Prediction-Assisted Online Distributed Deep Learning Workload Scheduling in GPU Clusters

Ziyue Luo*, Jia Liu*, Myungjin Lee†, Ness B. Shroff*
* Dept. of ECE, The Ohio State University, USA, Email: luo.1457@osu.edu, {liu, shroff}@ece.osu.edu
† Cisco Research, USA, Email: myungjle@cisco.com

IEEE INFOCOM 2025 - IEEE Conference on Computer Communications | 979-8-3315-4305-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INFOCOM55648.2025.11044761

***Abstract*—The recent explosive growth of deep learning (DL) models has necessitated a compelling need for efficient job scheduling for distributed deep learning training with mixed parallelisms (DDLwMP) in GPU clusters. This paper proposes an adaptive shortest-remaining-processing-time-first (A-SRPT) scheduling algorithm, a novel prediction-assisted online scheduling approach designed to mitigate the challenges associated with DL cluster scheduling. By modeling each job as a graph corresponding to heterogeneous Deep Neural Network (DNN) models and their associated distributed training configurations, A-SRPT strategically assigns jobs to the available GPUs, thereby minimizing inter-server communication overhead. Observing that most DDLwMP jobs recur, A-SRPT incorporates a random forest regression model to predict training iterations. Crucially, A-SRPT maps the complex scheduling problem into a single-machine instance, which is addressed optimally by a preemptive "shortest-remaining-processing-time-first" strategy. This optimized solution serves as a guide for actual job scheduling within the GPU clusters, leading to a theoretically provable competitive scheduling efficiency. We conduct extensive real-world testbed and simulation experiments to verify our proposed algorithms.**

I. Introduction

Distributed deep learning (DDL) has recently achieved remarkable successes across multiple domains, *e.g.*, natural language processing (NLP) [1], computer vision [2], and computer networks [3]. However, the training of deep neural network (DNN) models is compute-intensive, requiring dedicated, powerful, and expensive GPU clusters [4], [5], [6], This has necessitated developing algorithms to efficiently schedule distributed deep learning training jobs with mixed parallelisms (DDLwMP), including but not limited to data parallelism [7], model parallelism [8] and pipeline parallelism [9]. Such scheduling algorithms are pivotal for resource allocations in GPU clusters to orchestrate DDLwMP jobs' execution.

In the areas of DDL scheduling algorithm design, many early attempts adopted a preemptive scheduling approach that permits pausing, resumption, and reallocation of running jobs for better flexibility. However, with ever-increasing learning model sizes, interrupting DDL job executions, including saving/loading training models into/from the host memory and potentially reallocating jobs to a different set of GPUs, incurs large overhead on the order of seconds to minutes [5]. To pursue improved resource utilization and consistent processing of DDL jobs, some recent studies have shifted their focus towards designing non-preemptive ML cluster scheduling algorithms [10], [11], [12], where the scheduler dedicates a set of GPUs solely for each DDLwMP job to ensure that all allocated GPUs execute simultaneously without interruption until the job's completion. However, all aforementioned works are designed for DDL jobs *without* mixed parallelisms. To date, designing scheduling algorithms for DDLwMP remains in its infancy and there are several highly non-trivial challenges:

1) DDLwMP jobs differ significantly in their model architectures, consisting of diverse types of DNN layers. The mixture of parallelisms results in complex computation and communication patterns during training. Thus, optimally placing DDLwMP jobs across the available GPUs, taking into account their model architectures and parallel paradigms, is highly challenging. Further, resource fragmentation (available GPUs are scattered across partially occupied servers due to frequent small job allocations) exacerbates the problem.

2) The unpredictability of future workloads introduces another challenge, rendering the scheduling task an online problem. Due to the non-preemption constraints, greedily scheduling existing jobs to fully occupy the cluster's computational resources can lead to fragmentation issues and significantly delay incoming jobs, thus increasing overall latencies. Thus, strategic orchestration of the available jobs is needed to minimize the total job completion time: the algorithm should schedule sufficiently many jobs to maximize resource utilization while reserving resources for future job arrivals.

3) Many existing non-preemptive scheduling designs require the knowledge of training iterations upon jobs' submissions to estimate job training durations. However, DNN model training is a feedback-dependent exploration process [13]. It is common for users to submit multiple jobs exploring different configurations of hyper-parameters and terminate most jobs due to random errors or sub-optimal convergence performance [6],

This work has been supported in part by Cisco Research Award PO-USA000EP312336, by NSF grants CAREER CNS-2110259, IIS-2324052, NSF AI Institute (AI-EDGE) CNS-2112471, CNS-2312836, CNS-2106933, CNS-2106932, CNS-1955535, and CNS-1901057, by DARPA YFA D24AP00265, by ONR grant N00014-24-1-2729, by AFRL grant PGSC-SC-111374-19s, by Army Research Office under Grants W911NF-21-1-0244 and W911NF-24-2-0205, and was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-23-2-0225. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

[14]. This implies that the actual number of job training iterations is *uncertain*. Blindly scheduling jobs according to the user-specified training iterations could lead to suboptimal performance.

To address these challenges, in this paper, we propose an adaptive shortest-remaining-processing-time-first (A-SRPT) scheduling algorithm. Our design contains two key components: 1) a **GPU mapping** algorithm that judiciously assigns a DDLwMP job to a specific set of GPUs, thereby minimizing the data communication overhead during job training; and 2) a **prediction-assisted online scheduling** algorithm that strategically schedules DDLwMP jobs by incorporating a job total training iteration prediction model. Our main contributions and key results are summarized as follows:

- We represent DDLwMP jobs with various models and distributed training configurations as graphs, based on which we further develop the Heavy-Edge algorithm, a graph-cut-based method designed to strategically allocate each job to available GPUs across servers. Heavy-Edge emphasizes maximizing the use of high-bandwidth interconnection for GPUs within a server (*e.g.*, NVLink [15]), thereby improving overall training efficiency.
- We tackle the uncertain training duration challenge by leveraging the recurrence of DDLwMP jobs. First, we use a random forest regression method [16] to predict training iterations from historical job execution traces. Then, by leveraging this prediction model, we develop a prediction-assisted online scheduling framework called A-SRPT based on a *two-step* approach: 1) We show that the original complex multi-dimensional GPU clustering problem can be simplified as a preemptive *single-machine* scheduling problem with the predicted number of training iterations for each DDLwMP job. This simplification enables the use of the shortest remaining processing time (SRPT) principle [17], which is *optimal* in scheduling jobs in the hypothetical single-machine problem; 2) We use the virtual single-machine SRPT solution to guide our non-preemptive scheduling decisions for DDLwMP jobs in the actual cluster. This two-step approach allows us to design DDLwMP scheduling schemes with theoretical performance guarantee.
- To validate the effectiveness of our proposed designs, we conduct *real-world* trace-driven testbed experiments and simulation studies based on profiled DDL workloads with mixed DNN models and a two-month DL workload trace [6]. Our experimental results verify the superiority of our proposed algorithms over state-of-the-art DDL scheduling algorithms. Specifically, our proposed algorithm outperforms all baseline designs and achieves up to 92% total job completion time reduction.

II. Background and Related Work

1) Parallelisms for Distributed DNN Training: DNN training is an iterative process to minimize a loss function [18], where each iteration consists of forward propagation (FP), backward propagation (BP), and gradient update, all of which are based on mini-batches. The advent of large DNN models has driven the development of distributed DNN training to speed up DNN training. To enable distributed DNN training, data [7], model [8], and pipeline parallelisms [9], [19], [20], as shown in Fig. 1, are the most common.

