Contention-Free All-Reduce Learning Jobs Scheduling in Multi-Tenant GPU Computing Clusters

Menglu Yu[1] and Jia Liu[2,1]
[1]Department of Computer Science, Iowa State University
[2]Department of Electrical and Computer Engineering, The Ohio State University

ABSTRACT

Recent years have witnessed the large-scale adoption of distributed deep learning (DDL) over multiple GPUs to increase the training efficiency by leveraging the massive parallelism in computing clusters. Also, to sustain the rapid growing demand for DDL training, all-reduce collective communication operations have become the common practice in the industry to address the communication bottleneck limitations of the traditional server-worker architecture. However, simultaneously training multiple DDL jobs in a cluster with all-reduce could lead to communication contention, which, if not treated appropriately, could significantly diminish the benefits of all-reduce. So far in the literature, the contention problems of all-reduced-based training have not been considered. This motivates us to fill this gap in this paper by proposing a new DDL training job scheduler that considers both job placement and task scheduling to avoid contention among DDL jobs. Our main contributions are three-fold: i) We propose a contention-free DDL job scheduling framework to minimize the makespan for all-reduce-based DDL jobs; ii) To address the NP-Hardness and unique structural challenges in this problem, we transform the problem into an equivalent form to enable search-based approximation algorithm design; iii) By decomposing the problem into separate job placement and communication task scheduling subproblems, we propose a meticulously designed approximation algorithm with provable performance guarantee. Extensive experiments verify the theoretical performance results of our proposed algorithm and demonstrate the superiority of our algorithm over the baselines.

CCS CONCEPTS

• **Computing methodologies → Distributed algorithms**; • **Networks → Network performance analysis**.

KEYWORDS

GPU computing clusters, all-reduce communication, resource scheduling, approximation algorithm design

ACM Reference Format:
Menglu Yu[1] and Jia Liu[2,1] . 2022. Contention-Free All-Reduce Learning Jobs Scheduling in Multi-Tenant GPU Computing Clusters. In *The Twenty-third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '22), October 17–20, 2022, Seoul, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3492866.3549716

1 INTRODUCTION

In recent years, deep learning (DL) has achieved an astonishing success in many areas and spawned applications spanning computer vision, natural language processing, recommender systems, healthcare, automotive, retail, smart homes, to name just a few. However, the rise of DL-based applications has also generated an influx of resource-intensive computing jobs for training deep neural networks (DNNs). Due to the ever-increasing model complexity and dataset size, distributed deep learning (DDL) [4] has been widely adopted to leverage both data and model parallelisms in computing clusters to expedite the training process. This is evidenced by various popular DDL frameworks (e.g., TensorFlow [1] and PyTorch [11]). Also, to further improve the communication performance in DDL, such DDL frameworks have evolved from the traditional server-worker (SW) architecture [10] to the so-called *all-reduce collective communication operations* [16], which removes the dedicated parameter server to eliminate the communication bottleneck limitation of SW. In the industry, two of the most popular all-reduce mechanisms are the ring-all-reduce (RAR) [12] and tree-all-reduce (TAR) [14] (see Section 3.2 for further details).

Simply speaking, the basic idea of all-reduce mechanisms is to allow workers to collaboratively exchange and aggregate information without any coordination from a server. As a result, the information exchanges among workers are more "evenly spread" compared to SW, hence eliminating communication bottleneck. However, the bottleneck-free benefit of all-reduce is *not* achieved without costs: in a networking environment with multiple DDL jobs, the more-evenly-spread data exchanges usually imply more extensive communication contention among these jobs. If not treated appropriately, such communication contention could significantly diminish the benefits of all-reduce and result in prolonged job completion time. For example, it is observed in [17] that, in a four-GPU cluster connected by 10 Gbps Ethernet, when four same DDL jobs are scheduled and executed concurrently with RAR, the completion time per job is 675 seconds. This is more than twice as long as the training time of 295 seconds when only one DDL job is trained without contention. This significant training performance degradation highlights the compelling need for efficient and effective contention-aware scheduler for all-reduced-based DDL training.

So far, however, research on contention-aware scheduling algorithm design for all-reduce-based DDL jobs remains under-explored and results in this area are rather limited. This is in large part due to several technical challenges arising from resolving contentions among all-reduce-based DDL jobs. First, due to the contention between the jobs, the completion time of each job is influenced by other concurrent jobs. Due to this complex coupling, deriving a closed-form expression to evaluate each job's completion time is intractable. Second, there exists a fundamental trade-off between communication speed and resource utilization in all-reduce-based worker placement decisions. On one hand, while allocating all workers of a DDL job to the same server enjoys a much higher intra-server communication rate between workers, this approach could result in resource fragmentation. On the other hand, even though having a better utilization of dispersed GPU resources, allocating workers of a DDL job over multiple servers could lead to extensive communication contention and overhead for establishing connections between servers. Moreover, the more sophisticated contention relationship in all-reduce mechanisms, coupled with the alternating and heterogeneous computation-communication patterns among different DDL jobs, adds another layer of challenges in scheduling algorithm design. Last but not least, as in most resource scheduling problems, all-reduce-based DDL scheduling also contains a mix of covering-type (to satisfy training workload requirement) and packing-type (due to the resource capacity of each server) constraints, which renders the problem NP-hard.

In this paper, we overcome the above challenges and develop a suite of algorithmic techniques for efficient scheduling of all-reduce-based DDL training in multi-tenant GPU clusters. Our main results and technical contributions are summarized as follows:

- We first propose an analytical model for all-reduce-based DDL training, which jointly considers job placement and task scheduling with contention avoidance. To address the intractability challenge of job completion time evaluation due to contention, we transform the problem into an equivalent formulation that trades dimensionality for structural simplicity to enable provable approximation algorithm design.
- Based on the analytical modeling above, we develop an efficient scheduling algorithm by dividing the reformulated problem into two stages: job placement and task scheduling. In the job placement stage, our algorithm balances the trade-off between communication contention and overhead. In the task scheduling stage, we consider both preemptive and non-preemptive settings and show that, given any job placement, our task scheduling strategy achieves a 2-approximation ratio in the non-preemptive setting and is *optimal* in the preemptive setting, respectively.
- We conduct extensive experiments to show the superiority of our proposed algorithm over baseline scheduling policies in the literature. We also test our scheduling algorithms under both non-preemptive and preemptive settings to verify the theoretical performance guarantee results.

Collectively, our results contribute to a comprehensive and fundamental understanding of all-reduce-based DDL scheduler design under communication contention. The remainder of this paper is organized as follows. In Sec. 2, we review the literature to put our work in comparative perspectives. Sec. 3 presents the necessary background and familiarize readers with terminologies. Sec. 4 introduces the system model and problem formulation. Sec. 5 presents our algorithms and Sec. 6 analyzes their performance. Sec. 7 presents numerical results and Sec. 8 concludes this paper.

2 RELATED WORK

As mentioned in Section 1, DDL training job scheduling in GPU/CPU clusters have attracted a great amount of interest in recent years. Early attempts in this area include several heuristic engineering designs, such as Gandiva [18], a resource manager to improve GPU time-slicing and job placement by exploiting the performance predictability of DDL training jobs. Similarly, Optimus [13] leveraged online-fit performance model to adjust the job placement to achieve the minimum job completion time. In contrast to the above heuristic approaches, another line of research is to leverage learning-based methods for resource scheduling and job placement. For example, Harmony [2], a deep-learning-based ML cluster scheduler, aimed to reduce the interference among jobs and to minimize training completion time. A scheduling framework called Spear is proposed in [7] to minimize the makespan of DDL jobs by jointly considering the task dependencies and heterogeneous resource demands. However, none of these works considered all-reduce-based DDL jobs scheduling and no theoretical performance guarantee was provided.

To our knowledge, results on all-reduced-based DLL job scheduling are very limited. For example, the work in [17] utilized a system-dependent online-fit model to predict the execution time of each all-reduce-based DDL job. However, the authors did not formulate any optimization scheduling problem, and the proposed heuristic-based algorithms had no performance guarantee. In contrast, the GADGET algorithm in [20] characterized RAR-based job scheduling as an optimization problem, where they provided theoretical analysis on the proposed algorithms based on the assumption that each job had a dedicated bandwidth. Hence, no communication contention was considered.

The most related work to ours is [19], where the authors relaxed the bandwidth reservation of RAR in [20] to avoid channel underutilization and developed a scheduler for multiple RAR-based DDL jobs competing for shared network resources. However, they only developed a general framework on job placement without addressing how to resolve communication contention. This work differs from [19] in the following aspects: i) Our scheduler is applicable for all types of all-reduce operations beyond just RAR. Also, compared to [19], our scheduler is developed for the task-level, which is of finer granularity and enables better task control without contention; ii) Our task scheduling algorithm is low-complexity and works for both non-preemptive and preemptive task scheduling and provide theoretical performance guarantees. In particular, the preemptive scheduling has been shown to be optimal given a job placement.

