job $i$ is rejected. Next, we focus on the other case where job $i$ is accepted with its schedule $l_i$. Here, we introduce some new variables. Let $\alpha_i(t)$ indicate the unit price of data transmission at time $t$ after accepting job $i$. $\delta_{p,i}(t)$ means the unit price of the type-$k$ resource of site $r$ at $t$ after processing job $i$. We have

$$\alpha_i(t) - \alpha_i(t-1) = \frac{\gamma_d^{(i-1)(t)} \gamma_k^{(i-1)(t)}}{\gamma_d^{(i-1)(t)}} - \frac{\gamma_d^{(i-1)(t)}}{\gamma_d^{(i-1)(t)}} = \frac{(\gamma_d^{(i-1)(t)})^2}{(\gamma_d^{(i-1)(t)})^2} = \frac{(\gamma_d^{(i-1)(t)})^2 - 1}{\gamma_d^{(i-1)(t)}} - 1.$$}

Then the formula $\alpha_i(t) - \alpha_i(t-1) \leq (\alpha_i(t) + 1) \log \left( \frac{\gamma_d^{(i-1)(t)}}{\gamma_d^{(i-1)(t)}} \right)$ holds by using an inequality $2x - 1 \leq x$, for any $x \in [0, 1]$. Similarly, we can get $p_i(t) - p_i(t-1) \leq \left( \lambda_d^{(i-1)(t)} + 1 \right) \log \left( \gamma_d^{(i-1)(t)} \right)^{\frac{1}{\gamma_d^{(i-1)(t)}}}$. Since $x_d = 1$, according to Algorithm 1, constraint (4a) can be guaranteed. The increased value of problem (4)’s objective is

$$\Lambda_i - \Lambda_{i-1} = \mu_i + \sum_{r \in K} \sum_{k \in K} \sum_{t \in T} C^{(i)}_k \left( p_i(t) - p_i(t-1) \right)$$

$$\leq \sum_{t \in T} B\left( \alpha(t) - \alpha(t-1) \right)$$

$$\leq \sum_{t \in T} B\left( \alpha(t) - \alpha(t-1) \right)$$
A. Implementation

We apply Kubernetes 1.19 to implement a distributed ML system based on Apache MXNet. Fig. 2 illustrates the testbed's overall architecture. The implemented testbed consists of nine physical machines. Each machine is equipped with 12 CPU cores, 1 GeForce RTX 2060 GPU, 16 GB RAM, 500 GB HDDs and a dual-port 1GbE NIC. In particular, we implement another physical machine as the centralized scheduler (Master Node). A shared Hadoop Distributed File System (HDFS) across machines will store and transmit all training data and corresponding parameters. Note that data chunk's default size is 2 MB, i.e., $\theta = 2$ MB. In each time slot, we will checkpoint and store the checkpointed model parameters from HDFS.

VI. IMPLEMENTATION AND EVALUATION

In this section, we present testbed implementation in Section VI-A, and then describe testbed setup, simulator and three baselines in Section VI-B. The performance of Eris is presented in Section VI-C.

A. Implementation

We apply Kubernetes 1.19 to implement a distributed ML system based on Apache MXNet. Fig. 2 illustrates the testbed's overall architecture. The implemented testbed consists of nine physical machines. Each machine is equipped with 12 CPU cores, 1 GeForce RTX 2060 GPU, 16 GB RAM, 500 GB HDDs and a dual-port 1GbE NIC. In particular, we implement another physical machine as the centralized scheduler (Master Node). A shared Hadoop Distributed File System (HDFS) across machines will store and transmit all training data and corresponding parameters. Note that data chunk’s default size is 2 MB, i.e., $\theta = 2$ MB. In each time slot, we will checkpoint and store the checkpointed model parameters from HDFS.

**Testbed Setup:** Six types of DL jobs are presented in Table II. In addition, each worker or PS is jobs implemented on a Docker container. The resources of each worker/PS are randomly chosen within the integer set $[0,1]$ GPU, $[1,3]$ CPU cores and $[2,6]$ RAM. The number of epochs $E_i$ for jobs will be set to 20 or 30. In addition, we can obtain the processing capacity of workers ($P_i$) through pre-training. The time span of each experiment is 50 time slots, each is set to 20 minutes. In the experiments, we assume that there are 10 jobs, which will arrive randomly in the first 40 time slots. Typically, each job takes 2 to 4 hours to run. Other settings are consistent with the settings in the following simulator.