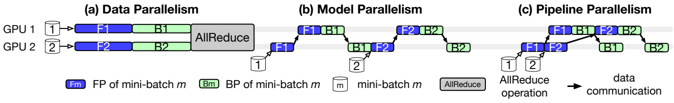

Fig. 1: Three typical parallelisms for distributed DNN training.

Data parallelism (Fig.1(a)) trains mini-batches on different GPUs, followed by gradient synchronization using ring AllReduce (RAR) [21] or tree AllReduce (TAR) [22]. RAR forms a logical ring for communication [23], while TAR uses double binary trees [22] (e.g., NVIDIA NCCL [24]). This method requires each GPU to host a full DNN model, limiting it to small-size models. Model parallelism (Fig. 1(b)) trains large models by distributing FPs and BPs across GPUs, each hosting a different model stage. However, model parallelism suffers from low utilization as only one GPU is active at a time.

Building on model parallelism, pipeline parallelism (Fig.1(c)) sequentially injects mini-batches into the system to allow simultaneous GPU processing. Each model stage can have multiple replicas [25], [20] trained with data parallelism to reduce stage processing time. Pipeline parallelism can be further divided into asynchronous and synchronous pipelines. Synchronous pipeline [19], [20] maintains a synchronization barrier between training iterations, enforcing synchronous gradient updates across all model stages to achieve a better convergence performance. However, such synchronization barriers may interrupt the pipeline and delay new mini-batch entries, leading to low GPU utilization. Asynchronous pipeline [26] improves GPU utilization by continuously injecting mini-batches to increase training throughput at the price of (slight) model convergence degradation [9]. In this work, we consider asynchronous pipeline due to its higher training efficiency.

2) Online DDL Job Scheduling: Early attempts on online DDL job scheduling focused on preemptive algorithms. For instance, Optimus [27] constructs resource-performance models for dynamic GPU scaling to minimize completion time of data-parallel jobs. Gandiva [4] uses scaling heuristics for GPU-sharing across multiple jobs. GADGET [23] balances communication overhead and contention for resource scheduling for RAR jobs. Tiresias [14] prioritizes jobs based on training duration metrics. Pollux [5] adapts resources to optimize *good-put*, a metric combining throughput and statistical efficiency. Non-preemptive scheduling research is more limited. SPIN [10] focuses on minimizing makespan for placement-sensitive jobs. An online framework in [11] addresses communication contention among DDL jobs. An offline approximation algorithm in [12] tackles communication overhead and network contention for RAR jobs. However, all existing methods above only considered a *single* parallelism. By stark contrast, in this work, we propose a *non-preemptive* online scheduling algorithm for DDLwMP DDL training jobs with theoretical performance guarantees.

It is worth noting that most previous DDL job scheduling works rely on the knowledge of job training duration/iterations (some using predictive techniques based on historical runtimes [10], [14], [27]). Abdullah *et al.* [28] proposed to enhance ML job completion predictability using weighted-fair-queueing for bounded preemption. However, prioritizing jobs by predicted execution time can lead to inaccurate GPU allocation and long wait times for short jobs. Inspired by recent advances in learning-augmented online preemptive scheduling for single machine [29], we propose an online prediction-assisted algorithm for non-preemptive DDLwMP job scheduling to delay long jobs to expedite shorter ones.

III. System Model

We consider a GPU cluster consisting of M inter-connected homogeneous GPU servers. Each server $m \in [M]$[1] is equipped with g GPUs, yielding a total of $G = Mg$ GPUs within the cluster. The bidirectional (*i.e.*, incoming and outgoing) NIC bandwidth on each machine is denoted as $B_{\mathtt{inter}}$. The intra-server bidirectional GPU communication bandwidth (*e.g.*, PCIe, and NVLink [15]) is denoted as $B_{\mathtt{intra}}$, which is typically one to two orders of magnitude greater than $B_{\mathtt{inter}}$. The system works in a time-slotted fashion, over a potentially large span of T time-slots. There are I DDLwMP jobs in total in the cluster, and job $i \in [I]$ is submitted at time $r_i \in [T]$. We note that our proposed scheduling algorithm for DDLwMP jobs also includes single-GPU jobs as a special case, thus offering general support for all DDL workloads. In what follows, we zoom into two key components in our system modeling.

A. Workload Scheduling for DDL Jobs in GPU Cluster

In our DDLwMP training setting, each job $i \in [I]$ requests to train a DNN model $\mathcal{D}_i$ for n_i iterations using a specific distributed configuration. $\mathcal{D}_i$ is divided into S_i stages, each of which consists of some consecutive DNN layers. For improved training efficiency, stages can further be replicated across multiple GPUs in a data-parallel fashion [9], [25], allowing varying degrees of data-parallelism across different stages. The processing of a single mini-batch by a stage is distributed over the GPUs. Let $k_{i,s}$ denote the number of data-parallel replicas for stage $s \in [S_i]$ of job i, which equals the required GPUs for this stage. Thus, the total GPUs needed for job i is $g_i = \sum_{s\in[S_i]} k_{i,s}$. A single-GPU job is a special case with one non-replicated stage. Our distributed training configuration covers the following parallelisms as special cases: 1) data parallelism (single-stage, multiple replicas), 2) model parallelism (multiple non-replicated stages), and 3) pipeline parallelism (other cases). We assume parallel configurations are given through pipeline planning [30], [20].

On a given GPU, the time required for the FP (resp. BP) of a mini-batch over a replica of stage s for job i is denoted by $p^f_{i,s}$ (resp. $p^b_{i,s}$). The incoming and outgoing data size (*i.e.*, activations during FP and gradients during BP) for each training iteration per replica of stage s in job i are denoted by $d^{in}_{i,s}$ and $d^{out}_{i,s}$ respectively. We use $h_{i,s}$ to represent the size of trainable parameters for job i and stage s.

We use $x^m_{i,s}$ to represent the number of GPUs allocated on server m to host stage s of job i, and use t_i to denote the starting time of job i. Accordingly, an amount of $x^m_{i,s}/g$ bandwidth for the stage is reserved at the incoming and outgoing NIC. Let $\alpha_i(\{x^m_{i,s}\})$ represent the per-iteration training time of job i given its GPU allocation $\{x^m_{i,s}\}$, which will often be simplified as α_i for notational simplicity henceforth if no confusion arises from the context. The characterization of α_i will be specified later in this section. To ensure schedule feasibility, we have the following constraints:

$$t_i \geq r_i, \forall i \in [I], \tag{1}$$

$$\sum_{m\in[M]} x^m_{i,s} = k_{i,s}, \forall i \in [I], s \in [S_i], \tag{2}$$

$$\sum_{i\in[I]:t_i\leq t\leq t_i+n_i\alpha_i} \sum_{s\in[S_i]} x^m_{i,s} \leq g, \forall m \in [M], t \in [T]. \tag{3}$$

Here, Constraint (1) ensures that each job is scheduled to start only after its submission; Constraint (2) implies that all stage replicas of job i are allocated in the cluster; and Constraint (3) guarantees that the allocated GPUs for active jobs do not exceed each server's capacity limit.