3 PRELIMINARIES

In this section, we give a brief overview of DDL training and all-reduce communication patterns to familiarize readers with the necessary background and facilitate later discussions.

3.1 Distributed Deep Learning (DDL) Training

Training a DNN amounts to solving an optimization problem through a process of iteratively adjusting the DNN model parameters until a convergence criterion or a limit of iterations is reached. The goal of the optimization is to minimize a loss function $f(\mathbf{W}, D)$, where $\mathbf{W}$ denotes the model weight parameters and D represents the training data. We let Ω be the set of workers for the training job, and each

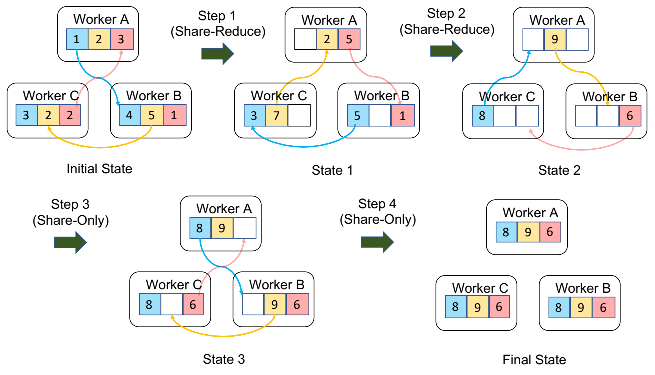

Figure 1: A 3-worker ring-all-reduce example.

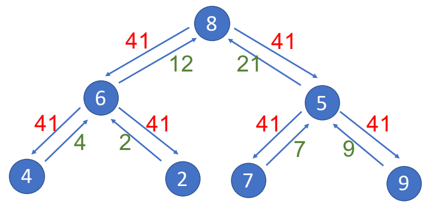

Figure 2: A binary tree-all-reduce example.

Table 1: Performance summary of TAR and RAR.

All-reduce Algorithm	TAR	RAR
Existence of bottleneck	No	No
Number of rounds	$2\lceil\log_2 w\rceil$	$2(w-1)$
Per-round latency	$\frac{d}{r}$	$\frac{d}{w}r$
Overall latency	$\frac{2\lceil\log_2 w\rceil d}{r}$	$\frac{2(w-1)d}{wr}$

worker $\omega \in \Omega$ has a local dataset D_ω with size $|D|/|\Omega|$. In the i-th training iteration, each worker ω fetches a mini-batch D^i_ω from D_ω to first compute the loss $f(\mathbf{W}^i, D^i_\omega)$ from the input to the output layers, which is often referred to as the forward pass (FP). The FP loss will be then used to compute a stochastic gradient $\nabla f(\mathbf{W}^i, D^i_\omega)$ in the reverse order, which is termed the backward pass (BP). Then, the aggregated average gradients $\sum_{\omega\in\Omega} \nabla f(\mathbf{W}^i, D^i_\omega)$ from all workers (also called "reduced gradient") is computed and distributed to all the workers, which can be done using *all-reduce collective operators* (see Section 3.2). On the worker side, once the aggregated result is received at a worker, the worker updates its model parameters (e.g., using the SGD-type update $\mathbf{W}^{i+1} = \mathbf{W}^i - \frac{s}{|\Omega|}\sum_{\omega\in\Omega}\nabla f(\mathbf{W}^i, D^i_\omega)$, where $s > 0$ is the learning rate).

3.2 All-Reduce Collective Opterators

It is clear from the DDL process described above, computing the reduction of stochastic gradients from all workers is communication-intensive. As mentioned in Section 1, in the traditional SW architecture, the parameter server naturally becomes a communication bottleneck as the number of workers increases. To address this challenge, all-reduce collective operators have been proposed and adopted in the industry. In what follows, we briefly review two all-reduce mechanisms that are the most popular, namely ring-all-reduce (RAR) and tree-all-reduce (TAR).

1) Ring-All-Reduce: In RAR, all workers form a ring topology. As shown in Fig. 1, in a w-worker RAR operator, each worker divides its stochastic gradient into w equal-size sub-vectors. The process can be divided into two stages called "share-reduce" and "share-only," each of which lasts $w-1$ rounds in each training iteration. In the share-reduce stage, each worker receives a sub-vector from its upstream neighbor, performs reduction, and sends the result to its downstream neighbor. In the share-only stage, each worker receives a reduced sub-vector from its upstream neighbor and sends this particular information to its downstream neighbor. We refer readers to [12] for further details of RAR.

2) Tree-All-Reduce: In TAR, all workers form a tree topology. The TAR operator can also be divided into "share-reduce" and "share-only" stages. Consider a binary-tree TAR operator in Fig. 2 as an example. In the share-reduce stage, starting from the leaf nodes, each worker first receives the stochastic gradients from its children and performs reduction and then passes the result upward to its parent. In the share-only stage, upon the completion of reduction at the root node, the aggregated result will be passed downward layer by layer until it reaches every leaf node. We refer readers to [3] for further details of TAR.

Both RAR and TAR eliminate the communication bottleneck problem in DDL training in the sense that the worst-case traffic amount processed at each node is asymptotically independent of the number of workers w. Also, all-reduce mechanisms have different performances in terms of different performance metrics. To conclude this section, we summarize the performances of RAR and TAR operators in Table 1, where d denotes the gradient size and r represents the link rate between workers.

4 SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first present our system model in Section 4.1, which is followed by our problem formulation in Section 4.2.

4.1 System Model

Before we formally define our analytical system model, we first use a three-job scenario in Fig. 3 as an illustrative example to help readers better visualize the scheduling of all-reduce-based DDL jobs in a GPU cluster. In this example, the scheduling decision is as follows: Job 1 uses one worker (i.e., one GPU) on Server 1 and is the second to start; Job 2 uses two workers on Server 2 and is the third to start; and Job 3 uses two workers across two different GPU servers (Servers k and $k+1$) and is the first to start.

Note here that, in all-reduce-based DDL job scheduling, workers can be allocated within the same server or across different servers, which entails a trade-off between communication speed and resource utilization. On one hand, if the scheduling strategy is to allocate all workers of a job to the same server (e.g., Job 2), then this job's all-reduce operator uses *intra-server communication* that is much faster than the *inter-server communication* when multiple servers are used. However, this strategy could result in more severe

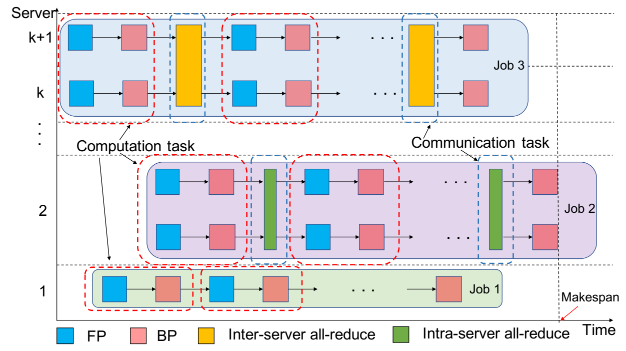

Figure 3: A three-job illustrative example of all-reduce-based DDL job scheduling.

resource fragmentation. On the other hand, if the scheduling strategy is to spread the workers of a job across multiple servers (e.g., Job 3), then it may incur significant communication contention with other DDL jobs, although GPU resources on different servers can be better utilized. Thus, one of the key goals in our scheduling design is to ensure that the DDL jobs are free of contention when multiple jobs are placed across different servers.

As shown in Fig. 3, each DDL job can be viewed as a *chain*-structure consisting of a sequence of computation and communication tasks executed in an alternating fashion. To see why this representation makes sense, recall that each iteration of a DDL training job consists of three sequential steps: FP, BP, and an all-reduce operation. Note that there is no dependence between workers in terms of FP and BP tasks. Also, due to the fact that BP always immediately follows FP, we can combine FP and BP into a single computation task. Also, note that all-reduce tasks are for communications only and they cannot start until the BP tasks of workers complete, each of these all-reduce operators can be viewed as a communication task that follows a computation task.

In scheduling multiple DDL jobs in GPU clusters, one of the most important performance metrics is the *makespan* of all jobs. For example, Job 2 in Fig. 3 is the last to finish among all jobs and its completion time marks the makespan of all three jobs. In this paper, our goal is to determine the optimal scheduling decision (including the *placement* of each job and the *starting time* of each job) to minimize the makespan of all jobs, subject to the constraint that jobs allocated across multiple servers should be *contention-free*.

