Toward efficient online scheduling for large-scale distributed machine learning system

by

Menglu Yu

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Jia Liu, Major Professor
Peng Wei
David Fernandez-Baca

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

DEDICATION

I would like to dedicate this thesis to my advisor without whose support I would not have been able to complete this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Jia Liu for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Peng Wei and Dr. David Fernandez-Baca. Lastly, I would like to thank Dr. Bo Ji for his guidance and efforts to this work.

ABSTRACT

Thanks to the rise of machine learning (ML) and its vast applications, recent years have witnessed a rapid growth of large-scale distributed ML frameworks, which exploit the massive parallelism of computing clusters to expedite ML training jobs. However, the proliferation of large-scale distributed ML frameworks also introduces many unique technical challenges in computing system design and optimization. In a networked computing cluster that supports a large number of training jobs, a central question is how to design efficient scheduling algorithms to allocate workers and parameter servers across different machines to minimize the overall training time. Toward this end, in this paper, we develop an online scheduling algorithm that jointly optimizes resource allocation and locality decisions. Our main contributions are three-fold: i) We develop a new analytical model that considers both resource allocation and locality; ii) Based on an equivalent reformulation and close observations on the worker-parameter server locality configurations, we transform the problem into a mixed cover/packing integer program, which enables approximation algorithm design; iii) We propose a meticulously designed randomized rounding approximation algorithm and rigorously prove its performance. Collectively, our results contribute to a comprehensive and fundamental understanding of distributed ML system optimization and algorithm design.

CHAPTER 1. OVERVIEW

1.1 Introduction

Fueled by the rapid growth of data analytics and machine learning (ML) applications, recent years have witnessed an ever-increasing hunger for computing power. However, with the celebrated Moore's law nearing its end, it has been widely recognized that the only viable solution to sustain such computing power needs is to exploit *parallelism* at both machine and chip scales. Indeed, the recent success of deep learning (a revival of artificial neural networks but with a much larger number of hidden layers) is enabled by the use of distributed ML frameworks, which exploit the massive parallelism over large-scale computing clusters and the abundance of GPU resources. These distributed ML frameworks have significantly accelerated the training of deep neural network (DNN) for many applications (e.g., image and voice recognition, natural language processing, etc.). To date, prevailing distributed ML frameworks include TensorFlowAbadi et al. (2016), MXNetChen et al. (2016), Cognitive ToolKit (CNTK), CaffeJia et al. (2014), to name just a few.

However, the proliferation of distributed ML frameworks also introduces many unique technical challenges on large-scale computing system design and network resource optimization. Particularly, due to the decentralized nature, at the heart of distributed learning system optimization lies the problem of scheduling ML jobs and resource provisioning across different machines to minimize the total training time. Such scheduling problems involve dynamic and combinatorial virtual-machine-based worker and parameter server allocations, which are inherently NP-hard. Also, the allocations of workers and parameter servers should take *locality* into careful considerations, since co-located workers and parameter servers can avoid costly network communication overhead. However, locality optimization adds yet another layer of difficulty in scheduling algorithm design. Exacerbating the problem is the fact that the future arrival times of training jobs at an ML computing cluster are hard to predict, which necessitates *online* algorithm design without the knowledge of future job arrivals.

So far in the literature, there remains a lack of holistic theoretical studies that address all the aforementioned challenges. Most of the existing scheduling schemes are based on simple heuristics without performance guarantee (see Section 2.1 for more in-depth discussions). This motivates us to fill this gap in this paper and pursue efficient online scheduling designs for distributed ML resource optimization, which offer *provable* performance guarantee.

The main contribution of this paper is that we develop an online scheduling algorithmic framework that *jointly* yields resource scheduling and locality optimization decisions with strong competitive ratio performance. Furthermore, we reveal interesting insights on how distributed ML frameworks affect online resource scheduling optimization. Our main technical results are summarized as follows:

- By abstracting the architectures of prevailing distributed ML frameworks, we formulate an online resource scheduling optimization problem that: i) models the training of ML jobs based on asynchronous stochastic gradient descent method; and ii) explicitly takes *locality* optimization into considerations. We show that, due to the heterogeneous internal (between virtual machines) and external (between physical machines) communications, the locality-aware scheduling problem contains *non-deterministic* constraints and is far more complex compared to the existing works that are locality-oblivious (see, e.g., Chun et al. (2016); Bao et al. (2018)).

- To solve the locality-aware scheduling problem, we develop an equivalent problem reformulation to enable subsequent developments of online approximation algorithms. Specifically, upon carefully examining the locality configurations of worker-server relationships, we are able to transform the original problem to a special-structured integer nonlinear program with mixed cover/packing-type constraints, whose salient features enable low-complexity approximation algorithm design with provable performance.

- To tackle the integer nonlinear problem with mixed cover/packing-type constraints, we propose an approximation algorithm based on a meticulously designed randomized rounding scheme and then rigorously prove its performance. We note that the results of our randomized rounding scheme is general and could be of independent theoretical interest. Finally, by putting all algorithmic

designs together, we construct a primal-dual online resource scheduling (PD-ORS) scheme, which has an overall approximation ratio that only *logarithmically* depends on ML jobs characteristics (e.g., required epochs, data chunks, mini-batches, etc.).

Collectively, our results contribute to a comprehensive and fundamental understanding of distributed machine learning systems optimization. The remainder of this paper is organized as follows. In Section 2.1, we review the literature to put our work in comparative perspectives. Section 3.1 introduces the system model and problem formulation. Section 3.2 presents our algorithms and their performance analysis. Section 4.1 presents numerical results and Section 5.1 concludes this paper.

CHAPTER 2. REVIEW OF LITERATURE

2.1 Related Work

As mentioned in Section 1.1, due to the high computational load of ML applications, many distributed ML frameworks have been proposed to leverage modern large-scale computing clusters. From an abstraction standpoint, a common architecture behind these distributed ML frameworks is the provisioning of virtual-machine-based parameter servers and workers. Coupled with the *iterative* ML training phase based on asynchronous stochastic gradient descent (Async-SGD), the interactions between machines in the computing cluster are significantly different from those in traditional cloud computing platforms. As a result, existing job scheduling algorithms for cloud systems (see, e.g., Huang et al. (2015); Chen and Liu (2017) and references therein) do not work well for distributed ML frameworks.

In the distributed ML systems literature, most of the early attempts (see, e.g., Li et al. (2014); Chilimbi et al. (2014) and references therein) only considered static allocation of workers and parameter servers. To our knowledge, the first work on understanding the performance of distributed ML frameworks is Yan et al. (2015), where Yan *et al.* developed analytical models to quantify the impacts of models-data partitioning and system provisioning for DNN. Subsequently, Chun *et al.* Chun et al. (2016) developed heuristic dynamic system reconfiguration algorithms to allocate workers and parameter servers to minimize the runtime, but without providing optimality guarantee. In the literature, the first dynamic distributed scheduling algorithm with optimality guarantee is reported in Sun et al. (2017), where Sun *et al.* used standard mixed integer linear program (MILP) solver to dynamically compute the worker-parameter server partition solutions. However, due to the NP-hardness of MILP, this approach cannot be scaled up to handle large-size distributed ML systems in practice.

The most related work to ours is Bao et al. (2018), where Bao *et al.* developed an online primal-dual approximation algorithm called OASiS to solve the scheduling problem for distributed ML systems. However, our work differs from Bao et al. (2018) in the following key aspects: In Bao et al. (2018), the workers and parameter servers are allocated on two *strictly separated* sets of physical machines. In other words, *no* worker and parameter server can share the same physical machine in their setting. By stark contrast, in this work, we allow workers and parameter servers to be *co-located* on the same physical machine to increase communication and resource utilization efficiency. As will be shown later, the co-location setting leads to an integer *non-convex* optimization problem with *non-deterministic* constraints, which is much harder than that in Bao et al. (2018) and necessitates new algorithm designs. More importantly, we consider the setting with co-located workers and parameter servers because it is the *reality* for ML frameworks in practice (see, e.g., Chen et al. (2016)). The co-location setting was considered inPeng et al. (2018) recently. However, the scheduling algorithm therein is a heuristic and does not provide performance guarantee. This motivates us to develop new algorithms with provable performance to fill this gap.

CHAPTER 3. METHODS AND PROCEDURES

3.1 System Model and Problem Formulation

In this section, we first provide a quick overview on the architecture of distributed ML frameworks to familiarize readers with the necessary background. Then, we will introduce our analytical modes for ML jobs and resource constraints, as well as the overall problem formulation.

1) Distributed Machine Learning: A Primer. As illustrated in Fig. 3.1, a distributed ML system is usually implemented over a connected computing cluster that contains multiple physical machines. Conceptually, the key components of a distributed ML system include parameter servers, workers, and the training dataset. Parameter servers and workers are usually implemented on virtual machines and could spread over multiple physical machines, as shown in Fig. 3.2. In practice, parameters of the same job are evenly divided among its parameter servers. The training dataset of an ML job is stored in a database and divided into equal-sized data chunks. Each data chunk contains multiple equal-sized mini-batches.