B. Characterization of Per-Iteration Training Time α_i

As mentioned in Section II, we focus on the widely adopted asynchronous pipeline parallel training [9], [26]. We note that our design can be straightforwardly extended to synchronous pipeline parallelism [19] by following the analytic model proposed in [20] for α_i. Under asynchronous pipeline parallelism, as the execution of all stages is fully pipelined, the per-iteration training time is the maximum per-stage computation-communication time of a single stage (*i.e.*, the bottleneck stage) [9], [30]. We use $\beta^m_{i,s}$ to denote the per-iteration training time of stage s of job i on machine m, which consists of the computation time for the current batch of the stage replicas on server m in one iteration (denoted as $comp^m_{i,s}$), the data communication time for sending activations and gradients of the current batch into and out of the stage (denoted as $comm^m_{i,s}$), and the communication costs for synchronizing parameters among all the stage replicas using AllReduce operations ($AllReduce^m_{i,s}$). The communication time (including both the FP and the BP) can be calculated as follows:

$$comp^m_{i,s} = \begin{cases} p^f_{i,s} + p^b_{i,s}, & x^m_{i,s} > 0, \\ 0, & x^m_{i,s} = 0. \end{cases} \tag{4}$$

To compute the inter-stage communication time when stage $s-1$ and/or s are replicated over multiple GPUs, we evenly distribute the data being transmitted across inter-stage links. Thus, the per-iteration data communication time between each replica of stage $s-1$ and s is $\frac{2d^{out}_{i,s-1}}{k_{i,s}} = \frac{2d^{in}_{i,s}}{k_{i,s-1}}$ [20]. Hence, for stage $s \in [2, 3, \ldots, S_i - 1]$, if $x^m_{i,s} > 0$, we have:

[1] We use $[X]$ to denote the set $\{1, 2, \ldots, X\}$.

$$comm^m_{i,s} = \frac{(2d^{in}_{i,s}\frac{k_{i,s-1}-x^m_{i,s-1}}{k_{i,s-1}} + 2d^{out}_{i,s}\frac{k_{i,s+1}-x^m_{i,s+1}}{k_{i,s+1}})x^m_{i,s}}{(x^m_{i,s}/g)B_{\mathtt{inter}}} + \frac{2d^{in}_{i,s}\frac{x^m_{i,s-1}}{k_{i,s-1}} + 2d^{out}_{i,s}\frac{x^m_{i,s+1}}{k_{i,s+1}}}{B_{\mathtt{intra}}}, \quad (5)$$

and $comm^m_{i,s} = 0$ otherwise. The term $comm^m_{i,s}$ for the first and last stages can be calculated similarly. The data size communicated for each stage replica of stage s in the AllReduce operation can be calculated as $\frac{2(k_{i,s}-1)}{k_{i,s}}h_{i,s}$ [31] for both RAR and TAR, and the data communication time is bottlenecked by the minimum bandwidth between stage replicas. Hence, the time taken by the AllReduce operation for job i stage s is:

$$AllReduce^m_{i,s} = \begin{cases} \frac{2(k_{i,s}-1)h_{i,s}}{k_{i,s}\frac{x^m_{i,s}}{g}B_{\mathtt{inter}}}, & \text{if } x^m_{i,s} < k_{i,s}, \\ \frac{2(k_{i,s}-1)h_{i,s}}{k_{i,s}B_{\mathtt{intra}}}, & \text{if } x^m_{i,s} = k_{i,s}. \end{cases} \quad (6)$$

Here, in the first case, the bottleneck is due to the server NIC bandwidth, while in the second case, all data communication is conducted via the intra-server connection. Lastly, by putting all things together and in line with existing formulations on pipeline scheduling [25], [30], [20], we obtain the per-iteration training time α_i for processing a single batch as follows:

$$\begin{aligned} \alpha_i &= \max_{m\in\mathcal{M}, s\in[S_i]} \beta^m_{i,s} \\ &= \max_{m\in\mathcal{M}, s\in[S_i]} (comp^m_{i,s} + comm^m_{i,s} + AllReduce^m_{i,s}). \quad (7) \end{aligned}$$

Additionally, some distributed communication engines (*e.g.*, BytePS [32]) enable strategic overlapping of AllReduce operations with backward computation. For example, gradients for layer l can be synchronized using AllReduce while simultaneously computing gradients for layer $l-1$. To account for this overlapping, one can apply model-dependent coefficients to the backward computation time and AllReduce time [33].

Let $\alpha_i^{\max}$ and $\alpha_i^{\min}$ denote the maximum and minimum per-iteration training times of job i given a GPU assignment, respectively. $\alpha_i^{\max}$ can be computed using Eq. (7) if the job is assigned to g_i servers, with each server holding a single-stage replica and assigned a bandwidth of $1/g \times B_{\mathtt{inter}}$. However, evaluating $\alpha_i^{\min}$ for each job requires searching through an exponential number of possible GPU assignments, which is computationally intractable. To address this challenge, we will propose an estimation strategy to be described in Sec. IV-B.

C. The Online DDLwMP Job Scheduling Problem

In this paper, our goal is to minimize the total DDLwMP job completion in a time horizon of length T, which can be evaluated as $\sum_{i\in[I]}(t_i + n_i\alpha_i)$. Putting all modeling together, we can formulate our DDLwMP job scheduling problem as:

$$\begin{aligned} &\text{Minimize } \sum_{i\in[I]}(t_i + n_i\alpha_i) \quad (8) \\ &\text{subject to (1)–(3)}, x^m_{i,s} \in \mathbb{N}, \forall m \in [M], i \in [I], s \in [S_i], \\ &\qquad t_i \in [T], \forall i \in [I]. \end{aligned}$$

We note that Problem (8) is an integer non-convex program due to the intricate modeling of the per-iteration training time α_i. Moreover, another key challenge in Problem (8) stems from the uncertain job submission time r_i and the unknown number of job training iterations n_i, which necessitates online algorithmic designs. In fact, the offline variant of Problem (8), where r_i, n_i and α_i are all predetermined (rendering the problem of scheduling parallelizable tasks [34]) is NP-hard. To address these challenges, we propose a prediction-assisted algorithm for optimizing the online DDLwMP job scheduling in GPU clusters.

IV. Prediction-Assisted Online Job Scheduling Algorithm

A. Basic Idea

The complexities of Problem (8) arise from two distinct perspectives: 1) The sensitivity of DDLwMP jobs to GPU placement (the per-iteration training time, can significantly vary with different placements); and 2) the inherent online nature of the problem (not only are the job arrival times unknown, but the actual number of job execution iterations is also typically uncertain in practice).

To address these unique challenges, we introduce a new online scheduling algorithm named adaptive shortest-remaining-processing-time-first (A-SRPT) to solve Problem (8) based on the following key observations: First, we note that the complex computation-communication structure of DDLwMP jobs can be effectively modeled using graphs. This realization leads us to develop a strategic graph partitioning algorithm called Heavy-Edge. This algorithm favors co-locating replicas with substantial communication requirements, thereby enhancing overall scheduling efficiency.

Utilizing Heavy-Edge for job placement, we propose an online job scheduling framework for DDLwMP jobs with a theoretical competitive ratio guarantee. This framework is inspired by the proven optimality of preemptive Shortest Remaining Processing Time (SRPT) scheduling for jobs based on their predicted durations on a single machine [29]. In our approach, we construct a single-machine preemptive scheduling instance based on the original non-preemptive scheduling problem. This construction considers the size of each job and its predicted number of training iterations.

We then apply SRPT to preemptively schedule these jobs within this hypothetical single-machine instance. The results obtained from this single-machine scheduling model are then used to guide the non-preemptive job allocation in the actual cluster environment. In this way, jobs with larger predicted workloads are scheduled later, creating space for potentially future smaller jobs to be scheduled first, thus reducing the total job completion time.

B. The Heavy-Edge GPU Mapping Algorithm

In our Heavy-Edge GPU mapping algorithm, each job i is assigned to a set of servers $\mathcal{M}_i$ for execution. Each server $m \in \mathcal{M}_i$ has g_m available GPUs to host job i's stage replicas, such that $\sum_{m\in\mathcal{M}_i} g_m = g_i$ ($g_m \le g$ as some GPUs in the

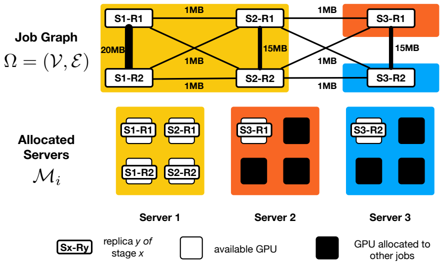

Fig. 2: GPU mapping: An illustrative example.

server may be occupied by existing jobs). We now map each stage replica of job i to a GPU, with the goal to reduce inter-server communication to improve job training efficiency.