4.2 Problem Formulation

With the system modeling in Section 4.1, we are now ready to formulate our all-reduce-based DDL scheduling problem. Consider a multi-tenant GPU cluster with a server set $\mathcal{S}$ and a set of DDL jobs denoted as $\mathcal{J}$. Each DDL job requests a certain number of workers and specifies the total number of training iterations. We consider a time-slotted system with a training time horizon $\mathcal{T}$, which is assumed to be sufficiently long for scheduling all DDL jobs. In this paper, we consider the "gang-scheduling" discipline that is widely adopted in practical large-scale GPU clusters [6], i.e., once a job is scheduled, all workers of this job are allocated simultaneously. The resources occupied by the job will also be released simultaneously upon completion. We further assume each GPU can only be occupied by one worker at a time. In what follows, we further specify the key components in our problem formulation.

1) Execution and Dependency Modeling: We use a binary variable $x_{j,i}^{u,v}[t]$ to indicate whether the link from server u to server v is used by the i-th iteration of job j in time-slot t ($x_{j,i}^{u,v}[t]=1$) or not ($x_{j,i}^{u,v}[t]=0$). Also, we let $y_{js}[t]$ denote the number of GPUs allocated to job j on server s in time-slot t. Correspondingly, we use a vector $\mathbf{y}_j[t] \triangleq [y_{js}[t], \forall s]^\top \in \mathbb{R}^{|\mathcal{S}|}$ to collect all placement decisions of job j in time-slot t. Clearly, the set of edges $\mathcal{E}(\mathbf{y}_j[t])$ in the all-reduce task of job j in time-slot t is determined by $\mathbf{y}_j[t]$. We let $v_{j,i} \triangleq \max_t\{x_{j,i}^{u,v}[t] > 0, \forall (u,v) \in \mathcal{E}(\mathbf{y}_j[t])\}$ denote the completion time-slot of the i-th communication task of job j.

Recall that each all-reduce-based DDL job can be represented as a chain of computation and communication tasks. We use $z_{j,i}[t]$ to indicate whether the computation task at i-th iteration of job j gets started in time-slot t ($z_{j,i}[t]=1$) or not ($z_{j,i}[t]=0$). Let F_j denote job j's required number of training iterations. To ensure that the computation task of each iteration can only be scheduled exactly once, we have:

$$\sum_{t\in\mathcal{T}} z_{j,i}[t] = 1, \quad \forall j \in \mathcal{J}, i = 1,\ldots,F_j. \tag{1}$$

We let p_j denote the processing time of job j's computation tasks. We use $\mu_{j,i}$ to denote the start time of the i-th communication task of job j. Since moving to the next iteration can only occur after the communication task for the current iteration is finished, we have:

$$\sum_{t\in\mathcal{T}} t z_{j,i+1}[t] \geq v_{j,i}, \quad \forall j \in \mathcal{J}, i = 1,\ldots,F_j. \tag{2}$$

Similarly, to ensure that a communication task cannot start before the completion of its proceeding computation task in the same iteration, we have:

$$\sum_{t\in\mathcal{T}} t z_{j,i}[t] + p_j \leq \mu_{j,i}, \quad \forall j \in \mathcal{J}, i = 1,\ldots,F_j. \tag{3}$$

2) Learning Job Modeling: We let G_j be the number of GPUs requested by job j. Let a_j and T_j be the start and completion times of job j, respectively. To ensure that job j's requested number of GPUs is satisfied throughout its training period, we have:

$$\sum_{s\in\mathcal{S}} y_{js}[t] = G_j, \quad \forall j \in \mathcal{J}, a_j \leq t \leq T_j. \tag{4}$$

In gang-scheduling, once a job is scheduled, its placement remains fixed throughout its training process. This can be modeled as:

$$y_{js}[t] = y_{js}[t-1], \quad \forall j \in \mathcal{J}, s \in \mathcal{S}, a_j < t \leq T_j. \tag{5}$$

Let m_j represent job j's gradient dimensionality, and let $b^{u,v}$ represent the bandwidth between servers u and v. Since the transmission time of a communication task is determined by the link with the smallest bandwidth in the edge set, we have $\tau_j \triangleq \frac{m_j}{\min_{(u,v)\in\mathcal{E}(\mathbf{y}_j[t])} b^{u,v}}$. To guarantee a sufficient number of time-slots are scheduled to complete the communication task in each iteration of job j, we have:

$$\sum_{t\in\mathcal{T}} x_{j,i}^{u,v}[t] \geq \tau_j, \quad \forall j \in \mathcal{J}, (u,v) \in \mathcal{E}(\mathbf{y}_j[t]), i = 1,\ldots,F_j. \tag{6}$$

It is worth noting that time precedence in (6) is guaranteed by constraint (2), meaning that the start time in the $(i+1)$-th iteration cannot be earlier than the completion time of the i-th iteration.

In this paper, we consider both preemptive and non-preemptive settings. To ensure that communication tasks are not interrupted once they have started under the non-preemptive setting, we have:

$$x_{j,i}^{u,v}[t] = x_{j,i}^{u,v}[t-1], \quad \forall j \in \mathcal{J}, \\ (u,v) \in \mathcal{E}(\mathbf{y}_j[t]), \mu_{j,i} < t \le \mu_{j,i} + \tau_j, i = 1, \dots, F_j. \quad (7)$$

3) Resource Constraint and Contention-Free Assurance: We use N_s to denote the capacity of GPU server $s \in \mathcal{S}$. To ensure that the GPU allocation does not exceed the limit of each server at all times, we have:

$$\sum_{j \in \mathcal{J}} y_{js}[t] \le N_s, \quad \forall s \in \mathcal{S}, t \in \mathcal{T}. \quad (8)$$

To ensure contention-free, we need to guarantee that at most one communication task is using an inter-server link at any time-slot t:

$$\sum_{j \in \mathcal{J}} \sum_{i=1,\dots,F_j} x_{j,i}^{u,v}[t] \le 1, \quad \forall u \in \mathcal{S}, v \in \mathcal{S}\backslash\{u\}, t \in \mathcal{T}. \quad (9)$$

4) Completion Time Modeling: Let T_j be the latest time slot where one or more servers are still hosting workers for job j, i.e.,

$$T_j = \arg\max_{t \in \mathcal{T}} \{y_{js}[t] > 0, \exists s \in \mathcal{S}\}, \quad \forall j \in \mathcal{J}. \quad (10)$$

5) Objective Function and Problem Statement: In this work, our objective is to minimize the makespan of all all-reduce-based DDL jobs, subject to contention-free constraints. Thus, the DDL training scheduling (DDLTS) problem can be formulated as follows:

$$\textbf{DDLTS}: \min_{\mathbf{x},\mathbf{y}} \max_{j \in \mathcal{J}} T_j$$
$$\text{subject to Constraints (1) – (10).}$$

5 SOLUTION APPROACH

Problem DDLTS is a mixed-integer non-convex programming problem, which is NP-Hard in general. Additionally, the special-structured constraint in (10) with the arg max operator, makes it difficult to apply conventional optimization techniques. Due to all these challenges, our goal is to design an approximation algorithm with *provable* performance. Our *basic idea* for solving DDLTS is as follows:

- To address the non-amenable structure in constraint (10), instead of attacking Problem DDLTS directly, we will first reformulate DDLTS into a logically equivalent problem, which uses a higher dimensional search space to trade for a simpler problem structure. This not only eliminates the argmax operator in (10), but also allows many of the constraints to be automatically satisfied by a search-based algorithm design approach.
- The simplified problem structure of the reformulated problem allows us to solve the problem in a divide-and-conquer fashion by decomposing the solution into two stages: *i) job placement* and *ii) communication task scheduling*, each of which can then be solved in an efficient manner with performance guarantees.

Next, we will describe each key component in our solution.

5.1 Reformulation of Problem DDLTS

First, we observe that if we know the space of all scheduling combinations that are feasible to the complex structure of Problem DDLTS, then the problem can be solved by a search-based approach in the space of feasible schedules. We let $\mathcal{N} = \{1, \dots, N\}$ be the set of all GPUs in the cluster. Let $\mathcal{Y} = \{\mathbf{y}^1, \dots, \mathbf{y}^{|\mathcal{Y}|}\}$ be the set of all feasible *job placements*, where $\mathbf{y}^k = \{\mathbf{y}_1^k, \dots, \mathbf{y}_J^k\}$ and $\mathbf{y}_j^k = \{h_{js}^k[t], \forall s, t\}$. Here, $h_{js}^k[t]$ defines the number of workers allocated to server s for job j in time-slot t under placement $\mathbf{y}^k$.

Note that once a feasible job placement $\mathbf{y}^k$ is given, the schedules of all jobs' computation tasks are fully specified. However, it remains to specify the schedules of all jobs' communication task schedules, so that all jobs are *contention-free*. Toward this end, we let $\mathcal{P}^k = \{\mathbf{p}_1^k, \dots, \mathbf{p}_{|\mathcal{P}^k|}^k\}$ represent the set of all contention-free communication task schedules under placement $\mathbf{y}^k$. Here, $\mathbf{p}_l^k = \{\mathbf{p}_{l,1}^k, \dots, \mathbf{p}_{l,J}^k\}$ is the l-th contention-free schedule that contains all jobs' communication task schedules, where $\mathbf{p}_{l,j}^k$ specifies, under the l-th schedule, how each inter-server link will be utilized by job j in each time-slot of the training horizon.