To date, most distributed ML frameworks adopt the *asynchronous stochastic gradient descent method* (Async-SGD) as the default training algorithm due to its low-complexity. Under Async-SGD, the interactions between workers and parameter servers are illustrated in Fig 3.3: Once becoming idle, a worker will request the current values of the parameters (e.g., the weights of a DNN) from all parameter servers. Meanwhile, the worker retrieves a new data chunk from the database. During the training stage, each worker processes one mini-batch from the data chunk at a time to compute a gradient (i.e., directions and magnitudes of parameter changes). For example, in a DNN model, gradients can be computed by the well-known "back-propagation" approach. Upon finishing a mini-batch, the worker sends the gradient back to parameter servers and then continues to work on the next mini-batch. After finishing the current data chunk, the worker will repeat the same process on a new data chunk. On the parameter server side, parameters are updated

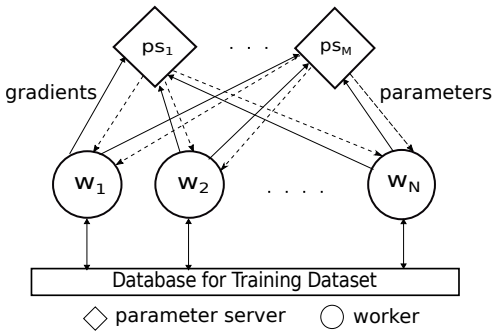

Figure 3.1: An illustration of a distributed machine learning framework.

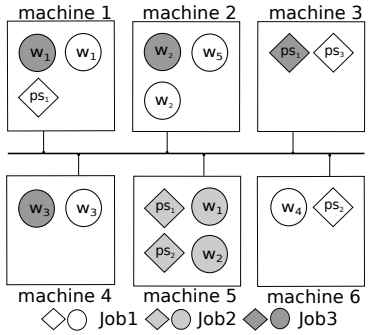

Figure 3.2: Colocated parameter servers and workers on physical machines.

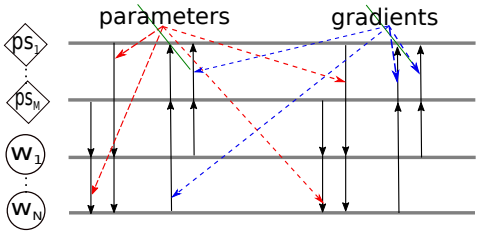

Figure 3.3: The workflow of Async-SGD on a distributed ML system.

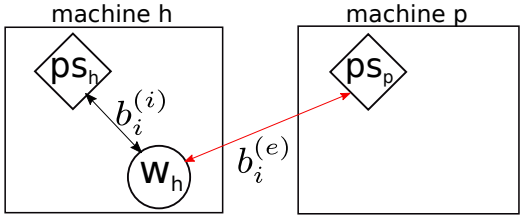

Figure 3.4: $b_i(h,p)$'s value is locality-dependent (internal or external).

as $w[k] = w[k-1] + \alpha_k g[k]$, where $w[k]$, α_k, and $g[k]$ denote the parameter values, step-size, and stochastic gradient in the k-th update, respectively.

We can further see from Fig. 3.3 that the training progress at different workers is *not* synchronized: Each parameter server updates its parameters without coordinating with other parameter servers, hence the name Async-SGD. It has been shown that Async-SGD achieves the same $O(1/\sqrt{k})$ convergence rate as its synchronous counterpartLian et al. (2015); Huo and Huang (2017), while avoiding technical complexities such as maintaining a common clock, bottlenecks due to slower machine(s), periodic spikes of information exchanges and congestions, etc. Upon understanding distributed ML systems, we will develop analytical models to facilitate scheduling algorithm design.

2) Learning Job Modeling: In this paper, we consider a time-slotted system. The scheduling time-horizon is denoted as $\mathcal{T}$ with $|\mathcal{T}| = T$. We use $\mathcal{I}$ to represent the set of training jobs and let a_i denote the arrival time-slot of job $i \in \mathcal{I}$. We let $\mathcal{H}$ represent the set of physical machines.

We use $w_{ih}[t], s_{ih}[t] \in \mathbb{Z}_+$ to represent the numbers of workers and parameter servers on machine $h \in \mathcal{H}$ in each time-slot $t \geq a_i$, respectively. Further, we let $\mathcal{P}_i[t] \triangleq \{h \in \mathcal{H} | s_{ih}[t] > 0\}$ and $\mathcal{W}_i[t] \triangleq \{h \in \mathcal{H} | w_{ih}[t] > 0\}$ denote the sets of machines having parameter servers and workers for job i in time-slot t, respectively.

We use a binary variable $x_i \in \{0, 1\}$ to indicate whether job i is admitted ($x_i = 1$) or not ($x_i = 0$). We use τ_i to denote the training time for each mini-batch of job i. We let $b_i(h, p)$ denote the data rate of the link between a worker for job i (on machine h) and a parameter server (on machine p). As shown in Fig. 3.4, the value of $b_i(h, p)$ is *locality-dependent*:

$$b_i(h,p) = \begin{cases} b_i^{(i)}, & \text{if } h = p, \\ b_i^{(e)}, & \text{otherwise,} \end{cases}$$

where $b_i^{(i)}$ and $b_i^{(e)}$ denote the internal and external rates, respectively, with $b_i^{(i)} \gg b_i^{(e)}$ in practice. Let g_i denote the size of gradient and parameters of job i. Then, $\tau_i + \frac{2g_i}{\min_{p \in \mathcal{P}_i[t]} b_i(h,p)}$ is the total amount of time to train a mini-batch on machine h and then communicate the result with its parameter server(s). The number of mini-batches trained on machine h for job i in time-slot t can then be computed as: $w_{ih}[t]/(\tau_i + \frac{2g_i}{\min_{p \in \mathcal{P}_i[t]} b_i(h,p)})$.

Suppose that, for job i, there are K_i data chunks available for training with M_i mini-batches per data chunk. In ML systems, an epoch is defined as a round of training that exhausts all data chunks. We let E_i denote the number of epochs needed by job i. Then, the total number of mini-batches to be processed for job i over the entire training process is $E_i K_i M_i$. To make sure that there are sufficient workers allocated for job i over the entire training horizon, we have:

$$\sum_{t \in \mathcal{T}} \sum_{h \in \mathcal{H}} \frac{w_{ih}[t]}{\tau_i + \frac{2g_i}{\min_{p \in \mathcal{P}_i[t]} b_i(h,p)}} \geq x_i E_i K_i M_i, \forall i \in \mathcal{I}. \tag{3.1}$$

We note that, with co-located workers and parameter servers on each machine, Eq. (3.1) is *non-deterministic* due to the existence of the $\min\{\cdot\}$ operator. As will be shown later, this non-determistic constraint makes the scheduling design far more complicated than related worksLi et al. (2014); Chilimbi et al. (2014); Chun et al. (2016); Bao et al. (2018).

To model that the largest number of assigned concurrent workers is no more than the number of data chunks in each time slot (otherwise, some workers will be idle), we have:

$$\sum_{h\in\mathcal{H}} w_{ih}[t] \le x_i K_i, \quad \forall i \in \mathcal{I}, a_i \le t \le T. \tag{3.2}$$

3) Resource Constraint Modeling: We let $\mathcal{R}$ denote the set of resources (e.g., CPU/GPU, memory, storage, etc.). Let α_i^r and β_i^r be the amount of type-r resource required by a worker and a parameter server for job i, respectively. Let C_h^r be the capacity of type-r resource on machine h. To ensure the allocated resources do not exceed type-r's limit, we have:

$$\sum_{i\in\mathcal{I}} (\alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t]) \le C_h^r, \forall t \in \mathcal{T}, r \in \mathcal{R}, h \in \mathcal{H}. \tag{3.3}$$

In a distributed ML system, the parameter servers should not be the bottleneck during gradient/parameter exchanges. To this end, we let B_i denote the communication data rate of job i's parameter server. Then, we have:

$$\sum_{h\in\mathcal{W}_i[t]} \sum_{p\in\mathcal{P}_i[t]} w_{ih}[t] b_i(h,p) \le \sum_{h\in\mathcal{P}_i[t]} s_{ih}[t] B_i, \forall i \in \mathcal{I}, t \in \mathcal{T}. \tag{3.4}$$

In practice, the number of parameter servers is upper bounded by the number of workers in each time slot for any job. This can be modeled as follows:

$$\sum_{h\in\mathcal{P}_i[t]} s_{ih}[t] \le \sum_{h\in\mathcal{W}_i[t]} w_{ih}[t], \quad \forall i \in \mathcal{I}, t \in \mathcal{T}. \tag{3.5}$$

Note that for job i, its completion time $\tilde{t}_i$ corresponds to the latest time-slot where there remain some active workers allocated for it. Therefore, we have:

$$\tilde{t}_i = \arg\max_{t\in\mathcal{T}} \left\{ \sum_{h\in\mathcal{H}} w_{ih}[t] > 0 \right\}, \quad \forall i \in \mathcal{I}. \tag{3.6}$$

To ensure that no workers and parameter servers are allocated before job i's arrival, we have:

$$w_{ih}[t] = s_{ih}[t] = 0, \quad \forall i \in \mathcal{I}, h \in \mathcal{H}, t < a_i. \tag{3.7}$$

4) Objective Function and Problem Statement: Let $u_i(\tilde{t}_i - a_i)$ be the utility function for job i, which is non-increasing with respect to the training time $\tilde{t}_i - a_i$. That is, the more time

job i takes to finish, the less the utility gain it will obtain. In this paper, our goal is to maximize the overall utility for all jobs. Putting all constraints and the objective function together, the offline (with knowledge of a_i, $\forall i$) distributed ML resource scheduling problem (DMLRS) can be formulated as:

$$\textbf{DMLRS:} \quad \underset{x,w,s}{\text{Maximize}} \quad \sum_{i\in\mathcal{I}} x_i u_i(\tilde{t}_i - a_i)$$
$$\text{subject to} \quad \text{Constraints (3.1) – (3.7).}$$

For quick reference, we summarize the key notations used in this paper in Table 3.1. It can be seen that Problem DMLRS is an integer nonlinear program, which is NP-hard in generalHochbaum (1997). Also, Problem DMLRS involves two *non-deterministic* constraints in (3.1) and (3.6), which are not amenable for conventional optimization techniques. Moreover, the arrivals $\{a_i, \forall i\}$ are unknown, which necessitates online optimization. Overcoming these challenges constitutes the main subjects in the next section.