Toward this end, we model each job i as a graph $\Omega = (\mathcal{V}, \mathcal{E})$, where vertices $\mathcal{V}$ represent stage replicas and edges $\mathcal{E}$ denote data communication, with edge weights indicating communication data size. For inter-stage communication between stages $s-1$ and s, we assign edges with weight $\frac{2d^{out}_{i,s-1}}{k_{i,s}} = \frac{2d^{in}_{i,s}}{k_{i,s-1}}$ for each replica pair. For intra-stage communication (AllReduce) in stage s, we handle RAR and TAR differently. In RAR, replicas form a ring with edges weighted $\frac{2(k_{i,s}-1)}{k_{i,s}} h_{i,s}$. For TAR, edges connect replica pairs linked in double binary trees, weighted $\frac{(k_{i,s}-1)}{k_{i,s}} h_{i,s}$, which is halved compared to RAR. This reduction is due to the structure of the double binary trees, where each tree processes half of the total data [24].

As a result, the GPU mapping problem is equivalent to a graph cut problem that partitions a graph into $|\mathcal{M}_i|$ subgraphs of size g_m to minimize inter-server communication (total cut weight among subgraphs) and maximize intra-server communication (total edge weights within subgraphs). Fig. 2 illustrates an example of GPU mapping. The job consists of three stages, each with two replicas. The job is assigned to three servers with four, one, and one available GPU(s), respectively. We partition the job graph into three subgraphs, each corresponding to the set of GPUs in a server with the same color. Unfortunately, this graph partitioning problem is an NP-complete balanced graph cut problem [35] even with equal GPUs per server, and not to mention with varying GPU availability. To address this challenge, we propose the Heavy-Edge approach, which greedily assigns heavily connected stage replicas to servers as follows.

In Heavy-Edge, we start by sorting the servers in $|\mathcal{M}_i|$ based on the available GPU numbers in a descending order, denoted as $\{m_1, m_2, \ldots, m_{\mathcal{M}_i}\}$. Vertices in $\mathcal{V}$ (i.e., stage replicas) are then assigned to these servers from m_1 to $m_{\mathcal{M}_i}$. We denote the current server for assignment as m and use `node_set` to denote the set of vertices assigned to m, which is initialized as $\emptyset$. Next, we consider two cases: 1) if $|\mathcal{V}|$ equals m's GPU count, all replicas are assigned to it; 2) for single-GPU servers, we assign the vertex with the minimum total edge weight. In the case of a server with multiple GPUs and there are remaining vertices, the GPU mapping process follows the "Heavy-Edge" principle: we iteratively add vertices to `node_set` by finding the heaviest edge between assigned and unassigned vertices, prioritizing intra-server communication efficiency. If no connecting edge exists, we randomly assign an unassigned vertex. This process continues until `node_set` matches m's number of available GPUs.

We use an example in Fig. 2 to further illustrate our Heavy-Edge GPU mapping algorithm. The process begins by identifying the heaviest communication edge, (`S1-R1`, `S1-R2`), with a data size of 20MB, and assigning these nodes to `node_set`, *i.e.*, the first server. To optimize communication efficiency, we then allocate unassigned nodes directly connected to this pair (*i.e.*, `S2-R1` and `S2-R2`), each with a 1MB connection to `S1-R1` and `S1-R2` respectively, to the same server, maximizing intra-server communication. This process continues sequentially for subsequent servers until all nodes are assigned, effectively minimizing inter-server communication overhead.

With the Heavy-Edge GPU mapping algorithm, we obtain the minimum achievable per-iteration training time $\tilde{\alpha}_i^{\min}$ for jobs, helping predict job training times. To minimize per-iteration time, each job is allocated to the fewest servers possible, utilizing the maximum number of interconnected high-bandwidth GPUs. For job i, a set of machines $\mathcal{M}_i$ is assigned, where servers m_1 to $m_{|\mathcal{M}_i|-1}$ contribute all g GPUs, and the last server $m_{|\mathcal{M}_i|}$ contributes $g' \leq g$ GPUs. Heavy-Edge determines the GPU mapping, and $\tilde{\alpha}_i^{\min}$ is estimated using (7).

C. The A-SRPT Online DDLwMP Job Scheduling Algorithm

1) Adaptive Shortest-Remaining-Processing-Time-First: Our online scheduling algorithm is inspired by the online SRPT framework proposed in [36], which is optimal for online scheduling for single-machine jobs with known durations over parallel machines. However, our problem is far more complex due to two critical aspects: 1) Each job in our setting can span *multiple* GPUs, inducing complex inter-job communication patterns; 2) The actual number of training iterations of jobs in our setting becomes known only upon job completion. Assume that we have a prediction model that predicts the number of training iterations $\tilde{n}_i$ for each training job i. We define the prediction error for job i, denoted by ϵ_i, as the total absolute difference between the predicted and actual numbers of training iterations, *i.e.*, $|n_i - \tilde{n}_i|$. Let ϵ and $\bar{\epsilon}$ denote the total prediction and average prediction errors, respectively, which can be computed as:

$$\epsilon = \sum_{i \in [I]} \epsilon_i = \sum_{i \in [I]} |n_i - \tilde{n}_i|, \quad \text{and} \quad \bar{\epsilon} = \frac{\epsilon}{I}. \tag{9}$$

Our proposed design adopts the Shortest Remaining Processing Time (SRPT) strategy, which prioritizes available jobs with the least processing time. This approach is known to be delay-optimal in single-machine preemptive settings [17] and has been proven competitive even when job processing times are unknown until completion but can be estimated [29].

We present an overview of our algorithmic idea in Fig. 3. We "virtualize" the entire GPU cluster as a 'single machine' and proportionally scale down each job's workload (①). Specifically, let instance A denote the original online DDLwMP scheduling problem. We then define a new hypothetical single-machine preemptive online scheduling problem A_1,

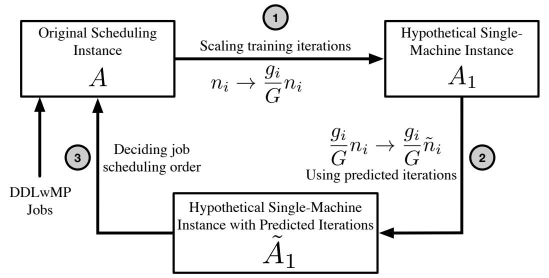
Original Scheduling Instance
A
1
Scaling training iterations
$n_i \rightarrow \frac{g_i}{G} n_i$
Hypothetical Single-Machine Instance
A_1
$\frac{g_i}{G} n_i \rightarrow \frac{g_i}{G} \tilde{n}_i$
2
Using predicted iterations
3
Deciding job scheduling order
DDLwMP Jobs
Hypothetical Single-Machine Instance with Predicted Iterations
$\tilde{A}_1$

Fig. 3: Algorithmic idea overview.