Note that if $\mathbf{y}^k$ and $\mathbf{p}_l^k$ are given, then each job j's starting time and training time are fully specified, which we denote as $a_j(\mathbf{y}^k, \mathbf{p}_l^k)$ and $\rho_j(\mathbf{y}^k, \mathbf{p}_l^k)$, respectively. Then, Problem DDLTS can be reformulated as searching a schedule from $\mathcal{Y}$ and $\mathcal{P}^k$ to minimize the makespan of all jobs. Let $\chi_{l,j}^k$ be the binary variable to indicate whether job j follows placement $\mathbf{y}^k$ and schedule $\mathbf{p}_l^k$ ($\chi_{l,j}^k = 1$) or not ($\chi_{l,j}^k = 0$). Then, Problem DDLTS can be reformulated as the following logically equivalent problem:

$$\textbf{R-DDLTS:} \min_{\chi_{l,j}^k} \max_j \; \chi_{l,j}^k \big(a_j(\mathbf{y}^k, \mathbf{p}_l^k) + \rho_j(\mathbf{y}^k, \mathbf{p}_l^k)\big) \quad (11)$$

$$\text{subject to} \sum_{k \in \{1,\dots,|\mathcal{Y}|\}} \sum_{l \in \{1,\dots,|\mathcal{P}^k|\}} \chi_{l,j}^k = 1, \; \forall j \in \mathcal{J}, \quad (12)$$

$$\chi_{l,j}^k = \chi_{l,j'}^k, \forall j, j' \in \mathcal{J}, k \in \{1, \dots, |\mathcal{Y}|\}, l \in \{1, \dots, |\mathcal{P}^k|\}, \quad (13)$$
$$\chi_{l,j}^k \in \{0,1\}, \forall j \in \mathcal{J}, k \in \{1, \dots, |\mathcal{Y}|\}, l \in \{1, \dots, |\mathcal{P}^k|\}.$$

Here, constraint (12) ensures that all jobs will choose exactly one schedule. Constraint (13) requires all jobs follow the same job placement and communication task schedule. Although Problem R-DDLTS clearly has an exponentially large search space, it is an integer linear min-max problem with a much simpler problem structure in (11)–(13) than the original problem. This enables our subsequent tractable approximation algorithm design.

5.2 Main Algorithm for Problem R-DDLTS

We note that Problem R-DDLTS remains a challenging NP-hard problem due to the following major difficulties:

(1) Problem R-DDLTS has an *unknown* and exponentially large search space in $\mathcal{Y}$ and $\mathcal{P}^k, \forall k$, which means that the search is highly non-straightforward.
(2) Problem R-DDLTS has a min-max objective, which precludes the use of on-the-shelf integer programming solvers and necessitates customized algorithm design.
(3) The objective coefficients $(a_j(\mathbf{y}^k, \mathbf{p}_l^k) + \rho_j(\mathbf{y}^k, \mathbf{p}_l^k))$ in Problem R-DDLTS are difficult to characterize in closed-form expressions since the processing time $\rho_j(\mathbf{y}^k, \mathbf{p}_l^k)$ can potentially affected by worker idleness due to contention avoidance.

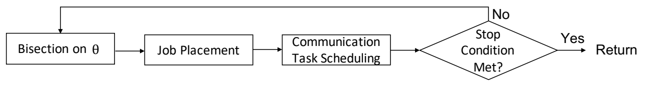

Figure 4: The overall algorithmic idea of the search-based approach for indirectly solving Problem R-DDLTS.

To address the challenges above, we propose to *indirectly* solve Problem R-DDLTS based on the following key observation: the makespan of all DDL jobs is intimately related to the workload of each individual GPU. Moreover, a necessary condition for achieving low makespan is to ensure a low workload at each individual GPU, i.e., to perform load balancing. This key observation motivates the following search-based idea to solve Problem R-DDLTS.

We use a parameter θ to keep track of the maximum workload upper bound of all GPUs. Under the constraint that the workload on each GPU to not exceed θ, we determine whether there exists a feasible job placement and its associated contention-free communication tasks schedules. Clearly, if θ is too small, then no feasible job placement and contention-free communication task schedule can be found. On the other hand, if θ is too large, the gap between the makespan of the identified solution could be far from the optimal makespan. Under this approach, solving makespan minimization in Problem R-DDLTS can be transformed into searching for the smallest possible θ that admits a feasible job placement and its corresponding contention-free communication task schedule. This approach eliminates the need for the knowledge of $\mathcal{Y}$, $\mathcal{P}^k$, and $\rho_j(\mathbf{y}^k, \mathbf{p}_l^k)$-information. Better yet, since θ is a scalar, the search for a good θ-value can be efficiently done by the bisection method. This algorithmic idea is illustrated in Fig. 4.

Specifically, the bisection method for searching the tightest workload upper bound θ is presented in Algorithm 1. For a given training time horizon $[1, T]$, we choose the middle point $(1 + T)/2$ as the initial θ-value (Line 3). Inspired by the technique in [19], we use a parameter κ to represent the number of servers that can be used by a job, which will be used as an input in our job placement and communication task scheduling subroutines (to be defined later). Then, we iterate over all possible values of κ (Line 4). Given θ- and a κ-values, we use Algorithm 2 to try constructing a feasible job placement and use Algorithm 3 to construct a contention-free communication task schedule and calculate the corresponding makespan (Lines 5-6). If a feasible job placement and contention-free communication task schedule is found, we update them (Lines 7-8). Finally, based on the feasibility of the current θ, we determine the next shrink search space for θ (Lines 9-12). With the above main algorithm, in what follows, we specify the subroutines for job placement and contention-free communication task scheduling.

5.3 The Job Placement Algorithm

To identify a feasible job placement to potentially satisfy the workload upper bound θ, our key idea is to balance the workload among the workers by selecting the workers with the lightest workload. Let $\kappa > 0$ be a job size threshold value. For a small job j with $G_j \leq \kappa$, we directly select the top-G_j workers with the lightest workload, where G_j is the number of GPUs requested by job j (cf. Eq. (4)). If there are not enough available GPUs, we wait for some jobs to finish (Lines 2-8). For a large job j with $G_j > \kappa$, we first sort the servers by the average per-worker workload and select the top-k servers with their total GPU capacity more than G_j, then pick the top-G_j workers from the selected server set (Lines 10-11). Then, we follow the same procedure as that for small jobs to pick the top-G_j workers from the selected server set $\mathcal{S}'$ (Line 12).

Algorithm 1 Workload Upper Bound Bisection Search.

```
1: $\mathbf{y} \leftarrow \emptyset$, $l \leftarrow 1$, $r \leftarrow T$
2: repeat
3:     $\theta \leftarrow (l + r)/2$
4:     for $\kappa = 1, 2, \ldots, n_g$ do
5:         return $\mathbf{y}_\theta^\kappa$ using Algorithm 2
6:         return $\mathbf{p}_\theta^\kappa$ using Algorithm 3 given $\mathbf{y}_\theta^\kappa$ and $\theta$
7:         if $\mathbf{p}_\theta^\kappa \neq \emptyset$ then
8:             $\mathbf{y} \leftarrow \{\mathbf{y}_\theta^\kappa, \mathbf{p}_\theta^\kappa\}$
9:     if exists $\theta, \kappa$ such that $\mathbf{p}_\theta^\kappa \neq \emptyset$ then
10:        $r \leftarrow \theta - 1$
11:    else
12:        $l \leftarrow \theta + 1$
13: until $l <= r$
14: return $\mathbf{y}$
```

Algorithm 2 Placement of Top-κ Lightest Workload First (LWF-κ).

```
1: for $j = 1, 2, \ldots, |\mathcal{J}_s|$ do
2:     if $G_j \leq \kappa$ then
3:         if at least $G_j$ available workers in $\mathcal{S}$ then
4:             Sort jobs by $G_j$ in non-decreasing order to yield $\mathcal{J}_s$
5:             Pick $G_j$ workers with LWF as $\mathbf{y}_{j,\theta}^\kappa$
6:             Update workers workload and go to Line 1
7:         else
8:             Wait until some job exits, then go to Line 3
9:     else
10:        Sort servers by average per-worker workload
11:        Select top-$k$ servers s.t. $\sum_{s=1}^k N_s \geq G_j$, denote set as $\mathcal{S}'$
12:        Go to Line 3 with $\mathcal{S} \leftarrow \mathcal{S}'$
13: return $\mathbf{y}_\theta^\kappa \leftarrow \{\mathbf{y}_{j,\theta}^\kappa, \forall j\}$
```

5.4 The Contention-Free Communication Task Scheduling Algorithm

In Algorithm 1, once the job placement is determined, the next step is to determine a contention-free communication task schedule for each active job in each time-slot (i.e., a job that is allocated with GPU resources and not yet finished). Also, such a schedule should satisfy the given GPU workload upper bound θ upon the completion of all jobs. Recall from Section 4.1 that a DDL can be modeled as a chain-structured graph consisting of computation and communication tasks in an alternating fashion. Although computation tasks can be performed on each GPU independently, communication tasks from different jobs that happen to perform all-reduce in the same time-slot need to be carefully scheduled since they may contend for a set of shared inter-server links.