**B. Experimental Setup**

**Simulator:** We design a MATLAB simulator to evaluate Eris’ effectiveness at a larger scale setting by leveraging jobs’ traces from our testbed. By default, there are 50 (R) sites. And 100 (I)
jobs randomly arrive in 150 (T) time slots. We simulate all job events in simulator, including job arrival, completion time and corresponding resource occupation. To cater the large scale, we increase the resource configuration of each site. We set resource configuration of each site as following: 24 CPUs cores, 4 GPUs and 32 GB RAM. The overall budget of data transmission per site is set to 20 GB. Then we set the utility function as

$$w_{i,j} = \frac{\alpha_{i,j} + \lambda_i}{1 + \rho(t_i)}$$

where $\alpha_i$ is a latency-sensitive factor in [1,2]. Accordingly, we set the value of $\gamma_d$ and $\gamma_k$ to 1000, i.e., the price functions are set as $\eta(x) = 1000 \cdot \frac{\tau_a}{\tau_f} - 1$ and $\beta_t(r_i(t)) = 1000 \cdot \frac{\tau_i}{\tau_f} - 1$.

**Baselines:** To evaluate the superiority of Eris, we implement three following representative schedulers:

- **Tiresias [11]:** a preemptive scheduler attempts to minimize average Job completion time. Tiresias allocates resources for jobs based on the number of resources, (e.g., GPUs) and the multiplication of the job’s remaining workload.

- **AntMan [22]:** a cluster scheduler, which introduces two types of jobs: opportunistic job and resource-guarantee job. AntMan schedules resource-guarantee jobs first and allocates sufficient GPU resources to them. For opportunistic jobs, AntMan aims to utilize free resources to the best of its ability. Resource-guarantee jobs that suffer long queuing delay will be automatically executed as opportunistic jobs.

- **Liquid [23]:** a cluster network-efficient scheduler. Liquid tends to use the network communication cost and resource scheduling plans to obtain the best trade-off, that is, $\min \lambda \cdot cost_k + (1 - \lambda) \cdot \sum_j scheduling_plan_j$, where $scheduling_plan_j$ mainly indicates the number of workers used for the job.

Note that AntMan and Liquid will assign the fixed number of workers and PSs for each new arrival job according to the unit price of worker at job’s arrival time $\alpha_i$, i.e., $\sum_{k \in K} w_{i,k} \cdot \beta_t(a_i)$. Accordingly, to reduce data transmission consumption, the training data of each new arrival job will be re-distributed evenly and greedily to sites. For Tiresias, there exists an extreme case that job’s payoff is smaller than 0 due to frequent preemption. We adopt it by omitting those jobs with negative payoff.

**C. Evaluation Results**

1) **Eris in Testbed Experiments:** In the experiments, we depict the total social welfare of four algorithms and the optimal solution in Fig. 3, which indicates that Eris still get a good performance than three baselines. Note that the optimal solution is obtained by leveraging MATLAB optimization tool-box. Besides, we use a supplementary metric (the number of rejected jobs) to assess and validate the algorithm’s effectiveness. This metric provides a clear indication of the scheduling algorithms’ efficiency under limited resources, elucidating how Eris achieves nearly-optimal social welfare. The results in Fig. 4 show that under the same resource condition, Eris can allocate resources for ML jobs more optimally than baselines, leading to the acceptance of more jobs (and the refusal of fewer).

The performance ratio of Eris with baselines are presented in Fig. 5. The performance ratio is defined as the ratio of the social welfare of the offline optimal solution to the offline social welfare of the target algorithm. This ratio is greater than 1, and if the ratio is near (far) 1, it means the algorithm performs well (poor) when compared to the optimal. In Fig. 5, we can see that Eris achieves a small performance ratio (< 1.5), which coincides with the theoretical analysis in Theorem 5.

In Fig. 6, we show the accuracy of three respective datasets. Especially, we increase the number of epochs from 20 ~ 30 (default values) to 40 to illustrate the dynamic deployment of
workers would not affect the training process, and can still guarantee a good performance. Here, Fig. 7 demonstrates more details about the normalized resource utilization of the entire testbed in each time slot during Eris’s execution. Note that the normalized resource utilization represents resource utilization divided by overall testbed capacity. Especially, we depict three main resources consumed: GPU, CPU and RAM. From Fig. 7, one can see that CPU and RAM occupation are relatively stable since the price mechanism can avoid those extreme cases as much as possible. The normalized GPU utilization sometimes reaches up to 100% because of the insufficient number of GPUs.

2) Eris in Large-Scaled Simulations: We evaluate the scalability of Eris in the case of submitting numerous jobs in a cluster with hundreds of sites. Figs. 8 and 9 show the total social welfare under different I and R, respectively. From these two figures, one can observe that Eris outperforms three schedulers, especially in a large scale setting. Therefore, we can claim that our algorithm Eris always outperforms three baselines regardless of the input scale. Furthermore, the social welfare of all algorithms in Fig. 8 increases with the number of sites as a result of increasing available resources. On the contrary, Fig. 9 shows that the social welfare of algorithms decreases with I. When the number of jobs submitted within a fixed period of time grows, it leads to a serious shortage of resources, resulting in fewer jobs completed.