sharing A's job set. In A_1, the number of training iterations for job i is scaled to $\frac{g_i}{G} n_i$, while the arrival time r_i is kept unchanged. As the actual per-iteration training time α_i of a job can only be obtained after placement, to estimate the job's GPU requirements and its minimum attainable per-iteration training time, we optimistically employ the minimum per-iteration training time $\tilde{\alpha}_i^{\min}$, which is determined in the previous section. Thus, the job duration in instance A_1 is calculated as $\frac{g_i}{G} n_i \tilde{\alpha}_i^{\min}$. Furthermore, since the actual number of job training iterations n_i is unknown at the time of scheduling, we introduce another instance, $\tilde{A}_1$. This instance substitutes A_1's training iteration number, $\frac{g_i}{G} n_i$, with the predicted value, $\frac{g_i}{G} \tilde{n}_i$ (②). Consequently, the predicted job duration in $\tilde{A}_1$ is represented as $\frac{g_i}{G} \tilde{n}_i \tilde{\alpha}_i^{\min}$. We schedule all jobs in $\tilde{A}_1$ first, and order jobs according to their completion time in $\tilde{A}_1$. We then perform job scheduling on the actual cluster following the order (③). By doing so, jobs with larger predicted workloads $\frac{g_i}{G} \tilde{n}_i \tilde{\alpha}_i^{\min}$ are scheduled later due to longer completion times in $\tilde{A}_1$. Therefore, the goal of A-SRPT is to create space for potentially future smaller jobs to be scheduled first, thus reducing the total job completion time.

Our A-SRPT algorithm is detailed in Algorithm 1. The job completion order in $\tilde{A}_1$ is maintained in `pending_queue`. Let i denote the current head of `pending_queue`, *i.e.*, the job to be scheduled. If the number of GPUs required by job i (*i.e.*, g_i) is less than or equal to the available number of GPUs in the cluster, job i can be scheduled (Line 5), and removed from `pending_queue` (Line 7). Otherwise, we proceed to the next time-slot (Line 25).

To further improve resource utilization, we classify jobs as either "communication-heavy" or "non-communication-heavy," thereby tailoring the scheduling policy to each job's communication pattern. The *rationale* behind this strategy is that communication-heavy jobs have per-iteration training times highly sensitive to GPU mapping due to large communication data sizes, making their worst-case training time $\alpha_i^{\max}$ (with inter-server bandwidth $B_{\texttt{inter}}$) much higher than when allocated to the fewest possible servers. Jobs are classified by the ratio $\alpha_i^{\max}/\tilde{\alpha}_i^{\min}$. If this ratio exceeds `COMM_HEAVY` (1.5 in our experiments), the job is communication-heavy; otherwise, it is non-communication-heavy. Communication-heavy jobs are delayed until sufficient server resources are available, while non-communication-heavy jobs are initiated immediately to maintain workflow efficiency.

For communication-heavy jobs, we prioritize server consolidation (Lines 8–20). We select servers based on maximum availability and calculate $\alpha_i(x_{i,s}^m)$. If $\alpha_i(x_{i,s}^m)/\tilde{\alpha}_i^{\min} \leq$ `COMM_HEAVY`, we schedule immediately. Otherwise, we delay up to $\tau \frac{g_i}{G} \tilde{n}_i \tilde{\alpha}_i^{\min}$, constantly reassessing allocations for a more efficient configuration, *i.e.*, a lower α_i.

Algorithm 1: The A-SRPT Algorithm.

```
   Input: $I, \{S_i\}, \{k_{i,s}\}, g_i, \{p_{i,s}^f, p_{i,s}^b\}, \{d_{i,s}^{in}, d_{i,s}^{out}\}, \{h_{i,s}\}, M, g, B_{\texttt{inter}}, B_{\texttt{intra}}$
   Output: $\{t_i, \{x_{i,s}^m\}\}_{i \in [I]}$
1  while $t \leq T$ do
2      Append completed jobs in $\tilde{A}_1$ using SRPT to pending_queue
3      while pending_queue is not empty do
4          $i \leftarrow$ head of pending_queue
5          if $g_i \leq$ available number of GPUs in the cluster then
6              $\mathcal{M}_i \leftarrow \emptyset$
7              Pop $i$ from pending_queue
8              if $\alpha_i^{\max}/\tilde{\alpha}_i^{\min} \geq$ COMM_HEAVY then
9                  Select $g_i$ GPUs from servers with most available GPUs; $\mathcal{M}_i \leftarrow$ these servers
10                 $\{x_{i,s}^m\} \leftarrow$ Heavy-Edge$(i, \mathcal{M}_i)$
11                 Calculate $\alpha_i(\{x_{i,s}^m\})$ using (7)
12                 if $\alpha_i(\{x_{i,s}^m\})/\tilde{\alpha}_i^{\min} \leq$ COMM_HEAVY then
13                     $t_i \leftarrow t$
14                 else
15                     $\kappa \leftarrow \alpha_i(\{x_{i,s}^m\})$
16                     for $t \in \{t+1, \ldots, t + \tau \frac{g_i}{G} \tilde{n}_i \tilde{\alpha}_i^{\min}\}$ do
17                         Calculate $\{x_{i,s}^m\}$ and $\alpha_i$ based on current server availability
18                         if $\alpha_i < \kappa$ then
19                             $t_i \leftarrow t$; break
20                     $t_i \leftarrow t$
21             else
22                 Select $g_i$ GPUs from servers with least available GPUs; $\mathcal{M}_i \leftarrow$ these servers
23                 $\{x_{i,s}^m\} \leftarrow$ Heavy-Edge$(i, \mathcal{M}_i)$; $t_i \leftarrow t$
24         else
25             $t \leftarrow t + 1$
26 return $\{t_i, \{x_{i,s}^m\}\}_{i \in [I]}$
```

For non-communication-heavy jobs, we prioritize immediate execution using a fragmentation-aware strategy (Lines 21–23). Since their per-iteration training times are less affected by placement, we allocate them to servers with lower availability, reserving higher-availability servers for communication-heavy jobs. We then use the Heavy-Edge algorithm for GPU mapping and promptly initiate the job.

2) Theoretical Performance Analysis: Let Γ_A denote the total job completion time achieved by A-SRPT for the GPU cluster scheduling problem A, and let OPT_A represent the true optimal job completion time. Also, let OPT_{A_1} and $OPT_{\tilde{A}_1}$ denote the total job completion times of the SRPT-based schedules for instances A_1 and $\tilde{A}_1$, respectively. Due to space limitation, we omit the proofs of Lemma 1–3 in this paper.

Lemma 1. *$OPT_{A_1} \leq \rho OPT_A$, where $\rho = \max_{i \in [I]} \frac{\alpha_i^{\max}}{\alpha_i^{\min}}$.*

Lemma 2. *Γ_A is no larger than*

$$(1 + \tau + \frac{\rho G}{G - g^{\max}}) OPT_{\tilde{A}_1} + I \frac{g^{\max} \alpha^{\max}}{G - g^{\max}} \epsilon + \rho OPT_A,$$

where $g^{\max} = \max_{i\in[I]} g_i$, *and* $\alpha^{\max} = \max_{i\in[I]} \alpha_i^{\max}$

Lemma 3. $OPT_{\tilde{A}_1} \leq OPT_{A_1} + I\frac{g^{\max}\alpha^{\max}}{G}\epsilon$.

Then, the total job completion time performance result of A-SRPT immediately follows from Lemmas 1–3:

Theorem 1 (Total job completion time achieved by A-SRPT)**.** Γ_A *is no larger than*

$$(2+\tau+\frac{\rho G}{G-g^{\max}})\rho + \frac{2\rho g^{\max}\alpha^{\max}}{\alpha^{\min}}(1+\tau+\frac{(1+\rho)G}{G-g^{\max}})\bar{\epsilon}$$

times the optimal job completion time OPT_A*, where* $\alpha^{\min} \triangleq \min_{i\in[I]} \alpha_i^{\min}$.