In this paper, we consider contention-free communication task scheduling for both preemptive and non-preemptive settings. In

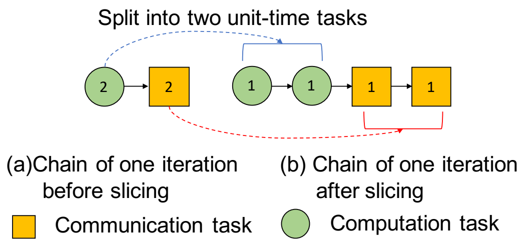
Split into two unit-time tasks
2
2
1
1
1
1
(a)Chain of one iteration before slicing
(b) Chain of one iteration after slicing
Communication task
Computation task

Figure 5: An illustrative example of task slicing.

the non-preemptive setting, a task cannot be interrupted once its execution begins. In contrast, a preemptive task is interruptible during its execution. Interestingly, thanks to the chain-structured of DDL jobs, the preemptive setting can be unified with the non-preemptive setting through "task slicing" as shown in Fig. 5, where one slices each task into "atomic subtasks" that are of the smallest time-unit and cannot be further preempted.

For both preemptive and non-preemptive settings, the main challenge of contention-free communication task scheduling stems from the exclusive use of the inter-server links: if two or more all-reduce tasks are competing for the same link, they have to time-share this link following some scheduled order to avoid contention. At first glance, this problem appears to share some similarity with a variant of the classic multi-processor task scheduling problem [5]. However, our contention-free task scheduling is far more challenging than the classic multi-processor task scheduling problem in that there exist two *distinct* types of processors (computation and communication) and each processor is reserved for specific tasks (computation tasks are executed by GPUs and communication tasks are executed by inter-sever links). Also, the computation-communication dependency further complicates the scheduling decisions.

Our key idea in contention-free communication task scheduling is based on the *"largest remaining execution time first"* (LRETF) policy. Intuitively, LRETF achieves load balancing by greedily equalizing workload, which benefits the makespan performance. Before we present our algorithm, we use a simple two-job RAR example in Fig. 6 to illustrate the basic idea of LRETF. At the beginning of the scheduling horizon, we construct a multi-chained graph denoted as $G(\mathbf{y}_\theta^\kappa)$ based on the placement $\mathbf{y}_\theta^\kappa$. In this case, the workers of both jobs are placed across servers 1 and 2, leading to contention for the inter-server link. Suppose that both jobs have two iterations. In each iteration, job 1 contains one atomic computation and one atomic communication subtasks, while job 2 has one atomic computation and two atomic communication subtasks. Under LRETF, the execution order of each job's atomic communication and computation subtasks is indicated by the number on each subtask, which denotes the execution time-slot index. Fig. 6 shows that, under LRETF, only one atomic communication subtask is executed in any given time-slot. The red and blue circles in Fig. 6 indicate how a group of potentially interfering atomic communication subtasks resolve their contentions. We note that, except for the initial computation subtask, all other computation subtasks are executed concurrently with some communication subtask due to the independence of computation and communication across different jobs.

Our LRETF approach is formally presented in Algorithm 3. Given the multi-chained graph $G(\mathbf{y}_\theta^\kappa)$ under placement $\mathbf{y}_\theta^k$, we start with

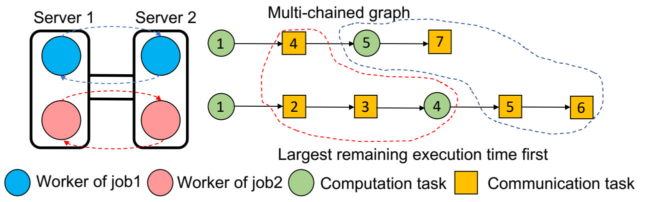

Figure 6: An example of multi-chained graph with task slicing (number in each task is the starting execution time slot).

Algorithm 3 Contention-Free Task Scheduling with Largest Remaining Execution Time First (LRETF).

```
1:  Input: Multi-chained graph $G(\mathbf{y}_\theta^\kappa)$ and $\theta$
2:  if Preemption is allowed then
3:      Perform task-slicing on $G(\mathbf{y}_\theta^\kappa)$
4:  $\mathbf{p}_\theta^\kappa \leftarrow \{\}$
5:  for $t = 1, \ldots, T$ do
6:      $\mathcal{F} \leftarrow \emptyset$
7:      Sort the active tasks in non-increasing order based on their
    remaining chain length to yield the set $\mathcal{G}[t]$
8:      if $t > \theta$ and $|G(\mathbf{y}_\theta^\kappa)| > 0$ then
9:          return $\emptyset$
10:     if $|G(\mathbf{y}_\theta^\kappa)| = 0$ then
11:         return task schedule $\mathbf{p}_\theta^\kappa$
12:     for Each task $l \in \mathcal{G}[t]$ do
13:         if $l$ is a computation task then
14:             $G(\mathbf{y}_\theta^\kappa) \leftarrow G(\mathbf{y}_\theta^\kappa) \setminus \{l\}, \mathbf{p}_\theta^\kappa[t] \leftarrow \mathbf{p}_\theta^\kappa[t] \cup \{l\}$
15:         else if $l$ has no shared link with tasks in $\mathcal{F}$ then
16:             $\mathcal{F} \leftarrow \mathcal{F} \cup \{l\}, \mathbf{p}_\theta^\kappa[t] \leftarrow \mathbf{p}_\theta^\kappa[t] \cup \{l\}, G(\mathbf{y}_\theta^\kappa) \leftarrow G(\mathbf{y}_\theta^\kappa) \setminus \{l\}$
```

performing task slicing on the graph if preemption is considered (Lines 2-3). Then, we iterate over time t from 1 to T (Line 5). In each time slot, the active tasks are sorted in non-increasing order based on their remaining execution time, which yields the job set $\mathcal{G}[t]$ (Line 7). If not all jobs can be finished within θ time-slots, we return an empty set to indicate infeasibility (Lines 8-9); otherwise, we return the task schedule (Lines 10-11). For each task in the job set $\mathcal{G}[t]$, if it is a computation task, it will be executed immediately (Lines 13-14); otherwise, it only will be executed if no conflict exists with an executing task in the set $\mathcal{F}$ (Lines 15-16).

In the next section, we will show that our solution approach in Algorithms 1–3 enjoys strong theoretical performance guarantees.

6 THEORETICAL PERFORMANCE ANALYSIS

In this section, we will establish the theoretical performance guarantee of our proposed algorithms.

First, we show the performance guarantee of Algorithm 3 in Theorems 1 and 2. Note that we do not consider any non-worker-conserving policies since they can not be optimal. We let T^* be the optimal makespan obtained from an offline algorithm, and let T' be the makespan obtained from our proposed algorithm. Then, we have the following results for Algorithm 3.

Theorem 1. *The task scheduling approach in Algorithm 3 under non-preemptive setting enjoys a 2-approximation ratio.*

We relegate the proof to Appendix A. It is evident that the proved ratio in Theorem 1 can serve as an upper bound for the task scheduling under the preemptive setting since it can be unified with the non-preemptive setting through "task slicing." Surprisingly, we show that Algorithm 3 is optimal for the preemptive task setting.

Theorem 2. *The task scheduling approach in Algorithm 3 under preemptive setting is optimal.*

The proof of Theorem 2 can be found in Appendix B. The intuition behind the optimality of Algorithm 3 for the preemptive setting is that the LRETF policy aims to evenly distribute the workload across the jobs, thus resulting in an improved makespan. Specifically, being able to divide the tasks into atomic segments and preempt renders it possible to make more precise workload adjustments, which, in turn, can help alleviate the problem of excessively skewed chain lengths. We note that the optimality result of LRETF under the preemptive setting could also be of independent interest for other scheduling problems.