3.2 Solution Approach and Online Scheduling Algorithm Design

In this section, we structure the key components of our online scheduling algorithm design for solving Problem DMLRS into three steps from Sections 3.2.1 to 3.2.3. Theoretical performance results are provided in Section 3.2.4.

3.2.1 Handling Non-Deterministic Completion Time Constraint (3.6)

The first obstacle in solving Problem DMLRS stems from the non-deterministic "argmax" structure in constraint (3.6). In the literature, a commonly used technique to handle argmax-type constraints is via an equivalent reformulation that enumerates all possible schedules in the system (see, e.g., Bao et al. (2018)).

Specifically, we let Π_i be the set of all feasible schedules for job $i \in \mathcal{I}$ that satisfy constraints (3.1), (3.2), (3.4), and (3.5). Each schedule $\pi_i \in \Pi_i$ is defined by the numbers of workers $w_{ht}^{\pi_i}$ and parameter servers $s_{ht}^{\pi_i}$ allocated for job i on machine h in each time-slot t, i.e., $\pi_i \triangleq \{w_{ht}^{\pi_i}, s_{ht}^{\pi_i}, \forall t \in$

$\mathcal{T}, h \in \mathcal{H}\}$. We define a binary variable $x_{\pi_i} \in \{0,1\}$ that is equal to 1 if job i is admitted and scheduled according to π_i or 0 otherwise. We let $\tilde{t}_{\pi_i}$ denote job i's completion time under schedule π_i. Then, one can equivalently reformulate Problem DMLSR as:

R-DMLRS:

$$\underset{\mathbf{x}}{\text{Maximize}} \quad \sum_{i\in\mathcal{I}} \sum_{\pi_i\in\Pi_i} x_{\pi_i} u_i(\tilde{t}_{\pi_i} - a_i)$$

$$\text{subject to} \quad \sum_{i\in\mathcal{I}} \sum_{\pi_i\in\Gamma(t,h)} (\alpha_i^r w_{ht}^{\pi_i} + \beta_i^r s_{ht}^{\pi_i}) x_{\pi_i} \le C_h^r, \tag{3.8}$$

$$\forall t \in \mathcal{T}, r \in \mathcal{R}, h \in \mathcal{H},$$

$$\sum_{\pi_i\in\Pi_i} x_{\pi_i} \le 1, \quad \forall i \in \mathcal{I}, \tag{3.9}$$

$$x_{\pi_i} \in \{0,1\}, \quad \forall i \in \mathcal{I}, \pi_i \in \Pi_i,$$

where we use $\Gamma(t,h)$ to represent the set of feasible schedules that use machine h to deploy workers or parameter servers in time-slot t. Constraint (3.8) guarantees that, in any time-slot t and on any machine h, the total amount of consumed type-r resources will not exceed the capacity limit C_h^r. Constraint (3.9) ensures that, for each job i, at most one feasible schedule from Π_i will be selected. It is easy to see that the constraints in Problem R-DMLRS are equivalent to those in Problem DMLSR. Hence, a feasible solution to Problem R-DMLRS has a corresponding feasible solution to the original Problem DMLSR, and vice versa. Yet, the non-deterministic constraint (3.6) no longer exists in Problem R-DMLSR.

However, it remains difficult to directly solve Problem R-DMLSR since it has an exponential number of binary decision variables (x_{π_i}'s) due to the combinatorial nature of the problem. In this paper, we adopt a primal-dual online algorithmic framework, which is an effective approach to address this kind of challenge in the literature (see, e.g., Buchbinder and (Seffi) Naor (2009); Bao et al. (2018)).

3.2.2 Online Primal-Dual Framework for Problem R-DMLSR

The fundamental rationale behind the primal-dual approach is that, in the dual of Problem R-DMLSR, the number of dual variables is polynomial. Meanwhile, although there are an exponential number of constraints in the dual problem, one only needs to be concerned with the set of active (binding) constraints, which are easier to deal with.

To see this, we associate two sets of dual variables (prices) $p_h^r[t] \geq 0$, $\forall t \in \mathcal{T}$, $h \in \mathcal{H}$, $r \in \mathcal{R}$ and $\lambda_i > 0$, $i \in \mathcal{I}$, with constraints (3.8) and (3.9), respectively. Then, following the standard procedure of dualization and relaxing the integrality constraints, we obtain the following dual problem:

D-R-DMLRS:

$$\underset{\boldsymbol{\lambda},\mathbf{p}}{\text{Minimize}} \quad \sum_{i\in\mathcal{I}} \lambda_i + \sum_{t\in\mathcal{T}} \sum_{h\in\mathcal{H}} \sum_{r\in\mathcal{R}} p_h^r[t] C_h^r \tag{3.10}$$

$$\text{subject to} \quad \lambda_i \geq u_i(\tilde{t}_{\pi_i} - a_i) - \sum_{t\in\mathcal{T}(\pi_i)} \sum_{h\in\mathcal{H}(\pi_i[t])} \sum_{r\in\mathcal{R}} (\alpha_i^r w_{ht}^{\pi_i} + \beta_i^r s_{ht}^{\pi_i}) p_h^r[t], \quad \forall i \in \mathcal{I}, \pi_i \in \Pi_i, \tag{3.11}$$

$$p_h^r[t] \geq 0, \quad \forall t \in \mathcal{T}, h \in \mathcal{H}, r \in \mathcal{R},$$

$$\lambda_i \geq 0, \quad \forall i \in \mathcal{I},$$

where $\mathcal{T}(\pi_i)$ denotes the time-slots utilized by schedule π_i and $\mathcal{H}(\pi_i[t])$ denotes the set of machines containing workers and/or parameter servers under π_i in time-slot t. Here, $p_h^r[t]$ can be viewed as the price for type-r resource in time t, and λ_i can be viewed as the payoff of admitting job i under π_i.

Next, we examine the structural properties of Problem D-R-DMLRS. To minimize (3.10), we tend to reduce λ_i and $p_h^r[t]$ as much as possible until they hit zero. However, as λ_i and $p_h^r[t]$ decrease, the left-hand-side (LHS) and right-hand-side (RHS) of (3.11) decreases and increases, respectively (note that $u_i(\tilde{t}_{\pi_i} - a_i)$ is a constant given π_i). Therefore, λ_i will eventually drop to a value λ_i^*, which is equal to maximum of the RHS of (3.11) achieved by some schedule π_i^* and dual price $p_h^{r*}[t]$, i.e.,

$$\lambda_i^* = u_i(\tilde{t}_{\pi_i^*} - a_i) - \sum_{t\in\mathcal{T}(\pi_i^*)} \sum_{h\in\mathcal{H}(\pi_i^*[t])} \sum_{r\in\mathcal{R}} (\alpha_i^r w_{ht}^{\pi_i^*} + \beta_i^r s_{ht}^{\pi_i^*}) p_h^{r*}[t].$$

This optimality structural insight implies that Problem D-R-DMLRS is equivalent to finding an optimal schedule π_i^* and dual price $p_h^{r*}[t]$ to maximize the RHS of (3.11), which motivates the general primal-dual online resource scheduling (PD-ORS) framework in Algorithm 1 (also seeBuchbinder and (Seffi) Naor (2009); Bao et al. (2018)):

Algorithm 1: Primal-Dual Online Resource Scheduling (PD-ORS).

Initialization:

1. Let $w_{ih}[t] = 0$, $s_{ih}[t] = 0$, $\forall i, t, h$. Let $\rho_h^r[t] = 0$, $\forall h, r, t$. Choose some appropriate initial values for $p_h^r[0]$.

Main Loop:

2. Upon the arrival of job i, determine a schedule π_i^* to maximize the RHS of (3.11) and its corresponding payoff λ_i using Algorithm 2 (*to be specified*).
3. If $\lambda_i > 0$, set $x_i = 1$. Set $w_{ih}[t]$ and $s_{ih}[t]$ according to schedule π_i^*, $\forall t \in \mathcal{T}(\pi_i^*)$, $h \in \mathcal{H}(\pi_i^*[t])$. Update $\rho_h^r[t] \leftarrow \rho_h^r[t] + \alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t]$, $\forall t \in \mathcal{T}(\pi_i^*)$, $h \in \mathcal{H}(\pi_i^*[t])$, $r \in \mathcal{R}$. Update $p_h^r[t] = Q_h^r(\rho_h^r[t])$, $\forall t \in \mathcal{T}(\pi_i^*)$, $h \in \mathcal{H}(\pi_i^*[t])$, $r \in \mathcal{R}$. Schedule job i according to π_i^* and go to Step 2.
4. If $\lambda_i \leq 0$, set $x_i = 0$ and reject job i and go to Step 2.