In Fig. 10, we plot the service provider’s profit across all algorithms. Notably, the profits decrease as the number of sites grows but rise with an increasing number of jobs. This is rationale because a provider with greater computational capacity aims to complete jobs as soon as possible to attract more users in the future, even if it means a reduced profit. Consequently, as the number of submitted jobs increases, so does the corresponding profit.

In addition, to analyze the impact of price functions, we further investigate the value of social welfare under different estimated values of $\gamma_d, \gamma_k$. Note that we set the value of $\gamma_d$ and $\gamma_k$ to be the same. In Fig. 11, we observe that higher $\gamma_d (\gamma_k)$ leads to lower social welfare. This is because the price of unit resource $\alpha(t) (\beta_k(t))$ increases more quickly when it meets a larger $\gamma_d (\gamma_k)$, shown in (6). In this regard, the sudden increase of price may exclude some jobs that should be accepted, leading to a smaller social welfare.

VII. Conclusion

Unbiased distributed learning (UDL) is a new emerging learning paradigm, to guarantee the quality of smart IoT services. To address the new challenges of modeling and realizing UDL, we develop an online auction-based scheduling algorithm Eris. Eris computes the best schedule for UDL jobs by using a varying number of workers and PSs such that the social welfare can be maximized, meanwhile Eris dynamically adjusts resource prices based on current resource consumption. This is the first work which studies the joint problem of scheduling and pricing for supporting emerged AI applications with one specific technical concern (the unbiasedness of ML models). Rigorous theoretical analysis demonstrates that Eris achieves a good performance ratio within polynomial running time. Our evaluations show the superiority of Eris, and we observe that Eris can improve social welfare by up to 44% compared to three representative baselines.
REFERENCES


Jinlong Pang received the BE degree from the School of Power and Mechanery, and the second BE degree from the School of Computer both from Wuhan University, China, in 2019, and the ME degree from the School of Cyber Science and Engineering, Wuhan University, in 2022. Currently, he is working toward the PhD degree with the Department of Computer Science & Engineering, University of California, Santa Cruz. His current research interests include network economics, incentive mechanisms and algorithm optimization, both in the context of machine learning, especially distributed machine learning and federated learning.

Ziyi Han received the BE degree from the School of Cyber Science and Engineering, Wuhan University, China, in 2021. She is currently working toward the master’s degree with the School of Cyber Science and Engineering, Wuhan University. Her research interests include edge computing, online learning, and network optimization.

Ruiting Zhou (Member, IEEE) received the PhD degree from the Department of Computer Science, University of Calgary, Canada, in 2018. She is an associate professor with the School of Computer Science Engineering, Southeast University. Her research interests include cloud computing, machine learning, and mobile network optimization. She has published research papers in top-tier computer science conferences and journals, including IEEE INFOCOM, ACM MobiHoc, IEEE/ACM Transactions on Networking, IEEE Journal on Selected Areas in Communications, IEEE Transactions on Mobile Computing. She serves as the TPC chair for INFOCOM workshops ICCCN 2019-2023. She also serves as a reviewer for international conferences and journals, such as IEEE ICDCS, IEEE/ACM IWQoS, IEEE SECON, IEEE Journal on Selected Areas in Communications, IEEE/ACM Transactions on Networking, IEEE Transactions on Mobile Computing, and IEEE Transactions on Cloud Computing.

Renli Zhang received the BE degree in information security from Wuhan University, China, in 2020. He is currently working toward the MS degree with the School of Cyber Science and Engineering, Wuhan University. His research interests include UAV-enabled wireless networks, network optimization, and online scheduling.

John C.S. Lui (Fellow, IEEE) received the PhD degree in computer science from UCLA. He is currently the Choh-Ming Li chair professor with the Department of Computer Science and Engineering (CSE), The Chinese University of Hong Kong (CUIHK). After his graduation, he joined the IBM Laboratory and participated in research and development projects on file systems and parallel I/O architectures. He later joined the CSE Department with CUHK. His current research interests are in online learning algorithms and applications (e.g., multi-armed bandits, reinforcement learning), machine learning on network sciences and networking systems, large scale data analytics, network/system security, network economics, large scale storage systems, and performance evaluation theory. He is an elected member of the IFIP WG 7.3, fellow of ACM, senior research fellow of the Croucher Foundation, fellow of the Hong Kong Academy of Engineering Sciences (HKAES).

Hao Chen 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.