Next, we prove the overall approximation ratio for our proposed algorithm. Our proof strategy is to theoretically bound the gap between the optimal makespan T^* and the workload upper bound θ obtained by Algorithm 1. Toward this end, we use $\hat{W}_{jg}^{k,l}(\pi) \triangleq \chi_{l,j}^{k,\pi}\hat{\rho}_j(\mathbf{y}^{k,\pi}\mathbf{p}_l^{k,\pi})$ to denote the workload of GPU g added by job j under placement $\mathbf{y}^{k,\pi}$ and schedule $\mathbf{p}_l^{k,\pi}$ that are obtained from some algorithm π. Then, the workload of GPU g is the sum of all workloads added by all jobs that use g, i.e. $\hat{W}_g^{\pi} = \sum_j \sum_k \sum_l \hat{W}_{jg}^{k,l}(\pi)$. We let $\hat{W}_{\max}^{\pi} \triangleq \max_{g\in\mathcal{N}} \hat{W}_g^{\pi}$ represent the heaviest workload among all GPUs. Clearly, $\hat{W}_{\max}^{\pi} \le \theta$. However, note that the ground-truth execution time $\rho_j(\mathbf{y}^k, \mathbf{p}_l^k)$ is unknown, which creates difficulty in analyzing T^*. To address this challenge, we note that since $\rho_j(\mathbf{y}^k, \mathbf{p}_l^k)$ is bounded, there must exist two constants $\phi \le 1$ and $\Phi \ge 1$ such that $\hat{\rho}_j(\mathbf{y}^k, \mathbf{p}_l^k) \in [\phi\rho_j(\mathbf{y}^k, \mathbf{p}_l^k), \Phi\rho_j(\mathbf{y}^k, \mathbf{p}_l^k)]$. Thus, we can use $\frac{1}{\Phi}\hat{\rho}_j(\mathbf{y}^k, \mathbf{p}_l^k)$ as a lower bound for T^* in our analysis:

$$\sum_{j\in\mathcal{J}} \sum_{k\in\{1,\dots,|\mathcal{Y}|\}} \sum_{l\in\{1,\dots,|\mathcal{P}^k|\}} \chi_{l,j}^{k} \frac{\hat{\rho}_j(\mathbf{p}_l^k)}{\Phi} \le T^* \le \theta, \ \forall g \in \mathcal{N}, \quad (14)$$

Then, if we can bound the gap between the left-hand side (LHS) and right-hand side (RHS) in (14), the gap between T^* and θ can also be bounded. For this purpose, we have the following lemmas:

Lemma 3 (Maximum Workload Upperbound). *Algorithm 1 produces a schedule with the maximum workload $\hat{W}_{\max}^{\text{Alg1}} = \tilde{\theta}$, where $\tilde{\theta}$ is the workload upper bound found by our algorithm.*

We let T_g^{idle} and T_g^{busy} be the idle time introduced by gang-scheduling and the busy time of worker g.

Lemma 4 (Busy and Idle Time). *Algorithm 1 produces a schedule with $T_g^{\text{idle}} \le (G_j - 1)\hat{W}_{\max}^{\text{Alg1}}$ and $T_g^{\text{busy}} \le \hat{W}_{\max}^{\text{Alg1}}$.*

Lemma 5 (Makespan). *Algorithm 1 achieves a makespan at most $(n_g + 1)\hat{W}_{\max}^{\text{Alg1}}$ under the gang-scheduling discipline and the largest remaining execution time first task scheduling policy.*

Proof. According to Theorems 1 and 2, the makespan returned by Algorithm 3 is no more than twice the optimum given a schedule. In other words, the idle time introduced by contention is limited to $T_g^{'\text{idle}} \le T_g^{\text{busy}}$. It thus follows as:

$$\begin{aligned} T' &= \max_{g\in\mathcal{N}}(T_g^{\text{busy}} + T_g^{\text{idle}} + T_g^{'\text{idle}}) \\ &\overset{Lem.4}{\le} \max_{j\in\mathcal{J}} \left(2\hat{W}_{\max}^{\text{Alg1}} + (G_j - 1)\hat{W}_{\max}^{\text{Alg1}}\right) \\ &= \max_{j\in\mathcal{J}}(G_j + 1)\hat{W}_{\max}^{\text{Alg1}} = (n_g + 1)\hat{W}_{\max}^{\text{Alg1}}, \end{aligned}$$

and the proof is complete. □

The proofs of Lemmas 3-5 follow similar approaches in [19]. However, we note that, in comparison to [19], the need for contention-free communication task scheduling results in extra idling in addition to the synchronization barrier of gang-scheduling in [19]. This is due to the fact that some job's communication tasks can be blocked if multiple jobs compete for the same inter-server link. Thus, in addition to the busy time T_g^{busy} and idle time T_g^{idle} from gang-scheduling in [19], we also have to consider the idle time $T_g^{'\text{idle}}$ introduced by communication contention avoidance in the proofs of Lemmas 3-5. Here, $T_g^{'\text{idle}}$ implicitly quantifies the performance loss that is resulted from the task scheduling strategy implemented in Algorithm 3.

Next, we examine the gap between the workload upper bound $\tilde{\theta}$ found by Algorithm 1 and the optimal θ^* in the RHS of (14). Given a fixed task scheduling strategy l, we define β as the maximum ratio of workload for job j between two different configurations, i.e., $\beta \triangleq \max_{\forall j,k_1,k_2} \rho_j(\mathbf{y}^{k_1}, \mathbf{p}_l^{k_1})/\rho_j(\mathbf{y}^{k_2}, \mathbf{p}_l^{k_2})$. Then, we have:

Lemma 6. *The tightest wokload upperbound $\tilde{\theta}$ returned by Algorithm 1 satisfies $\tilde{\theta} \le 2\beta\frac{\Phi}{\phi}\theta^*$.*

Proof. We define k^* and $\tilde{k}$ as the optimal and Algorithm 1 returned job placement indices, respectively. Similarly, we define l^* and $\tilde{l}$ as the optimal and Algorithm 1 returned schedule indices, respectively. We let $\mathcal{G}(\mathbf{y}^k, \mathbf{p}_l^k)$ as the set of selected GPUs if the placement $\mathbf{y}^k$ and schedule $\mathbf{p}_l^k$ is used. We have

$$\begin{aligned} \tilde{\theta} &\overset{\text{Lem.3}}{=} \max_{g\in\mathcal{G}(\mathbf{y}^{\tilde{k}},\mathbf{p}_{\tilde{l}}^{\tilde{k}})} \sum_{j\in\mathcal{J}} \frac{\hat{\rho}_j(\mathbf{y}^{\tilde{k}}, \mathbf{p}_{\tilde{l}}^{\tilde{k}})}{\Phi} \overset{Thms.1\&2}{\le} \max_{g\in\mathcal{G}(\mathbf{y}^{\tilde{k}},\mathbf{p}_{l^*}^{\tilde{k}})} \sum_{j\in\mathcal{J}} \frac{2\hat{\rho}_j(\mathbf{y}^{\tilde{k}}, \mathbf{p}_{l^*}^{\tilde{k}})}{\Phi} \\ &\overset{(a)}{\le} \max_{g\in\mathcal{G}(\mathbf{y}^{\tilde{k}},\mathbf{p}_{l^*}^{\tilde{k}})} \sum_{j\in\mathcal{J}} \frac{2\beta\frac{\Phi}{\phi}\hat{\rho}_j(\mathbf{y}^{k^*}, \mathbf{p}_{l^*}^{k^*})}{\Phi} \\ &\overset{(b)}{\le} \max_{g\in\mathcal{G}(\mathbf{y}^{k^*},\mathbf{p}_{l^*}^{k^*})} \sum_{j\in\mathcal{J}} \frac{2\beta\frac{\Phi}{\phi}\hat{\rho}_j(\mathbf{y}^{k^*}, \mathbf{p}_{l^*}^{k^*})}{\Phi} \overset{\text{Eq.(14)}}{\le} 2\beta\frac{\Phi}{\phi}\theta^*, \end{aligned}$$

where (a) follows $\frac{\hat{\rho}_j(\mathbf{y}^{k_1},\mathbf{p}_l^{k_1})}{\hat{\rho}_j(\mathbf{y}^{k_2},\mathbf{p}_l^{k_2})} \le \frac{\Phi\rho_j(\mathbf{y}^{k_1},\mathbf{p}_l^{k_1})}{\phi\rho_j(\mathbf{y}^{k_2},\mathbf{p}_l^{k_2})} \le \frac{\beta\Phi}{\phi}$, and (b) can be proved by contraction following the similar method in Lem. 4 of work [19]. This completes the proof. □

Finally, by putting Lemmas 5–6 together, we have the following approximation ratio for Algorithm 1:

Theorem 7 (Approximation Ratio). *Our proposed Algorithm 1 is $2\beta(n_g+1)\frac{\Phi}{\phi}$-approximate.*

Proof. The stated approximation result follows from:

$$T' \overset{\text{Lem.5}}{\leq} (n_g+1)\hat{W}_{\max}^{\text{Alg1}} \overset{\text{Lem.3}}{=} (n_g+1)\tilde{\theta}$$
$$\overset{\text{Lem.6}}{\leq} 2\beta(n_g+1)\frac{\Phi}{\phi}\theta^* \overset{(a)}{\leq} 2\beta\frac{\Phi}{\phi}(n_g+1)T^*,$$

where (a) is due to the fact that we use the processing time estimation $\hat{\rho}_j(\mathbf{y}^k, \mathbf{p}_l^k)$ without considering idling, which implies $\theta^* \leq T^*$ and the proof is complete. □

Remark 1. Note that the result in Theorem 7 does not depend explicitly on the parameter κ in Algorithm 1. This is because Theorem 7 is only a worst-case upper bound that depends on $\tilde{\theta}$, which in turn depends on κ. Hence, κ is implicitly captured in Theorem 7.