Proof. Combining Lemmas 1, 2 and 3 yields:

$$\Gamma_A \leq (1+\tau+\frac{\rho G}{G-g^{\max}})OPT_{\tilde{A}_1} + I\frac{g^{\max}\alpha^{\max}}{G-g^{\max}}\epsilon + \rho OPT_A \leq$$
$$(2+\tau+\frac{\rho G}{G-g^{\max}})\rho OPT_A + I g^{\max}\alpha^{\max}\left[\frac{1+\tau}{G}+\frac{1+\rho}{G-g^{\max}}\right]\epsilon.$$

Assuming each job runs at least one iteration, we have $\rho OPT_A \geq OPT_{A_1} \geq \sum_{i=1}^{I}(i\times\alpha^{\min})/G = \alpha^{\min}\frac{I(I+1)}{2G}$. It then follows that

$$\frac{\Gamma_A}{OPT_A} < \left[2+\tau+\frac{\rho G}{G-g^{\max}}\right]\rho + 2\rho g^{\max}\bar{\rho}\left[1+\tau+\frac{(1+\rho)G}{G-g^{\max}}\right]\bar{\epsilon},$$

where $\bar{\rho} \triangleq \frac{\alpha^{\max}}{\alpha^{\min}}$. This completes the proof. □

We remark that our competitive ratio bound is for the worst-case scenario. In this scenario, it is assumed that all jobs could potentially be executed with the maximum per-iteration training time $\alpha_i^{\max}$, which rarely happens in practice. Our numerical evaluations based on real-world data traces and popular DNN models show that the performance of A-SRPT is much better than the worst-case competitive ratio bound suggests. Also, Theorem 1 says that the performance of A-SRPT is closely tied to the average error of the employed prediction model. In what follows, we propose an efficient prediction model that provides robust estimates based on the actual characteristics of the jobs.

3) Random Forest Based Prediction: Studies show that most DDL jobs are recurrent, with nearly 65% submitted at least five times over two months [6]. This recurrence provides the opportunity for GPU cluster to perform prediction based on repeated job submissions by applying a hashing function to meta-information (e.g., user details, training dataset, and command-line script), thus generating a unique `group id` for recurrent jobs. Leveraging `group id` and historical job data, we employ random forest regression [16] with mean squared error for tree splitting to predict training iterations based on `group id` and user id. We predict 0 iterations for unseen jobs, treating them as immediately complete in $\tilde{A}_1$ and adding them directly to `pend_queue` for swift execution, reducing wait times and enhancing efficiency. We use 100 trees in our random forest regression. The high efficiency of forest regression allows frequent retrainings (hourly/daily) for accurate predictions. Training with a two-month trace of 700,000 DDLwMP jobs [6] takes only 80 seconds. Combined with A-SRPT, our prediction model enables efficient resource allocation and job scheduling in GPU clusters.

V. Performance Evaluation

In this section, we conduct both *real-world* data-trace-driven testbed experiments and simulation studies to evaluate the performance and efficacy of our proposed A-SRPT algorithm.

A. Real-World GPU Cluster Testbed Experiments

1) System Settings: *1-a) Implementation and Testbed:* We implement A-SRPT using Python and PyTorch 2.1.1 [37] with 4634 lines of code. The evaluation of A-SRPT is conducted on a single server equipped with two NVIDIA H100 NVL GPUs. To simulate a GPU cluster, we utilize the Multi-Instance GPU (MIG) [38] technique, partitioning the two H100 GPUs into 14 virtual GPUs (vGPUs), each with 12 GB of GPU memory. The scheduling overhead per job is within 5s. Due to the MIG configurations, inter-vGPU communication is limited to the PCIe bandwidth of 128 GB/s. Consequently, GPU mapping does not significantly impact our testbed experiment. Therefore, we set the delay factor to zero in A-SRPT. For more heterogeneous inter-GPU networks, we evaluate the performance of A-SRPT in the simulation studies later in this section.

1-b) Deep Learning Workload: The dataset for our job analysis is obtained from an open-source two-month deep learning workload trace collected from a production cluster with 6000 GPUs [6]. This data-trace contains features including job duration, submission time, user id of the individual submitting the job, requested number of GPUs, and `group id` that identifies recurring jobs. After completing a data cleaning process, we obtain a total of 758,223 jobs for analysis.

However, this data-trace does not provide the training jobs' DNN model information. To address this issue, we profile nine representative DNN models on the vGPUs: three image classification models on the ImageNet dataset [39] and six natural language processing (NLP) models. The details of this model profiling are summarized in Table I. Here, BERT-large and XLNet-large are profiled on the SQuAD2.0 dataset [40]. For T5 and the three versions of GPT models that cannot be accommodated on a single GPU, we construct a smaller model consisting of three layers from the original model, which will be used for profiling with a token sequence length of 512. The distributed training configurations for each model are derived from the planner proposed in [20], which calculates multiple configurations per model. We assign each model and the derived distributed training configuration to a job group (*i.e.*, a group of recurrent jobs) following the GPU requirement in the trace. If a job in a group requires only a single GPU, we pair the group with a model with a single-GPU training configuration. Otherwise, if the job group demands more than one GPU, we randomly select a model and one of its training configurations for the group. The number of job training iterations is computed by dividing the job duration in the trace by its approximate minimum per-iteration training time, $\tilde{\alpha}_i^{\min}$.

Due to the limited size of our local testbed, we randomly selected three sets of 75 consecutive jobs from the original

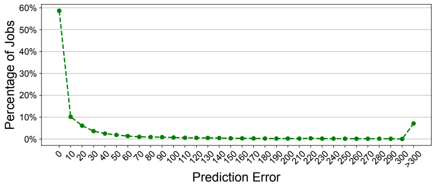

Fig. 4: Percentage of jobs: different prediction errors.

Model	# of Parameters	Mini-Batch Size
VGG19 [41]	144M	32
ResNet152 [2]	60M	4
Inception-V3 [42]	24M	32
BERT-large [43]	340M	4
XLNet-large [44]	550M	4
T5 [45]	11B	8
GPT [1]	6.7B/13B/175B	32/32/16

TABLE I: DNN models.

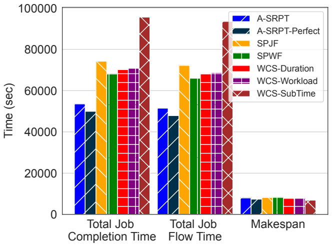

Fig. 5: Testbed experiment performance.

traces. We uniformly scaled down the job arrival times and training iterations to 10% of the original data, resulting in a scheduling period on the order of hours per method.

1-c) Prediction Model: We use the first 80% jobs in the trace to train our random forest regression prediction model, completing in just 84 seconds. The prediction error is depicted in Fig. 4, which shows that approximately 60% of the jobs are predicted correctly. Although there remains a non-negligible prediction error in a small fraction of jobs, our subsequent evaluation reveals that our algorithm outperforms the baseline performance even with imperfect predictions.

1-d) Baselines: Our A-SRPT algorithm is compared with five baseline GPU cluster scheduling algorithms: (1) *SPJF (Shortest Predicted Job First):* This approach schedules jobs based on their predicted durations as proposed by MLaaS [6]; (2) *SPWF (Shortest Predicted Workload First):* This policy proposed in Tiresias [14] schedules jobs according to the product of predicted durations and the number of required GPUs; (3) *WCS-Duration (Work-Conserving Scheduler, WCS [46] by Duration):* This approach continuously schedules jobs to use available GPUs within the cluster following the order based on their predicted duration; (4) *WCS-Workload*: Variant of (3), sequencing by predicted workload; (5) *WCS-SubTime*: Variant of (3), arranged by submission time. All baselines adopt the Heavy-Edge algorithm for GPU mapping in both testbed and simulation experiments.