The intuition of Algorithm 1 is as follows: By the complementary slackness condition of the Karush-Kuhn-Tucker (KKT) conditionsBazaraa et al. (2006), the primal constraint (3.9) must be tight when dual variable $\lambda_i > 0$, which implies that $x_i = 1$ (Step 3) in the original Problem DMLSR. Otherwise, if $\lambda_i = 0$, then the RHS of (3.11) is non-positive, meaning the utility is low compared to the cost of resource consumption under schedule π_i^*. Therefore, we should reject job i ($x_i = 0$ in Step 4).

However, in order for the PD-ORS algorithm to work, two challenging components need to be specified:

- *How to determine an optimal schedule π_i^* in Step 2?* Due to the exponential size of Π_i, it is intractable to enumerate all feasible schedules in Π_i. In fact, we will show later that this sub-problem is an NP-hard integer programming problem with mixed cover/packing-type constraints, meaning that we can at best pursue approximate solutions unless P = NP;

- *How to design the cost update function $Q_h^r(\cdot)$ for $p_h^r[t]$?* Note that since jobs arrive in an online fashion, we do not have the knowledge of their future arrivals. Therefore, even if we know π_i^*, we cannot directly compute $p_h^r[t]$ from the linear program (LP) implied by Problem D-R-DMLRS.

In what follows, we will first focus on designing $Q_h^r(\cdot)$ and defer the finding of π_i^* to Section 3.2.3. For the design of $Q_h^r(\cdot)$, we adopt the existing approach in the literature Buchbinder and (Seffi) Naor (2009); Bao et al. (2018) and consider the following choice of $Q_h^r(\cdot)$:

$$Q_h^r(\rho_h^r[t]) = L(U^r/L)^{\frac{\rho_h^r[t]}{C_h^r}}, \tag{3.12}$$

where constants U^r, $\forall r$, and L are defined as:

$$U^r \triangleq \max_{i\in\mathcal{I}} \frac{u_i(\lceil E_i M_i(\tau_i + 2g_i/b_i^{(i)})\rceil - a_i)}{\alpha_i^r + \beta_i^r}, \quad \forall r \in \mathcal{R}, \tag{3.13}$$

$$L \triangleq \frac{1}{2\mu} \min_{i\in\mathcal{I}} \frac{u_i(T - a_i)}{\sum_{r\in\mathcal{R}} \lceil E_i K_i M_i(\tau_i + 2g_i/b_i^{(e)})\rceil(\alpha_i^r + \beta_i^r)}. \tag{3.14}$$

The scaling factor μ in the definition of L satisfies $\frac{1}{\mu} \leq \frac{\lceil E_i K_i M_i(\tau_i + 2g_i/b_i^{(e)})\rceil \sum_{r\in\mathcal{R}}(\alpha_i^r+\beta_i^r)}{T\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} C_h^r}$. U^r represents the maximum unit-resource job utility to deploy workers and parameter servers with type-r resource . Here, $u_i(\lceil E_i M_i(\tau_i + 2e_i/b_i^{(i)})\rceil - a_i)$ is the largest utility job i can achieve by using the maximum number of *co-located* workers and parameter servers (hence communicating rate is $b_i^{(i)}$) at all times during all E_i epochs, so that $\lceil E_i M_i(\tau_i + 2e_i/b_i^{(i)})\rceil - a_i$ is the fastest possible job completion time. Similarly, L represents the minimum unit-time unit-resource job utility among all jobs, with $u_i(T - a_i)$ being the smallest utility for job i, and workers and parameter servers communicate at slow external rate $b_i^{(e)}$.

The $Q_h^r(\cdot)$ function has three important properties (proofs follow similarly from Buchbinder and (Seffi) Naor (2009); Bao et al. (2018) and are omitted due to space limitaton): i) At $t = 0$, $\rho_h^r[0] = 0, \forall h \in \mathcal{H}, r \in \mathcal{R}$. Hence, the price $p_h^r[0] = L$ is the lowest, $\forall h, r$, and any job can be admitted; ii) As the amount of allocated resources increases, the price increases exponentially fast to quickly reject early coming jobs with lower utility, so as to reserve resources for later arrived jobs with higher utility; iii) When some type-r resource is exhausted, i.e., $\rho_h^r[t] = C_h^r, \exists r \in \mathcal{R}$,

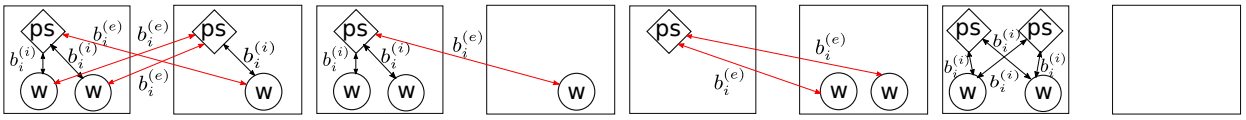

Figure 3.6 $|\mathcal{P}_i[t]| \neq 1$.

Figure 3.7 $|\mathcal{P}_i[t]| = 1, |\mathcal{W}_i[t]| \neq 1$.

Figure 3.8 $|\mathcal{P}_i[t]|=|\mathcal{W}_i[t]|=1, \mathcal{P}_i[t] \neq \mathcal{W}_i[t]$.

Figure 3.9 $|\mathcal{P}_i[t]|=|\mathcal{W}_i[t]|=1, \mathcal{P}_i[t]=\mathcal{W}_i[t]$.

Figure 3.10: Values of $\min_{p\in\mathcal{P}_i[t]} \frac{2g_i}{b_i(h,p)}$ under various settings of $\mathcal{P}_i[t]$ and $\mathcal{W}_i[t]$. Clearly, $\min_{p\in\mathcal{P}_i[t]} \frac{2g_i}{b_i(h,p)} = 2g_i/b_i^{(i)}$ if and only if under setting (d).

$Q_h^r[C_h^r] = U^r$ and no job that requires type-r resources will be admitted since the U^r is the highest price.

3.2.3 Determining π_i^* in Step 2 of Algorithm 1

Now, we focus on the subproblem of finding a schedule π_i^* in Step 2 of Algorithm 1 to maximize the RHS of (3.11), i.e.,

Find-Sch:

$$\underset{\tilde{t}_i,\mathbf{w},\mathbf{s}}{\text{Max}}\ u_i(\tilde{t}_i - a_i) - \sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} p_h^r[t](\alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t])$$

$$\text{s.t. } \alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}[t] \leq \hat{C}_h^r[t], \quad \forall t \in \mathcal{T}, r \in \mathcal{R}, h \in \mathcal{H},$$

$$\text{Constraints (3.1)(3.2)(3.4)-(3.7) for } x_i = 1,$$

where $\hat{C}_h^r[t] \triangleq C_h^r - \rho_h^r[t]$. However, Problem Find-Sch is a challenging integer nonlinear optimization problem with *non-deterministic* constraints in (3.1). In what follows, we will address these challenges one by one.

1) Handling nonlinear $u_i(\cdot)$ function: Observe that, given $\tilde{t}_i$, Problem FindSch can be simplified as follows:

$$\underset{\mathbf{w},\mathbf{s}}{\text{Minimize}} \quad \sum_{t\in[a_i,\tilde{t}_i]}\sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} p_h^r[t](\alpha_i^r w_{ih}[t]+\beta_i^r s_{ih}[t]) \tag{3.15}$$

$$\text{subject to} \quad \sum_{t\in[a_i,\tilde{t}_i]}\sum_{h\in\mathcal{H}} \frac{w_{ih}[t]}{\tau_i+\min_{p\in\mathcal{P}_i[t]}\frac{2g_i}{b_i(h,p)}} \geq V_i M_i, \tag{3.16}$$

$$\alpha_i^r w_{ih}[t]+\beta_i^r s_{ih}[t]\leq \hat{C}_h^r[t], \forall r,h,\forall t\in[a_i,\tilde{t}_i], \tag{3.17}$$

$$\text{Constraints (3.2)(3.4)(3.5) for all } t\in[a_i,\tilde{t}_i],$$

where $V_i \triangleq E_i K_i$ represents the total training workload. Note that in Problem (3.15), the only coupling constraint is (3.16). This observation leads to a dynamic programming approach to solve Problem (3.15): First, consider the following problem if training workload in a time-slot t is known (denoted as $V_i[t]$):

$$\underset{w_{ih}[t],s_{ih}[t],\forall h}{\text{Minimize}} \quad \sum_{h\in\mathcal{H}}\sum_{r\in\mathcal{R}} p_h^r[t](\alpha_i^r w_{ih}[t]+\beta_i^r s_{ih}[t]) \tag{3.18}$$

$$\text{subject to} \quad \sum_{h\in\mathcal{H}} \frac{w_{ih}[t]}{\tau_i+\min_{p\in\mathcal{P}_i[t]}\frac{2g_i}{b_i(h,p)}} \geq V_i[t] M_i, \tag{3.19}$$

$$\text{Constraints (3.2)(3.4)(3.5)(3.17) for the given } t.$$

Let $\Theta(\tilde{t}_i, V_i)$ and $\theta(t, V_i[t])$ denote the optimal values of Problems (3.15) and (3.18), respectively. Then, Problem (3.15) is equivalent to the following dynamic program (DP):

$$\Theta(\tilde{t}_i, V_i) = \min_{v\in[0,V_i]} \left\{\theta(\tilde{t}_i, v)+\Theta(\tilde{t}_i-1, V_i-v)\right\}. \tag{3.20}$$

Then, by enumerating all $\tilde{t}_i\in[a_i,T]$ and solving the dynamic program $\Theta(\tilde{t}_i, V_i)$ in (3.20) for every choice of $\tilde{t}_i$, we can solve Problem Find-Sch and determine the optimal schedule π_i^*. We summarize this procedure in Algorithm 2 and Algorithm 3:

Algorithm 2: Determine π_i^* in Step 2 of Algorithm 1.