Also, the time complexity of Algorithm 1 is:

Theorem 8 (Polynomial Running Time). *Time complexity of Algorithm 1 is $O(n_g|\mathcal{J}|N\log N\log T + T|\mathcal{J}|\log|\mathcal{J}|)$.*

Proof. The time complexity of our algorithm comes from the job placement and task scheduling steps. The sorting operation is the most time-consuming step in the job placement algorithm (Alg. 1). In the worst case ($G_j \leq \kappa$), we need to sort all GPUs in the cluster, which takes $O(N\log N)$ time. Thus, scheduling each job takes $O(N\log N)$ time, and scheduling all $|\mathcal{J}|$ jobs takes $O(|\mathcal{J}|N\log N)$ time. The job placement algorithm uses the bisection method to search for θ_ϕ, which requires $n_g\log T$ trials and contributes to the time complexity of $O(n_g|\mathcal{J}|N\log N\log T)$. The task scheduling algorithm (Alg. 3) has a worst-case running time of T, and at each time slot, we need to sort the jobs based on their remaining execution time, which takes $|\mathcal{J}|\log|\mathcal{J}|$, resulting in an overall time complexity of $O(n_g|\mathcal{J}|N\log N\log T + T|\mathcal{J}|\log|\mathcal{J}|)$. □

7 NUMERICAL RESULTS

In this section, we conduct experiments to verify the effectiveness of our proposed Algorithm 1.

1) Experiment Settings: We follow the same setup as in [17, 19], using the Microsoft job trace [9] as the job workload for fair comparisons. To create a more challenging communication-contention environment, we scale down the original job trace and select a total of 160 DDL jobs, which includes 80 single-GPU jobs, 14 2-GPU jobs, 26 4-GPU jobs, 30 8-GPU jobs, 8 16-GPU jobs, and 2 32-GPU jobs. The number of training iterations F_j for each job is randomly chosen from [1000, 6000]. The computing cluster has 60-100 servers, each of which has two to eight GPUs [8].

2) Comparison Baselines: Our proposed algorithm has two major components: job placement and communication task scheduling. For job placement, we compare our algorithm with three representative job placement algorithms. 1) *FF* (First-Fit) [15]: FF always chooses the first G_j available GPUs in order; 2) *LS* (List-Scheduling) [15]: LS chooses servers with the least workload first, then selects GPUs follows FF policy; 3) *RAND*: RAND randomly chooses GPUs to place its workers. For communication task scheduling, we compare our LRETF algorithm with: 1) *SRETF* (Smallest Remaining Execution Time First): SRETF chooses the communication task whose remaining chain length is the smallest; 2) *RAND*: RAND randomly choose the communication task to execute.

3) Results: We first compare the makespan achieved by our LWF-κ algorithm with the baselines with $T = 1400$ and $S = 60$, and the results are presented in Fig. 7. As shown in Fig. 7, our algorithm outperforms the other scheduling policies and produces the shortest makespan. FF suffers from a highly skewed and unbalanced workload. RAND has a more evenly distributed workload and better makespan compared to FF. The key advantage of our LWF-κ algorithm lies in its ability to strike a balance between even worker distribution and worker consolidation, the benefit of which becomes more pronounced when the resources are limited.

Next, we examine the impact of the big/small job size threshold parameter κ on our LWF-κ algorithm. We choose κ-value from the set $\{1, 2, 4, 8, 10, 16, 32\}$ and set $T = 1200$ and $S = 60$. The results are shown in Fig. 8. The makespan decreases initially as κ increases, but then increases and decreases again. When κ is small, more small jobs are placed on shared servers, leading to less resource fragmentation and a lower communication overhead for larger jobs. However, as κ continues to increase, larger jobs compete for the same network resources, causing more contention to resolve and increasing the makespan. Finally, as κ becomes sufficiently large, most of the jobs use shared servers to schedule their workers, which slightly reduces the overhead due to a smaller span of workers across servers.

Next, we evaluate the performance of our proposed scheduling policy under the preemptive setting. We compare it with the aforementioned baselines. The number of communication tasks of each job is chosen from $[200, 600]$. The sizes of computation and communication tasks are randomly selected between 1 and 10. To create a highly contended environment, we reduce the number of servers to three. The results are presented in Fig. 9, where we can see that our LRETF scheduling algorithm outperforms the others and has the lowest makespan. The "smallest remaining execution time first" (SRETF) policy has the worst performance. These results suggest that prioritizing tasks with the longest remaining execution time helps balance the workload, reducing the makespan, and improving resource utilization.

Lastly, we investigate the performance difference between non-preemptive and preemptive task scheduling using the same setting as in Fig. 9. As shown in Fig. 10, the makespan is reduced significantly by allowing preemption, reducing it by half. Additionally, as the number of jobs increases, the gap between these two settings policies becomes larger. It is also worth noting that when the communication tasks are much larger than computation tasks, the makespan performance gain of the preemptive setting are even more prominent. This is due to the fact that, in the non-preemptive setting, a larger communication task can block other jobs with longer remaining execution times, leading to a skewed remaining chain length and worse makespan.

8 CONCLUSION

In this paper, we investigated the problem of resource scheduling for all-reduce-based DDL jobs in multi-tenant computing clusters. We optimized makespan for all-reduce-based DDL job training with the goal of resolving communication contention among jobs. Toward

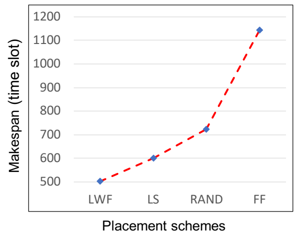

Figure 7: Makespan comparison on different placements.

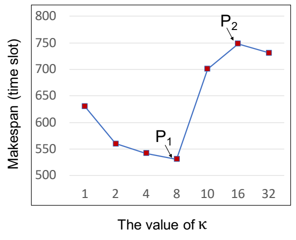

Figure 8: Impact of different κ on makespan.

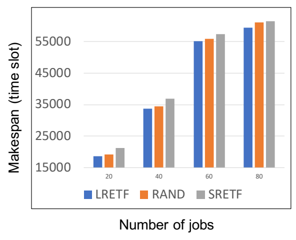

Figure 9: Makespan comparison on different schedules.

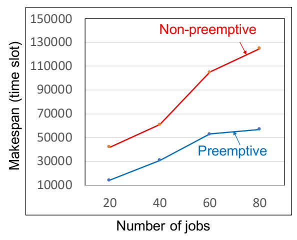

Figure 10: Preemptive vs non-preemptive.

this end, we first formulated the problem as a mixed-integer non-convex optimization problem. We then reformulated the problem to an equivalent integer problem to enable a search-based algorithm with provable performance guarantees. Lastly, we proposed an approximation algorithm by decomposing the problem into job placement and communication task scheduling subproblems. Collectively, our work contributes to the field of resource scheduling and optimization for DDL training in computing clusters.