2) Experimental Results: We present the real testbed results in Fig. 5, averaged over three job sets. The total job flow time is defined as the difference between each job's completion time and arrival time, and the makespan is the completion time of the final job. We include the baseline *A-SRPT-Perfect*, which uses A-SRPT with perfect knowledge of job durations (*i.e.*, perfect prediction). Our A-SRPT achieves performance close to *A-SRPT-Perfect*, with only 7% longer total job completion time, and significantly outperforms all other baselines. While *WCS* baselines achieve shorter system makespans, they prioritize scheduling longer training jobs whenever possible. This blocks the timely execution of later arriving shorter jobs, resulting in larger total job completion times. In contrast, our algorithm reduces the total job completion time by up to 44%.

B. Large-Scale Simulations

1) System Settings: *1-a) System Settings:* We consider a cluster consisting of 250 servers, each equipped with eight GPUs. The NIC bandwidth of each server is set to 10Gbps, and the inter-GPU communication bandwidth within each server is 300GB/s, based on the NVLink specs of NVIDIA V100 GPUs. We profiled all DNN models on a single NVIDIA V100 GPU. For scheduling, we randomly sample consecutive jobs from the original trace.

2) Experimental Results: *2-a) Different Number of Jobs:* As the number of jobs increases, the workload and job diversity grow, challenging the online algorithm's ability to handle varying job sizes. Fig. 6 shows total job completion times for A-SRPT and baselines with job counts from 37,500 to 150,000 (5% to 20% of the trace). *SPJF* performs the worst due to its rigid strategy based solely on predicted durations, neglecting varying GPU demands. If the shortest job does not fit, it will not schedule longer jobs with fewer GPU demands. *SPWF* balances job duration with GPU needs, leading to better workload distribution. *WCS-Duration* and *WCS-Workload* enhance GPU utilization but delay larger jobs by prioritizing smaller ones. A-SRPT consistently outperforms baselines, reducing total job completion times by 31% to 91%.

2-b) Different Percentages of Single-GPU Jobs: The original trace [6] has over 70% single-GPU jobs, making scheduling less challenging due to minimal server assignment. Thus, we fix the number of jobs at 75,000 and vary the percentage of single-GPU jobs, with jobs randomly set for single-GPU or distributed training. As the fraction of distributed jobs increases, the scheduling problem becomes harder due to higher workloads and complex communication. Fig. 7 shows that as single-GPU jobs decrease from 80% to 0%, A-SRPT increasingly outperforms baselines, reducing total job completion time by 16% to 57%.

2-c) Different Server NIC Bandwidths: We evaluate A-SRPT with server NIC bandwidths from 1 Gbps to 50 Gbps, using the job set with 0% single-GPU jobs. Lower bandwidth exacerbates communication overhead, yielding longer total job completion times under poor scheduling. Fig. 8 shows A-SRPT maintains consistent performance gains, while baselines falter at 1 Gbps. Notably, at 50 Gbps, A-SRPT outperforms the best baseline *WCS-Duration* by 12%, and at 1 Gbps, it reduces total job completion time by up to 92%, demon-

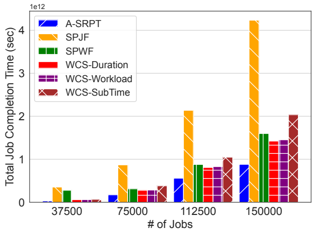

Fig. 6: Total job completion time comparisons with different numbers of jobs.

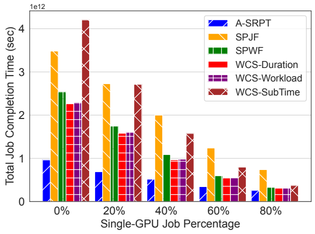

Fig. 7: Completion time comparisons with different percentages of single-GPU jobs.

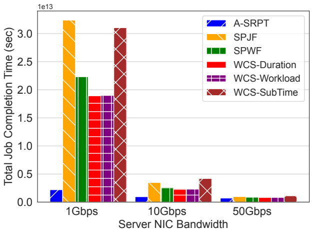

Fig. 8: Total job completion time with different server NIC bandwidths.

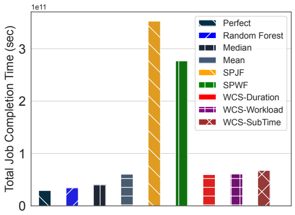

Fig. 9: Total completion time comparisons: different prediction models and baselines.

Model	Heavy-Edge		ILP	
	PITT (ms)	**PCT (ms)**	**PITT (ms)**	**PCT (ms)**
VGG19	88.11	1.94	82.96	55318.86
GPT-175B	10.14	1.52	10.14	2288.12

TABLE II: Per-iteration training time (PITT) and placement computation time (PCT): Heavy-Edge vs. ILP

strating its effectiveness in handling communication overhead and ensuring efficient job training.

2-d) Different Prediction Models: We now examine the performance of our prediction model in Fig. 9 using jobs with GPU demands following the original trace. Our random forest regression model is compared with simpler methods based on the mean and median of previous job iterations, as well as a perfect prediction model (*i.e.*, *A-SRPT-Perfect*). All other baselines use random forest regression. The average errors for the random forest, median-based, and mean-based models are 369, 563, and 593, respectively. The random forest model outperforms simpler methods due to lower average error and is only 14% less efficient than the perfect model, while less accurate models (e.g., mean-based) significantly degrade algorithm performance.

2-e) Heavy-Edge vs. Integer Linear Programming (ILP): Finally, we evaluate the performance of Heavy-Edge, with results shown in Table II. In comparison, the placement is formulated as an ILP problem based on [47] and solved optimally using the Gurobi Optimizer [48]. Experiments were conducted on a MacBook Pro (M1 MAX chip, 64 GB memory). We compare the per-iteration training time (PITT) and placement computation time (PCT) for two of our profiled models, averaging results over 20 cases with varying GPU availability per server. For the VGG19 model, the heterogeneity in computation time and data communication presents challenges in GPU mapping. Heavy-Edge achieves a PITT only 6% longer than the optimal ILP solution, while computing in under TWO milliseconds compared to ILP's 55+ seconds. Moreover, for the GPT-175B model, the uniform structure allows Heavy-Edge to find a solution 1500 times faster than the ILP.

VI. Discussions

We note that the landscape of parallelism for distributed deep learning training continues to evolve. New methods, such as tensor parallelism [8] and expert parallelism [49], have been key enablers for training extremely large-scale foundation models [50]. Reflecting on this, it is interesting to discuss how A-SRPT can be extended to work with emerging parallelisms to enable DDL scheduling designs for the future.

▷ **Tensor parallelism.** Tensor parallelism splits layers across multiple GPUs, necessitating extensive inter-GPU communication through AllReduce operations [8]. To adapt Heavy-Edge for tensor parallelism, we modify our graph model $\Omega = (\mathcal{V}, \mathcal{E})$ to represent tensor slices as vertices and AllReduce operations as weighted edges. For communication efficiency, all tensor slices of a layer must reside within a single server. To accommodate this, A-SRPT delays the start of tensor parallelism jobs until sufficient server capacity is available.

▷ **Expert parallelism.** Expert parallelism in Mixture-of-Experts (MoE) models distributes different 'expert' layers across GPUs, posing challenges in balancing workloads and managing inter-GPU communication [49]. We can represent expert groups as vertices in our graph-based model. Communication, characterized by sparse activations/gradients and token routing, is represented as weighted edges. Due to the dynamic data transfer patterns presented in MoE training, we can set the edge weights based on the estimated average communication costs. This allows MoE jobs to be integrated into our unified graph model, enabling effective placement and scheduling with Heavy-Edge and A-SRPT.