Initialization:

1. Let $\tilde{t}_i = a_i$. Let $\lambda_i = 0$, $\pi_i^* = \varnothing$, $w_{ih}[t] = s_{ih}[t] = 0$, $\forall t, h$.

Main Loop:

2. Compute $\Theta(\tilde{t}_i, V_i)$ by solving the DP in (3.20) using Algorithm 3. Denote the resulted schedule as π_i. Let $\lambda_i' = u_i(\tilde{t}_i - a_i) - \Theta(\tilde{t}_i, V_i)$. If $\lambda_i' > \lambda_i$, let $\lambda_i \leftarrow \lambda_i'$ and $\pi_i^* \leftarrow \pi_i$.

3. Let $\tilde{t}_i \leftarrow \tilde{t}_i + 1$. If $\tilde{t}_i > T$, stop; otherwise, go to Step 2.

Algorithm 3: Dynamic Programming for Solving $\Theta(\tilde{t}_i, V_i)$.

Initialization:

1. Let *cost-min* $= \infty$, $\pi_i = \varnothing$, and $v = 0$.

Main Loop:

2. Compute $\theta(\tilde{t}_i, v)$ using Algorithm 4 (*to be specified*). Denote the resulted cost and schedule as *cost-v* and $\hat{\pi}_i$.

3. Compute $\Theta(\tilde{t}_i - 1, V_i - v)$ by calling Algorithm 3 itself. Denote the resulted cost and schedule as *cost-rest* and $\tilde{\pi}_i$.

4. If *cost-min* $>$ *cost-v* $+$ *cost-rest* then *cost-min* $=$ *cost-v* $+$ *cost-rest* and let $\pi_i \leftarrow \hat{\pi}_i \cup \tilde{\pi}_i$.

5. Let $v \leftarrow v + 1$. If $v > V_i$ stop; otherwise go to Step 2.

---

2) Solving $\theta(t, v)$ (i.e., Problem (3.18)): In Algorithm 3, a key unresolved question is how to compute $\theta(t, v)$ in Step 2 (i.e., solving Problem (3.18)). To solve (3.18), a main obstacle is the *non-deterministic* constraint in (3.19), where $b_i(h, p)$ can be either $b_i^{(i)}$ or $b_i^{(e)}$. Therefore, we need to handle both cases in $\min_{p \in \mathcal{P}_i[t]} \frac{2s_{ih}[t]g_i}{\sum_{h \in \mathcal{H}} s_{ih}[t] b_i(h,p)}$. To this end, we observe a simple fact about $\min_{p \in \mathcal{P}_i[t]} \frac{2g_i}{b_i(h,p)}$ (also see Fig. 3.10), which will be useful in our subsequent analysis (proof omitted due to its simplicity):

Fact 1 *The function* $\min_{p \in \mathcal{P}_i[t]}(2g_i/b_i(h,p)) = 2g_i/b_i^{(i)}$ *if and only if* $|\mathcal{P}_i[t]| = |\mathcal{W}_i[t]| = 1$ *and* $\mathcal{P}_i[t] = \mathcal{W}_i[t]$*; otherwise,* $\min_{p \in \mathcal{P}_i[t]}(2g_i/b_i(h,p)) = 2g_i/b_i^{(e)}$.

2-1) Internal communication: To design an algorithm to handle cases with internal communication rate $b_i^{(i)}$, we start from analyzing the optimality structure of Problem (3.18). We note that if we temporarily ignore the workload-coupling constraint (3.19), Problem (3.18) can be *decomposed*

as follows:

$$\sum_{r\in\mathcal{R}}\sum_{h\in\mathcal{H}}\left\{\begin{array}{ll}\text{Min } p_h^r[t](\alpha_i^r w_{ih}[t]+\beta_i^r s_{ih}^r[t]),\\ \text{s.t. } \alpha_i^r w_{ih}[t]+\beta_i^r s_{ih}[t]\le \hat{C}_h^r[t],\\ \quad\text{Constraints (3.2)(3.4)(3.5) for given } r,h\end{array}\right\}, \tag{3.21}$$

in which each summand in (3.21) is an integer linear program (ILP) having a trivial solution $w_{ih}[t]=s_{ih}[t]=0$, $\forall h\in\mathcal{H}$. However, $w_{ih}[t]=0$, $\forall h\in\mathcal{H}$, clearly violates the workload constraint (3.19). Thus, when (3.21) is optimal, there should be *exactly one* machine $h'\in\mathcal{H}$ with $w_{ih'}[t]\ge 1$ and *exactly one* machine $h''\in\mathcal{H}$ with $s_{ih''}[t]\ge 1$. This observation shows that the optimal solution of (3.18) *tends to favor* $|\mathcal{P}_i[t]|=|\mathcal{W}_i[t]|=1$ if workload-coupling constraint (3.19) is not binding, which matches the internal case condition in Fact 1. This observation suggests that, for the "$b_i^{(i)}$" case, we should check the workload constraint (3.19) on each machine one by one (i.e., ensuring $|\mathcal{P}_i[t]|=|\mathcal{W}_i[t]|=1$ and $\mathcal{P}_i[t]=\mathcal{W}_i[t]$). In this setting, the workload constraint (3.19) becomes $w_{ih}[t]\ge V_i[t]M_i(\tau_i+2g_i/b_i^{(i)})$. After checking all machines, choose, if any, the machine h that satisfies (3.19) and has the lowest cost. Then, we return the schedule $(w_{ih}[t],s_{ih}[t])$ and the corresponding cost value.

2-2) External communication: For those settings that do not satisfy $|\mathcal{P}_i[t]|=|\mathcal{W}_i[t]|=1$ and $\mathcal{P}_i[t]=\mathcal{W}_i[t]$, Fact 1 indicates that parameter servers and workers are communicating at external rate $b_i^{(e)}$. In this case, the workload constraint (3.19) becomes: $\sum_{h\in\mathcal{H}} w_{ih}[t]\ge V_i[t]M_i(\tau_i+2s_{ih}[t]g_i/(\sum_{h\in\mathcal{H}} s_{ih}[t]b_i^{(e)}))$. For convenience, we let $m_w[t]\triangleq\lceil V_i[t]M_i(\tau_i+2s_{ih}[t]g_i/(\sum h\in\mathcal{H}s_{ih}[t]b_i^{\{e\}}))\rceil$ and $m_p[t]\triangleq\lceil m_w[t]b_i^{\{e\}}/B_i\rceil$ represent the minimum required numbers of workers and parameter servers given workload $V_i[t]$, respectively. Then, we can rewrite Problem (3.18) as:

$$\underset{w_{ih}[t],s_{ih}[t],\forall h}{\text{Minimize}} \quad \sum_{h\in\mathcal{H}} p_h^w[t]w_{ih}[t]+p_h^s[t]s_{ih}[t] \tag{3.22}$$

$$\text{subject to} \quad \alpha_i^r w_{ih}[t]+\beta_i^r s_{ih}[t]\le\hat{C}_h^r[t],\ \forall h,r, \tag{3.23}$$

$$\textstyle\sum_{h\in\mathcal{H}} w_{ih}[t]\ge m_w[t], \tag{3.24}$$

$$\textstyle\sum_{h\in\mathcal{H}} s_{ih}[t]\ge m_p[t], \tag{3.25}$$

where $p_h^w[t] \triangleq \sum_{r\in\mathcal{R}} p_h^r[t]\alpha_i^r$ and $p_h^s[t] \triangleq \sum_{r\in\mathcal{R}} p_h^r[t]\beta_i^r$ denote the combined prices of all resources of allocating worker and parameter server on machine h in time t, respectively.

However, Problem (3.22) is an integer programming problem, it is a problem with generalized packing and cover type constraints (i.e., integer variables rather than 0-1 variables) in (3.23) and (3.24)-(3.25), respectively, which is clearly NP-Hard. Also, it is well-known that there are *no* polynomial time approximation schemes (PTAS) even for the basic set-cover and bin-packing problems unless P = NP Hochbaum (1997). Hence, we will pursue a constant ratio approximation scheme to solve Problem (3.22) in this paper.