REFERENCES

[1] Abadi, M., Barham, P., et al. TensorFlow: A system for large-scale machine learning. In *Proc. of USENIX OSDI* (2016).
[2] Bao, Y., Peng, Y., and Wu, C. Deep learning-based job placement in distributed machine learning clusters. In *IEEE INFOCOM* (2019).
[3] Cai, Y. Tree-based allreduce communication on mxnet. Tech. rep., https://www.ece.ucdavis.edu/ ctcyang/pub/amaz-techreport2018.pdf, 2018.
[4] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (USA, 2012), NIPS'12, Curran Associates Inc., pp. 1223–1231.
[5] Erlebach, T., Kääb, V., and Möhring, R. H. Scheduling and/or-networks on identical parallel machines. In *Approximation and Online Algorithms* (Berlin, Heidelberg, 2004), R. Solis-Oba and K. Jansen, Eds., Springer Berlin Heidelberg, pp. 123–136.
[6] Feitelson, D. G., and Rudolph, L. Gang scheduling performance benefits for fine-grain synchronization. In *Journal of Parallel and distributed Computing* (1992), vol. 16, pp. 306–318.
[7] Hu, Z., Tu, J., and Li, B. Spear: Optimized dependency-aware task scheduling with deep reinforcement learning. In *in 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)* (2019).
[8] Jeon, M., Venkataraman, S., Phanishayee, A., Qian, J., Xiao, W., and Yang, F. Multi-tenant gpu clusters for deep learning workloads: Analysis and implications. Tech. rep., Microsoft Research, 2018.
[9] Jeon, M., Venkataraman, S., Phanishayee, A., Qian, J., Xiao, W., and Yang, F. Analysis of large-scale multi-tenant gpu clusters for dnn training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (2019).
[10] Li, M., Andersen, D. G., et al. Scaling distributed machine learning with the parameter server. In *Proc. of USENIX OSDI* (2014).
[11] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: an imperative style, high-performance deep learning library. In *NeurIPS* (2019).
[12] Patarasuk, P., and Yuan, X. Bandwidth optimal all-reduce algorithms for clusters of workstations. In *Journal of Parallel and Distributed Computing* (2009).
[13] Peng, Y., Bao, Y., Chen, Y., Wu, C., and Guo, C. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proc. of ACM EuroSys* (2018).
[14] Pjesivac-Grbović, J., Angskun, T., Bosilca, G., Fagg, G. E., Gabriel, E., and Dongarra, J. J. Performance analysis of mpi collective operations. *Cluster Computing* (2007), 127–143.
[15] Stavrinides, G. L., and Karatza, H. D. Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. In *Simulation Modelling Practice and Theory* (2011), vol. 19, pp. 540–552.
[16] Thakur, R., Rabenseifner, R., and Gropp, W. Optimization of collective communication operations in mpich. In *The International Journal of High Performance Computing Applications* (2005), vol. 19, pp. 49–66.
[17] Wang, Q., Shi, S., Wang, C., and Chu, X. Communication contention aware scheduling of multiple deep learning training jobs. In *arXiv:2002.10105* (2020).
[18] Xiao, W., Bhardwaj, R., Ramjee, R., Sivathanu, M., Kwatra, N., Han, Z., Patel, P., Peng, X., Zhao, H., Zhang, Q., Yang, F., and Zhou, L. Gandiva: Introspective cluster scheduling for deep learning. In *in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (2018), pp. 595–610.
[19] Yu, M., Ji, B., Rajan, H., and Liu, J. On scheduling ring-all-reduce learning jobs in multi-tenant gpu clusters with communication contention. In *Proc. ACM MobiHoc* (2022).
[20] Yu, M., Tian, Y., Ji, B., Wu, C., Rajan, H., and Liu, J. GADGET: Online resource optimization for scheduling ring-all-reduce learning jobs. In *IEEE INFOCOM* (2022).

A PROOF OF THEOREM 1

We let c_i be the completion time of i-th task, τ_i be its starting time and f_i be its processing time. The task c_{i-1} has a start time of τ_{i-1} and a finish time of $\tau_{i-1} + f_i$, which must be less than or equal to τ_i due to the precedence constraints. To satisfy the precedence constraints, we construct a series of tasks in the form of $1 \to 2 \to \ldots \to k-1 \to k$, with the task index starting from 1 (the first task without any predecessor). Since it is a precedence path, the makespan thus is lower-bounded by $T^* \geq \sum_{i=1}^{k} f_i$.

We next characterize the workload that can be processed between the finish time $\tau_i + f_i$ of one task of this chain and the starting time of the next task τ_{i+1} by discussing the following two cases: 1) If task i is a computation task and task $i+1$ is not executed immediately after it, then during the time interval $[\tau_i + f_i, \tau_{i+1})$, the communication processor is busy. 2) If task i is a communication task and then task $i+1$ will immediately get started since each job has an exclusive computation processor. We observe that the number of busy processors during this interval, denoted as H_i, is between 1 and J, depending on whether the model is computation dominant, i.e., reaches to J, or communication dominant, i.e., reaches to 1 in the worst case.

W define $H \triangleq \max_i H_i$, and calculate the maximum amount of workload that can be processed during the busy time as follows:

$$H_1\tau_1 + \sum_{i=1}^{k-1} H_i\big(\tau_{i+1} - (\tau_i + f_i)\big)$$

$$\leq H\Bigg(\tau_1 + \sum_{i=1}^{k-1}(\tau_{i+1} - (\tau_i + f_i))\Bigg) \leq H\sum_{i=1}^{k} f_i - \sum_{i=1}^{k-1} f_i \leq HT^* - \sum_{i=1}^{k-1} f_i,$$

which implies that $\tau_k \leq T^* + (1-\frac{1}{H})\sum_{i=1}^{k-1} f_i$. Since we have $T' = \tau_k + f_k$, then we have

$$T' \leq T^* + (1-\frac{1}{H})\sum_{i=1}^{k-1} f_i + f_k \leq (2-\frac{1}{H})T^*,$$

and the proof is complete.

B PROOF OF THEOREM 2

Note that for the special case that no jobs compete for the bandwidth, all scheduling policy will produce the same makespan, and thus the optimality trivially holds. Let $\mathcal{D}_t = \{l_1^t, l_2^t, \ldots, l_m^t\}$ and $\mathcal{D}'_t = \{l_{1'}^t, l_{2'}^t, \ldots, l_{m'}^t\}$ be the sets of remaining execution times of non-executed jobs at time slot t using our scheduling policy and any other arbitrary scheduling policy, respectively. Similarly, let $\mathcal{Q}_t = \{q_1^t, q_2^t, \ldots, q_n^t\}$ and $\mathcal{Q}'_t = \{q_{1'}^t, q_{2'}^t, \ldots, q_{n'}^t\}$ be the sets of remaining execution times of executed jobs using our scheduling policy and any other arbitrary scheduling policy, respectively, in time-slot t. Note that $m + n = m' + n'$. We re-index the jobs according to their remaining execution time in non-increasing order. We will prove the optimality of our scheduling policy by induction.

At time $t = t'$, when the first communication task scheduling is necessary due to contention, we prove that our scheduling policy in Algorithm 3 produces a smaller lower bound for the makespan than any other arbitrary scheduling strategy. Then, we have:

$$\max\{q_1^{t'}, l_1^{t'}\} \overset{(a)}{\leq} q_1^{t'} \overset{(b)}{=} q_{1^*}^{t'} \overset{\text{Def.}}{=} \max\{q_{1'}^{t'}, l_{1'}^{t'}\},$$

where (a) follows the fact that we greedily schedule jobs with "longest remaining execution time first," which implies that $q_1^{t'} \geq l_i^{t'}, i = 1, \ldots m$; and (b) follows the fact that t' is the first time slot to do the scheduling tasks, meaning the states of all the worker-conserving policies are the same. Thus, we have $q_1^{t'} = q_{1^*}^{t'}$, where $q_{1^*}^{t'} = \max\{q_{1'}^{t'}, l_{1'}^{t'}\}$. Therefore, it follows that $T^* \geq t' + \max\{q_1^{t'}, l_1^{t'}\}$ and $T' \geq t' + \max\{q_{1'}^{t'}, l_{1'}^{t'}\}$, since the extra waiting time may be added due to the contention.

We then suppose that at $t = t' + k$, where $k \in \mathbb{Z}^{++}$, our scheduling policy produces a smaller lower bound for makespan compared to any other arbitrary scheduling strategy. That is, we have:

$$\max\{q_1^{t'+k}, l_1^{t'+k}\} \leq q_1^{t'+k} \leq \max\{q_{1'}^{t'+k}, l_{1'}^{t'+k}\},$$

$$T^* \geq t' + k + \max\{q_1^{t'+k}, l_1^{t'+k}\} \text{ and } T' \geq t' + k + \max\{q_{1'}^{t'+k}, l_{1'}^{t'+k}\}.$$

Next, we prove that at $t = t' + k + 1$, the statement still holds. Since $l_1^{t'+k+1} \leq q_1^{t'+k+1} \leq q_1^{t'+k}$ and due to the hypothesis, we have

$$\max\{q_{1'}^{t'+k+1}, l_{1'}^{t'+k+1}\} \overset{(c)}{\geq} \max\{q_{1'}^{t'+k}, l_{1'}^{t'+k}\} - 1 \overset{\text{Assumption}}{\geq}$$
$$\max\{q_1^{t'+k}, l_1^{t'+k}\} - 1 \overset{(d)}{=} q_1^{t'+k+1} \overset{\text{Def.}}{=} \max\{q_1^{t'+k+1}, l_1^{t'+k+1}\},$$

where (c) follows the fact that in time-slot $t' + k + 1$, the arbitrary scheduling policy may not choose the task whose remaining chain length is the longest; and (d) is valid since Algorithm 3 follows the largest remaining execution time first policy. Therefore, we arrive at the conclusion that $\max\{q_1^t, l_1^t\} \leq \max\{q_{1'}^t, l_{1'}^t\}, \forall t$.

Lastly, we prove that $T^* \leq T'$ by contradiction. Suppose that $T^* > T'$. This implies that, in some time-slot t, we have $\max\{q_1^t, l_1^t\} > \max\{q_{1'}^t, l_{1'}^t\}$, contradicting to the relation $\max\{q_1^t, l_1^t\} \leq \max\{q_{1'}^t, l_{1'}^t\}$ we just proved. The proof is thus complete.