VII. Conclusion

In this paper, we investigated online scheduling for distributed deep learning with mixed parallelism (DDLwMP) jobs in GPU clusters. We introduced the adaptive shortest-remaining-processing-time first (A-SRPT) scheduling method, which integrates: 1) a GPU mapping algorithm that strategically assigns GPUs to job stages to minimize communication overhead by co-locating communication-intensive parts, and 2) an online scheduling algorithm that uses a prediction model for job scheduling. By modeling each DDL job as a graph, our GPU mapping algorithm reduces communication overhead effectively. Additionally, we proposed an online scheduling algorithm that transforms the complex GPU cluster scheduling problem into a single-machine instance, which can be optimally solved. The scheduling decisions from this simplified problem then guide the actual GPU cluster scheduling. Theoretical analysis and trace-driven experiments demonstrated A-SRPT's efficacy, achieving up to 92% reduction in total job completion time compared to baselines.

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language Models Are Few-Shot Learners," *arXiv preprint arXiv:2005.14165*, 2020.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of IEEE CVPR*, 2016.

[3] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling Deep Reinforcement Learning for Datacenter-Scale Automatic Traffic Optimization," in *Proc. of ACM SIGCOMM*, 2018.

[4] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: Introspective Cluster Scheduling for Deep Learning," in *Proc. of USENIX OSDI*, 2018.

[5] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning," in *Proc. of USENIX OSDI*, 2021.

[6] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters," in *Proc. of USENIX NSDI*, 2022.

[7] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling Distributed Machine Learning with the Parameter Server," in *Proc. of USENIX OSDI*, 2014.

[8] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using GPU Model Parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[9] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "PipeDream: Generalized Pipeline Parallelism for DNN Training," in *Proc. of ACM SOSP*, 2019.

[10] Z. Han, H. Tan, S. H.-C. Jiang, X. Fu, W. Cao, and F. C. Lau, "Scheduling Placement-Sensitive BSP Jobs with Inaccurate Execution Time Estimation," in *Proc. of IEEE INFOCOM*. IEEE, 2020.

[11] Q. Wang, S. Shi, C. Wang, and X. Chu, "Communication Contention Aware Scheduling of Multiple Deep Learning Training Jobs," *arXiv preprint arXiv:2002.10105*, 2020.

[12] M. Yu, B. Ji, H. Rajan, and J. Liu, "On Scheduling Ring-All-Reduce Learning Jobs in Multi-Tenant GPU Clusters with Communication Contention," in *Proc. of ACM MobiHoc*, 2022.

[13] S. K. Karmaker, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni, "AutoML to Date and Beyond: Challenges and Opportunities," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–36, 2021.

[14] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A GPU Cluster Manager for Distributed Deep Learning," in *Proc. of USENIX NSDI*, 2019.

[15] *NVIDIA NVLink*, https://www.nvidia.com/en-us/data-center/nvlink/.

[16] L. Breiman, "Random Forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[17] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, "Sequencing and Scheduling: Algorithms and Complexity," *Handbook in Operations Research and Management Science: Logistics of Production and Inventory*, vol. 4, 1993.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[19] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism," in *Proc. of NeurIPS*, 2019.

[20] Z. Luo, X. Yi, G. Long, S. Fan, C. Wu, J. Yang, and W. Lin, "Efficient Pipeline Planning for Expedited Distributed DNN Training," in *Proc. of IEEE INFOCOM*. IEEE, 2022.

[21] A. Sergeev and M. Del Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[22] P. Sanders, J. Speck, and J. L. Traff, "Two-Tree Algorithms for Full Bandwidth Broadcast, Reduction And Scan," *Parallel Computing*, vol. 35, no. 12, pp. 581–594, 2009.

[23] M. Yu, Y. Tian, B. Ji, C. Wu, H. Rajan, and J. Liu, "GADGET: Online Resource Optimization for Scheduling Ring-All-Reduce Learning Jobs," in *Proc. of IEEE INFOCOM*. IEEE, 2022.

[24] *Massively Scale Your Deep Learning Training with NCCL 2.4*, https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/.

[25] S. Fan, Y. Rong, C. Meng, Z. Cao, S. Wang, Z. Zheng, C. Wu, G. Long, J. Yang, L. Xia *et al.*, "DAPPLE: A Pipelined Data Parallel Approach for Training Large Models," in *Proc. of ACM PPoPP*, 2021, pp. 431–445.

[26] D. Narayanan, A. Phanishayee, K. Shi, X. Chen, and M. Zaharia, "Memory-Efficient Pipeline-Parallel DNN Training," in *Proc. of ICML*, 2021.

[27] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters," in *Proc. of EuroSys*, 2018.

[28] A. B. Faisal, N. Martin, H. M. Bashir, S. Lamelas, and F. R. Dogar, "When Will My ML Job Finish? Toward Providing Completion Time Estimates through Predictability-Centric Scheduling," in *Proc. of USENIX OSDI*, 2024.

[29] E. Bampis, K. Dogeas, A. V. Kononov, G. Lucarelli, and F. Pascual, "Scheduling with Untrusted Predictions," in *Proc. of IJCAI*, 2022.

[30] J. M. Tarnawski, D. Narayanan, and A. Phanishayee, "Piper: Multidimensional Planner for DNN Parallelization," in *Proc. of NeuIPS*, 2021.

[31] *Performance reported by NCCL tests*, https://github.com/NVIDIA/nccl-tests/blob/master/doc/PERFORMANCE.md.

[32] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A Generic Communication Scheduler for Distributed DNN Training Acceleration," in *Proc. of ACM SOSP*, 2019, pp. 16–29.

[33] M. Yu, C. Wu, B. Ji, and J. Liu, "A Sum-Of-Ratios Multi-Dimensional-Knapsack Decomposition for DNN Resource Scheduling," in *Proc. of IEEE INFOCOM*, 2021.

[34] J. Turek, J. L. Wolf, and P. S. Yu, "Approximate Algorithms for Scheduling Parallelizable Tasks," in *Proc. of ACM SPAA*, 1992.

[35] K. Andreev and H. Räcke, "Balanced Graph Partitioning," in *Proc. of ACM Symposium on Parallelism in Algorithms and Architectures*, 2004.

[36] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, "Approximation Techniques for Average Completion Time Scheduling," *SIAM Journal on Computing*, vol. 31, no. 1, pp. 146–166, 2001.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. of NeurIPS*, 2019.

[38] *NVIDIA Multi-Instance GPU*, https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.

[39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. of IEEE CVPR*, 2009.

[40] P. Rajpurkar, R. Jia, and P. Liang, "Know What You Don't Know: Unanswerable Questions for SQuAD," *arXiv preprint arXiv:1806.03822*, 2018.

[41] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Proc. of IEEE CVPR*, 2016.

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[44] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," in *Proc. of NeurIPS*, 2019.

[45] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[46] Y. Zheng, N. B. Shroff, R. Srikant, and P. Sinha, "Exploiting Large System Dynamics for Designing Simple Data Center Schedulers," in *Proc. of IEEE INFOCOM*, 2015.

[47] A. Archer, M. Fahrbach, K. Liu, and P. Prabhu, "Pipeline Parallelism for DNN Inference with Practical Performance Guarantees," *arXiv preprint arXiv:2311.03703*, 2023.

[48] *Gurobi Optimizer*, https://www.gurobi.com/.

[49] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram *et al.*, "Tutel: Adaptive Mixture-of-Experts at Scale," *Proc. of MLSys*, vol. 5, 2023.

[50] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the Opportunities and Risks of Foundation Models," *arXiv preprint arXiv:2108.07258*, 2021.