To this end, we propose a randomized rounding scheme to solve the new relaxed problem: First, we solve the linear programming relaxation of Problem (3.22). Let $\{\bar{w}_{ih}[t], \bar{s}_{ih}[t], \forall h, t\}$ be the fractional optimal solution. Let $G > 1$ be a constant and let $w'_{ih}[t] = G\bar{w}_{ih}[t], s'_{ih}[t] = G\bar{s}_{ih}[t]$, $\forall h, t$. Then, we randomly round $\{w'_{ih}[t], s'_{ih}[t], \forall h, t\}$ to generate an integer solution:

$$w_{ih}[t] = \begin{cases} \lceil w'_{ih}[t] \rceil, \text{ with probability } w'_{ih}[t] - \lfloor w'_{ih}[t] \rfloor, \\ \lfloor w'_{ih}[t] \rfloor, \text{ with probability } \lceil w'_{ih}[t] \rceil - w'_{ih}[t], \end{cases} \tag{3.26}$$

$$s_{ih}[t] = \begin{cases} \lceil s'_{ih}[t] \rceil, \text{ with probability } s'_{ih}[t] - \lfloor s'_{ih}[t] \rfloor, \\ \lfloor s'_{ih}[t] \rfloor, \text{ with probability } \lceil s'_{ih}[t] \rceil - s'_{ih}[t]. \end{cases} \tag{3.27}$$

We will later prove in Theorem 2 that the approximation ratio of this randomized rounding scheme in (3.26)-(3.27) enjoys a ratio that *logarithmically* depends on the problem size.

Finally, summarizing the results in 2-1) and 2-2) yields the following approximation algorithm for solving Problem (3.18):

Algorithm 4: Solving $\theta(t, v)$ (i.e., Problem (3.18)).

Initialization:

1. Let $w_{ih}[t] = s_{ih}[t] = 0, \forall h$. Let $h = 1$. Pick some $G > 1$.

   Let $D = \lceil vM_i(\tau_i + 2g_i/b_i^{(i)}) \rceil$. $h^* = \varnothing$. *cost-min*$= \infty$.

Handling Internal Communication:

2. If constraint (3.23) is not satisfied, go to Step 7.

3. If $w_{ih}[t] < D$, $w_{ih}[t] \leftarrow w_{ih}[t] + 1$ and go to Step 2.

4. If $w_{ih}[t] > K_i$, go to Step 7.

5. If $w_{ih}[t]b_i^{(i)} > s_{ih}[t]B_i$, $s_{ih}[t] \leftarrow s_{ih}[t]+1$ and go to Step 2.

6. If *cost-min* $> p_h^r[t](\alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}^r[t])$, then let *cost-min*$= p_h^r[t](\alpha_i^r w_{ih}[t] + \beta_i^r s_{ih}^r[t])$ and $h^* = h$.

7. Let $h \leftarrow h + 1$. If $h > H$, stop; otherwise, go to Step 1.

Handling External Communication:

8. Solve the linear programming relaxation of Problem (3.22) with $m_w[t] = \lceil vM_i\tau_i \rceil$, $m_p[t] = \lceil m_w[t]b_i^{(e)}/B_i \rceil$. Let $\{\bar{w}_{ih}[t], \bar{s}_{ih}[t], \forall h, t\}$ be the fractional optimal solution.

9. Let $w'_{ih}[t] = G\bar{w}_{ih}[t]$, $s'_{ih}[t] = G\bar{s}_{ih}[t]$, $\forall h, t$.

10. Generate an integer solution $\{w_{ih}[t], s_{ih}[t], \forall h, t\}$ following the randomized rounding scheme in (3.26)-(3.27).

11. If $\{w_{ih}[t], s_{ih}[t], \forall h, t\}$ is infeasible, go to Step 10.

Final Step:

12. Compare the solutions between internal and external case. Pick the cheaper one between them and return the cost and the corresponding schedule $\{w_{ih}[t], s_{ih}[t], \forall h, t\}$.

In the internal communication part of Algorithm 4, we check each machine one by one. If the resource capacity constraint (3.23) is satisfied (Step 2), we increase workers to satisfy the learning workload demand D (Step 3) and also increase parameter servers accordingly (Step 5). If we detect a machine with lower cost, we update the cost and schedule accordingly (Step 6). After exploring one machine, we move on to the next (Step 7). The external communication part is based on LP relaxation (Step 8), randomized rounding (Step 9-11) and heuristic search (Step 12).

3.2.4 Performance Analysis

We now examine the competitive ratio performance of our PD-ORS algorithm. Note that the key component in PD-ORS is our proposed randomized rounding scheme in (3.26)-(3.27), which

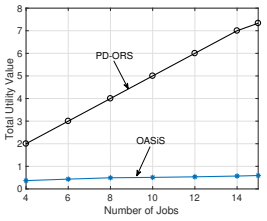

Figure 3.11: Total utility comparison between PD-ORS and OASiSBao et al. (2018).

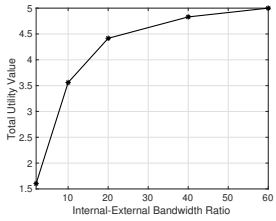

Figure 3.12: The impact of internal-external bandwidth ratio on total utility.

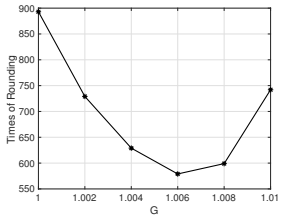

Figure 3.13: The impact of pre-rounding gain factor G on rounding efficiency.

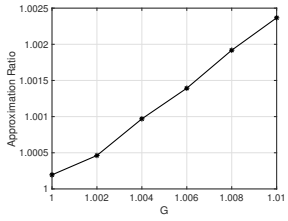

Figure 3.14: The impact of pre-rounding gain factor G on approximation ratio.

is in turn the foundation of Algorithm 1. Thus, we first prove the following result about the randomized rounding:

Consider an integer program with generalized cover/packing constraints: $\min\{\mathbf{c}^\top\mathbf{x} : \mathbf{A}\mathbf{x} \geq \mathbf{a}, \mathbf{B}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}_+^n\}$, where $\mathbf{A} \in \mathbb{R}_+^{m\times n}$, $\mathbf{B} \in \mathbb{R}_+^{r\times n}$, $\mathbf{a} \in \mathbb{R}_+^m$, $\mathbf{b} \in \mathbb{R}_+^r$, and $\mathbf{c} \in \mathbb{R}_+^n$. Let $\bar{\mathbf{x}}$ be a fractional optimal solution. Consider the randomized rounding scheme: Let $\mathbf{x}' = G\bar{\mathbf{x}}$ for some $G > 1$ (to be specified). Randomly round $\mathbf{x}'$ to $\hat{\mathbf{x}} \in \mathbb{Z}_+^n$ as: $\hat{x}_j = \lceil x'_j \rceil$ w.p. $x'_j - \lfloor x'_j \rfloor$ and $\hat{x}_j = \lfloor x'_j \rfloor$ o.w. Then, we have (see proof in Appendix A):

Lemma 1 (Rounding) *Let* $W_a \triangleq \min\{a_i/[\mathbf{A}]_{ij} : [\mathbf{A}]_{ij} > 0\}$ *and* $W_b \triangleq \min\{b_i/[\mathbf{B}]_{ij} : [\mathbf{B}]_{ij} > 0\}$. *Let* $\delta \in (0, 1]$ *and define* G *as:*

$$G \triangleq 1 + \frac{\ln(3m/\delta)}{W_a} + \sqrt{\left(\frac{\ln(3m/\delta)}{W_a}\right)^2 + \frac{2\ln(3m/\delta)}{W_a}}.$$

Then, with probability greater than $1 - \delta$, $\hat{\mathbf{x}}$ *achieves a cost at most* $\frac{3G}{\delta}$ *times the cost of* $\bar{\mathbf{x}}$, *and* $\hat{\mathbf{x}}$ *satisfies* $\Pr\{(\mathbf{B}\hat{\mathbf{x}})_i > b_i(1 + (\frac{3}{GW_b})^{\frac{1}{2}})G, \exists i\} \leq \frac{\delta}{3r}$.

Several important remarks for Lemma 1 are in order: i) The theoretical approximation ratio $\frac{3G}{\delta}$ is conservative. Our numerical studies show that the approximation ratio performance in reality is much smaller than $\frac{3G}{\delta}$; ii) The probability parameter δ controls the trade-off between approximation ratio and efficiency in finding a feasible rounding solution: A larger δ implies a smaller approximation ratio, but the probability of obtaining a feasible solution of this ratio is also smaller (i.e., more rounds of rounding needed). Interestingly, for $\delta = 1$, Lemma 1 indicates that there is still *non-zero* probability to achieve an approximation ratio not exceeding $3G$; iii) The probabilistic

guarantee of the packing constraint ($\mathbf{Bx} \leq \mathbf{b}$) is unavoidable and due to the fundamental hardness of the conflicting cover and packing constraints: A strategy trying to better satisfy the cover constraints (multiplying a G-factor in here) may increase the probability of violating the packing constraints; iv) The result in Lemma 1 is for general ILP with mixed cover/packing constraints, which could be of independent theoretical interest.

By specializing Lemma 1 with parameters in Problem (3.22), we have the following approximation result for Algorithm 4:

Theorem 2 (Algorithm 4) *Let* $W_1 \triangleq \min\{m_w[t], m_p[t]\}$, $W_2 \triangleq \min\{\hat{C}_h^r[t]/\alpha_i^r, \hat{C}_h^r[t]/\beta_i^r, \forall r, h\}$. *Let* $\delta \in (0, 1]$. *Define* G *as:*

$$G \triangleq 1 + \frac{\ln(6/\delta)}{W_1} + \sqrt{\left(\frac{\ln(6/\delta)}{W_1}\right)^2 + \frac{2\ln(6/\delta)}{W_1}}.$$

Then, with probability greater than $1-\delta$, *Algorithm 4 obtains a schedule* $\{w_{ih}[t], s_{ih}[t], \forall t, h\}$ *that has an approximation ratio at most* $\frac{3G}{\delta}$ *with* $\Pr\{LHS(3.23) > \hat{C}_h^r[t]G(1+(\frac{3}{GW_2})^{\frac{1}{2}})\} \leq \frac{\delta}{3HR}$.

Theorem 2 is a direct consequence of Lemma 1 and we omit the proof for brevity. With Theorem 2, we can establish the overall competitive ratio result for Algorithm 1 as follows:

Theorem 3 (Competitive Ratio) *Let* G *and* δ *be as defined in Theorem 2. Let* U^r *and* L *be as defined in (3.13) and (3.14), respectively. Then, PD-ORS in Algorithm 1 is* $\frac{6G}{\delta}\min_{r\in\mathcal{R}}(1, \ln\frac{U^r}{L})$*–competitive.*

Theorem 3 can be proved by weak duality and the approximation result in Theorem 2. We provide a proof in Appendix B. Finally, by combining Algorithms 1-5, it can be shown that the average time complexity of PD-ORS is $O(\frac{1}{\delta}TK_i^2E_i^2H^4)$, which is polynomial.

Table 3.1: List of notations.

$\mathcal{I}$	The set of jobs
$\mathcal{T}$	System timespan
$\tilde{t}_i$	Completion time of job i
a_i	Arrival time of job i
$\mathcal{R}$	The set of resource types
K_i	Number of data chunks in i
x_i	Admission decision variable to accept job i or not
$u_i(\cdot)$	Job i's utility function
E_i	Number of training epochs for job i
M_i	Number of mini-batches in a data chunk for job i
$\mathcal{H}$	Set of physical machines
C_h^r	Capacity of type-r resource on server h
α_i^r	Type-r resource required by a worker in job i
β_i^r	Type-r resource required by a parameter server in job i
$w_{ih}[t]$	Number of workers of job i on server h in t
$s_{ih}[t]$	Number of parameter servers of job i on server h in t
B_i	Bandwidth offered by a parameter server of job i;
$b_i(h,p)$	Bandwidth consumed by a worker of job i, where $b_i(h,p) = b_i^{\{e\}}$, if $h \neq p$ or $b_i^{\{i\}}$, otherwise.
τ_i	Time to train a mini-batch for job i
g_i	Size of gradients and parameters for job i
$\mathcal{W}_i[t]$	Set of physical machines containing workers for job i in t
$\mathcal{P}_i[t]$	Machines containing parameter servers for job i in t
x_i^π	Binary decision variable to select schedule π for job i or not
t_i^π	The completion time slot of job i with schedule π
$w_{ih}^\pi[t]$	Number of workers on server h in t for job i in schedule π
$s_{ih}^\pi[t]$	Number of parameter servers on server h for schedule π in t
Π_i	Set of all feasible schedules for job i

CHAPTER 4. RESULTS

4.1 Numerical Results

In this section, we conduct simulation studies to evaluate the efficacy of our proposed PD-ORS algorithm. In our experiments, we test an ML system with $E_i \in [50, 150]$, $M_i \in [20, 40]$, $K_i \in [5, 20]$, $g_i \in [50, 100]$, $\tau_i \in [0.01, 0.05]$, $b_i \in [100\text{Mbps}, 4\text{Gbps}]$, and $B_i = [4\text{Gbps}, 20\text{Gbps}]$, all generated uniformly at random. We consider three types of resources: GPU, memory, and storage. In our experiments, both worker and parameter servers requested 0 to 4 GPU, 2 to 30 GB memory, and 4 to 8 GB storage.

We first compare our PD-ORS algorithm with the OASiS algorithm inBao et al. (2018), which is the most related work to ours and the state-of-the-art of scheduling for distributed ML systems. As mentioned earlier, the key difference in OASiS is that parameter servers and workers are located on two strictly separated set of machines. Here, we let $H = 30$ and $T = 100$. For OASiS, half of the machines are parameter servers and the other half are workers. We set $b_i^{(i)}/b_i^{(e)} = 40$. For fair comparisons, both algorithms adopt the same utility function: $u_i(t - a_i) = \frac{1}{1+(t-a_i)}$. The comparison results are shown in Fig. 3.11. We can see that PD-ORS significantly outperforms OASiS. For example, with 15 jobs, PD-ORS's utility value is more than 7 times higher than that of OASiS.

Next, we investigate the impact of $b_i^{(i)}/b_i^{(e)}$ on total utility value and the results are shown in Fig. 3.12. In this experiment, we let $H = 20$, $T = 50$, and vary $b_i^{(i)}/b_i^{(e)}$ from 1 to 60. We can see that the total utility rises rapidly as $b_i^{(i)}/b_i^{(e)}$ increases initially, which shows that PD-ORS reacts aggressively to a large $b_i^{(i)}/b_i^{(e)}$-value. On the other hand, the increase of total utility becomes more gradual when $b_i^{(i)}/b_i^{(e)}$ is greater than 30. This is because beyond this point, most jobs already have a large number of co-located parameter servers and workers.

Lastly, we examine the performance of the randomized rounding scheme in Algorithm 4, which is the key of PD-ORS. We evaluate the efficiency of rounding in terms of how many times of rounding are needed to obtain a feasible solution and their according approximation ratio. The results are shown in Figs. 3.13 and 3.14, respectively. In this experiment, we let $H = 50$ and $T = 100$, which implies the total number of possible rounding choices is 2^{50}, an astronomical number. We vary the pre-rounding gain factor G from 1 to 1.01. We can see that the times of rounding initially decreases and reaches the lowest point at 570 when $G = 1.006$. This is because cover constraints are easier to satisfy with a larger G. When $G > 1.006$, we can see that times of rounding start to increase. This is because packing constraints are prone to be violated as G gets large. We note that, in all cases, the times of rounding are less than 1000, for which the runtime is short (especially given the size of the search space is 2^{50}). Also, we can see from Fig. 3.14 that the approximation ratios for all choices of G are close to 1 (≤ 1.0025), which shows that our randomized rounding scheme is not only much tighter than than the worse case bound suggested Theorem 3, it is also near-optimal.

CHAPTER 5. SUMMARY AND DISCUSSION

This is the opening paragraph to my thesis which explains in general terms the concepts and hypothesis which will be used in my thesis.

With more general information given here than really necessary.

5.1 Conclusion

In this paper, we investigated the problem of online resource scheduling design for large-scale distributed machine learning systems over computing clusters. We considered the most general setting where workers and parameter servers can be co-located on the same physical machine. We showed that this problem is a challenging integer nonlinear programming problem with *non-deterministic* constraints. In this paper, we developed an online scheduling algorithm with competitive ratio guarantee. Our main contributions are three-fold: i) We developed a new analytical model that jointly considers resource locality and allocation; ii) Through careful examinations of worker-server configuration relationships, we resolve the locality ambiguity in the model and reduce the problem to a mixed cover/packing integer program that leads to low-complexity approximation algorithm design; iii) We proposed a meticulously designed randomized rounding algorithm to solve the mixed cover/packing integer program and rigorously established its approximation ratio guarantee. Collectively, our results expand the theoretical frontier of optimization algorithm design for distributed machine learning systems.

BIBLIOGRAPHY

Abadi, M., Barham, P., et al. (2016). TensorFlow: A system for large-scale machine learning. In *Proc. of USENIX OSDI*.

Bao, Y., Peng, Y., Wu, C., and Li, Z. (2018). Online job scheduling in distributed machine learning clusters. In *Proc. of IEEE INFOCOM*.

Bazaraa, M. S., Sherali, H. D., and Shetty, C. M. (2006). *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons Inc., New York, NY, 3 edition.

Buchbinder, N. and (Seffi) Naor, J. (2009). *The Design of Competitive Online Algorithms via a Primal-Dual Approach*, volume 3. Now Publishers Inc.

Chen, L. and Liu, S. (2017). Scheduling jobs across geo-distributed datacenters with max-min fairness. In *Proc. of IEEE INFOCOM*.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2016). MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. In *Proc. of NIPS Workshop on Machine Learning Systems (LearningSys)*.

Chilimbi, T. M., Suzue, Y., Apacible, J., and Kalyanaraman, K. (2014). Project Adam: Building an efficient and scalable deep learning training system. In *Proc. of USENIX OSDI*.

Chun, B.-G., Cho, B., Jeon, B., et al. (2016). Dolphin: Runtime optimization for distributed machine learning. In *Proc. of ICML ML Systems Workshop*.

Hochbaum, D. S. (1997). *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company.

Huang, Z., Balasubramanian, B., Wang, M., Lan, T., Chiang, M., and Tsang, D. H. K. (2015). Need for speed: CORA scheduler for optimizing completion-times in the cloud. In *Proc. of IEEE INFOCOM*.

Huo, Z. and Huang, H. (2017). Asynchronous mini-batch gradient descent with variance reduction for non-convex optimization. In *AAAI*, pages 2043–2049.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proc. of the 22nd ACM International Conference on Multimedia*, pages 675–678.

Li, M., Andersen, D. G., et al. (2014). Scaling distributed machine learning with the parameter server. In *Proc. of USENIX OSDI*.

Lian, X., Huang, Y., Li, Y., and Liu, J. (2015). Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745.

Peng, Y., Bao, Y., Chen, Y., Wu, C., and Guo, C. (2018). Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proc. of ACM EuroSys*.

Sun, P., Wen, Y., Ta, N. B. D., and Yan, S. (2017). Towards distributed machine learning in shared clusters: A dynamically-partitioned approach. In *Proc. of IEEE Smart Computing*.

Yan, F., Ruwase, O., He, Y., and Chilimbi, T. (2015). Performance modeling and scalability optimization of distributed deep learning systems. In *Proc. of ACM KDD*.

APPENDIX A. PROOF OF LEMMA 1

To prove Lemma 1, consider the probabilities of the following "bad" events: 1) $\mathbf{c}^\top \mathbf{x} > \frac{3G}{\delta}\mathbf{c}^\top \bar{\mathbf{x}}$; 2) $\exists i$ such that $(\mathbf{A}\hat{x})_i < a_i$; and 3) $\exists i$ such that $(\mathbf{B}\hat{\mathbf{x}})_i > b_i$. Note that events 2) and 3) can be equivalently rewritten as: 2') $\exists i$ such that $\mathbb{E}\{(\mathbf{A}\mathbf{x})_i \frac{W_a}{a_i} < W_a\}$ and 3') $\exists i$ such that $\mathbb{E}\{(\mathbf{B}\mathbf{x})_i \frac{W_b}{b_i} < W_b\}$. Since $\mathbb{E}\{\hat{\mathbf{x}}\} = \mathbf{x}' = G\bar{\mathbf{x}}$, by linearity of expectation, we have:

$$\mathbb{E}\{\mathbf{c}^\top \hat{\mathbf{x}}\} = \mathbf{c}^\top \mathbb{E}\{\hat{\mathbf{x}}\} = \mathbf{c}^\top G\bar{\mathbf{x}} = G\mathbf{c}^\top \bar{\mathbf{x}}, \tag{A.1}$$

$$\mathbb{E}\left\{(\mathbf{A}\hat{\mathbf{x}})_i \frac{W_a}{a_i}\right\} = G\mathbb{E}\left\{(\mathbf{A}\bar{\mathbf{x}})_i \frac{W_a}{a_i}\right\} \geq GW_a, \tag{A.2}$$

$$\mathbb{E}\left\{(\mathbf{B}\hat{\mathbf{x}})_i \frac{W_b}{b_i}\right\} = G\mathbb{E}\left\{(\mathbf{B}\bar{\mathbf{x}})_i \frac{W_b}{b_i}\right\} \leq GW_b. \tag{A.3}$$

Then, by Markov inequality and (A.1), we have $\Pr\left\{\mathbf{c}^\top \hat{\mathbf{x}} > \frac{3G}{\delta}\mathbf{c}^\top \bar{\mathbf{x}}\right\} \leq \frac{\delta}{3}$.

Next, we note that each $\hat{x}_j$ can be viewed as a sum of independent random variables in $[0, 1]$ as follows: The fixed part of $\lfloor x'_j \rfloor$ is a sum of $\lfloor x'_j \rfloor$ random variables with value 1 with probability 1. Then, we have that $(\mathbf{A}\hat{\mathbf{x}})_i \frac{W_a}{a_i} = (\sum_j [\mathbf{A}]_{ij}\hat{x}_j)\frac{W_a}{a_i}$ is also a sum of independent random variables in $[0, 1]$. Using Chernoff bound, we have $\Pr\{(\mathbf{A}\hat{\mathbf{x}})_i \frac{W_a}{a_i} \leq (1-\epsilon)GW_a\} \leq \exp(-\epsilon^2 \frac{GW_a}{2})$. Setting $(1-\epsilon)G = 1$, i.e., $\epsilon = 1 - \frac{1}{G}$, we have:

$$\Pr\left\{(\mathbf{A}\hat{\mathbf{x}})_i \frac{W_a}{a_i} \leq W_a\right\} \leq \exp\left(-\left(1 - \frac{1}{G}\right)^2 G\frac{W_a}{2}\right). \tag{A.4}$$

Forcing $\exp\left(-(1-\frac{1}{G})^2 G\frac{W_a}{2}\right) \leq \frac{\delta}{m}$ and solving G, we have:

$$G \triangleq 1 + \frac{\ln(3m/\delta)}{W_a} + \sqrt{\left(\frac{\ln(3m/\delta)}{W_a}\right)^2 + \frac{2\ln(3m/\delta)}{W_a}}. \tag{A.5}$$

Using (A.3) and Chernoff bound and following similar arguments, we have: $\Pr\{(\mathbf{B}\hat{\mathbf{x}})_i \frac{W_b}{b_i} > (1+\epsilon)GW_b\} \leq \exp(-\epsilon^2 \frac{GW_b}{3})$. Forcing $\exp\left(-\epsilon^2 \frac{GW_b}{3}\right) = \frac{\delta}{3r}$ and solving for ϵ, we have $\epsilon = (\frac{3}{GW_b}\ln(\frac{3r}{\delta}))^{\frac{1}{2}}$. It follows that $\Pr\{(\mathbf{B}\hat{\mathbf{x}})_i \frac{W_b}{b_i} > (1 + (\frac{3}{GW_b}\ln(\frac{3r}{\delta}))^{\frac{1}{2}})\} \leq \frac{\delta}{3r}$, which implies that:

$$\Pr\left\{(\mathbf{B}\hat{\mathbf{x}})_i > \left(1 + \sqrt{\frac{3}{GW_b}\ln\left(\frac{3r}{\delta}\right)}\right) G, \exists i\right\} \leq \frac{\delta}{3r}. \tag{A.6}$$

By using union bound and (A.4) and (A.6), we have that events 1)–3) occur with probability less than $\frac{\delta}{3} + m \cdot \frac{\delta}{3m} + r \cdot \frac{\delta}{3r} = \delta$, and the proof is complete.

APPENDIX B. PROOF SKETCH OF THEOREM 3

Due to space limitation, we provide a proof sketch for Theorem 3 in here. From Theorem 2, we know that Algorithm 4 is a $\frac{3G}{\delta}$-approximate algorithm. Then, by induction, we can show that the dynamic programming approach in Algorithm 3 is also a $\frac{3G}{\delta}$-approximate algorithm.

Now, let $\hat{\pi}_i$ denote the approximate schedule obtained by Algorithm 2, which inexactly solves Problem D-R-DMLRS. Let P_i and D_i be the primal and dual objective values of Problems R-DMLRS and D-RMLRS after determining the schedule $\hat{\pi}_i$ in Algorithm 1. Then, we have $P_i - P_{i-1} = u_i(\tilde{t}_{\hat{\pi}_i} - a_i)$. Let $\hat{\lambda}_i$ be the dual price that corresponds to $\hat{\pi}_i$. Also, we let $\Theta^*(\tilde{t}_{\hat{\pi}_i}, V_i)$ be the true minimum cost under $\tilde{t}_{\hat{\pi}_i}$. Suppose that job i is admitted, we then have $u_i(\tilde{t}_{\hat{\pi}_i} - a_i) - \frac{\delta}{3G}\Theta(\tilde{t}_{\hat{\pi}_i}, V_i) \geq u_i(\tilde{t}_{\hat{\pi}_i} - a_i) - \Theta^*(\tilde{t}_{\hat{\pi}_i}, V_i) \geq u_i(\tilde{t}_{\hat{\pi}_i} - a_i) - \Theta(\tilde{t}_{\hat{\pi}_i}, V_i) = \hat{\lambda}_i$, which implies $u_i(\tilde{t}_{\hat{\pi}_i} - a_i) \geq \hat{\lambda}_i + \frac{\delta}{3G}\Theta(\tilde{t}_{\hat{\pi}_i}, V_i)$.

Also, note that $D_i - D_{i-1} = \hat{\lambda}_i + \sum_t \sum_h \sum_r (p_h^{r,i}[t] - p_h^{r,i-1}[t])C_h^r$. Then, following similar arguments in Buchbinder and (Seffi) Naor (2009); Bao et al. (2018), we can show that the following allocation-cost relationship holds under price function (3.12): $\Theta(\tilde{t}_{\hat{\pi}_i}, V_i) \geq \frac{\delta/3G}{\min_{r\in\mathcal{R}}(1, \ln\frac{U^r}{L})} \sum_t \sum_h \sum_r (p_h^{r,i}[t] - p_h^{r,i-1}[t])C_h^r$, which further implies $P_i - P_{i-1} \geq \frac{\delta/3G}{\min_{r\in\mathcal{R}}(1, \ln\frac{U^r}{L})}(D_i - D_{i-1})$. Then, by telescoping on $P_i - P_{i-1}$ from 1 to I and using weak duality with the fact that $D_0 \leq \frac{1}{2}OPT$ (due to the choice of constant L in the cost function), we reach the final competitive ratio in Theorem 3 and the proof is complete.