GADGET: Online Resource Optimization for Scheduling Ring-All-Reduce Learning Jobs

Menglu Yu[1]   Ye Tian[1]   Bo Ji[2]   Chuan Wu[3]   Hridesh Rajan[1]   Jia Liu[4,1]

[1]Department of Computer Science, Iowa State University
[2]Department of Computer Science, Virginia Tech
[3]Department of Computer Science, The University of Hong Kong
[4]Department of Electrical and Computer Engineering, The Ohio State University

Abstract—**Fueled by advances in distributed deep learning (DDL), recent years have witnessed a rapidly growing demand for resource-intensive distributed/parallel computing to process DDL computing jobs. To resolve network communication bottleneck and load balancing issues in distributed computing, the so-called "ring-all-reduce" decentralized architecture has been increasingly adopted to remove the need for dedicated parameter servers. To date, however, there remains a lack of theoretical understanding on how to design resource optimization algorithms for efficiently scheduling ring-all-reduce DDL jobs in computing clusters. This motivates us to fill this gap by proposing a series of new resource scheduling designs for ring-all-reduce DDL jobs. Our contributions in this paper are three-fold: i) We propose a new resource scheduling analytical model for ring-all-reduce deep learning, which covers a wide range of objectives in DDL performance optimization (e.g., excessive training avoidance, energy efficiency, fairness); ii) Based on the proposed performance analytical model, we develop an efficient resource scheduling algorithm called GADGET (greedy ring-all-reduce distributed graph embedding technique), which enjoys a provable strong performance guarantee; iii) We conduct extensive trace-driven experiments to demonstrate the effectiveness of the GADGET approach and its superiority over the state of the art.**

I. INTRODUCTION

In recent years, the rise of complex deep learning applications has led to a rapidly growing demand for resource-intensive (e.g., GPUs, memory, energy) distributed/parallel computing to process deep learning training tasks. Traditionally, most distributed deep learning (DDL) frameworks are based on the parameter server (PS)-worker architecture, which consists of a set of PS(s) and workers. Despite its simplicity, the PS-worker architecture suffers from two scalability limitations: i) The topology of the PS-worker architecture creates a *communication bottleneck* at each PS as the number of workers increases; and ii) The centralized PSs are vulnerable to the *single-point-of-failure risk*. To overcome these scalability weaknesses, the more sophisticated *"ring-all-reduce"* (RAR) parallel computing architecture has become increasingly popular for DDL training and has been supported by many mainstream DDL frameworks (e.g., Tensorflow [1],

Pytorch [2]). Specifically, by forming a ring between the workers to jointly perform parameter sharing and reduction, the RAR architecture removes the need for dedicated PS(s), hence alleviating the single point of failure.

However, with the increasing adoption of the RAR architecture for DDL training, an important question naturally emerges: *How could we design resource optimization algorithms to efficiently schedule RAR-based DDL training jobs over networked computing clusters?* Answering this question is critical because: i) Multi-core high-throughput GPU hardware for cloud-based machine learning services is expensive, which requires efficient GPU utilization. For example, an Amazon EC2 eight-core GPU instance with NVLink connection costs more than $31 per hour [3]; ii) Due to the resource competition among multiple RAR-based DDL training jobs in the cluster, if scheduling is not done strategically, DDL training jobs could suffer from large latency; and iii) RAR-based DDL jobs are often trained by variants of the *iterative* stochastic gradient descent (SGD) method to optimize thousands or even millions of parameters. Their completion times are defined by the convergence processes of these SGD-based methods, which often exhibit the *"diminishing return effect"* in terms of the gains in training accuracy as the number of iterations increases. For instance, it has been shown in [4] that approximately 75% of DDL jobs reach within 0.1% of the lowest training loss using only 40% of the epochs. Allocating too many computing resources to one job not only induces high GPU and energy costs as well as unfairness to other DDL jobs, but also leads to unnecessary training time with little learning accuracy performance gain in return.

However, optimizing scheduling and resource allocation for RAR-based DDL is highly non-trivial due to several technical challenges. *First*, the hop-by-hop dependence in the ring structure renders the placement of workers highly sensitive to intra- and inter-server communications, decided by the underlying computing network topology. Also, as will be shown later, the ring topology violates the loop-less assumption of many existing algorithms for virtual computing resource allocation problems [5]–[7], which necessitates new algorithmic design techniques. *Second*, the resource allocation for each RAR-based DDL job is subject to packing-type constraints (due to resource limits), which implies NP-Hardness. *Lastly*, the

scheduler is not aware of DDL job arrivals beforehand, which calls for *online* optimization algorithm design. Perhaps due to these challenges, to date, results on scheduling and resource allocation for RAR-based DDL training remain limited. This motivates us to fill this gap by proposing an efficient resource scheduling algorithm called GADGET (greedy ring-all-reduce distributed graph embedding technique), which addresses the aforementioned technical challenges with strong theoretical performance guarantees. The main results in this paper and their significance are summarized as follows:

- By extracting the key architectural features of RAR-based DDL training jobs, we develop a new analytical model for scheduling RAR-based DDL jobs over networked environments. Based on this model, we formulate a general online performance optimization framework for RAR-based DDL training. We note that, due to the heterogeneous internal (i.e., between containers in the same physical server) and external (i.e., between physical machines within or beyond the same rack) communications, the resource scheduling and allocation problem for RAR-based DDL training is far more challenging than those for the PS-worker architecture.

- To address the new challenges arising from RAR-based DDL training, we develop an efficient GADGET algorithm that can provide provably strong performance guarantee. Our GADGET approach is based on a "divide-and-conquer" approach. Specifically, we first show that the formulated RAR-based training scheduling problem over the temporal domain possesses a submodular structure due to its partition matroid nature, and hence can be decomposed in the temporal domain and solved by a greedy approach with worst-case approximation ratio guarantee.

- Next, we focus on the decomposed subproblem in each time-slot, which remains NP-Hard due to the packing-type constraints. We note that the ring-structure of RAR-based DDL jobs renders existing methods ineffective. To address this challenge, we propose a *generalized* virtual graph embedding (G-VNE) technique that guarantees a $\frac{1}{3\Gamma}$-fraction of the maximum utility value, where $\Gamma \geq 1$ is a problem-dependent constant. Combining this with the submodular property implies that GADGET achieves an $\frac{1}{3\Gamma+1}$ overall competitive ratio.

- Lastly, we conduct experiments to examine the performance of our GADGET algorithm. We demonstrate the good approximation ratio of G-VNE, which is the main component of GADGET algorithm by using real-world trace-driven simulations. We also show that, compared to existing baseline scheduling schemes, our GADGET algorithm can effectively meet the topology constraints, while achieving a good overall performance.

Collectively, our results contribute to a comprehensive and fundamental understanding of RAR-based machine learning system optimization. The rest of this paper is organized as follows. We review related work in Section II, and present preliminaries to familiarize readers with the necessary background of RAR in Section III. We then present the system model,

problem formulation, and an overview of our algorithmic ideas in Section IV. We propose our online resource scheduling algorithm by decoupling it into time-independent subproblems in Section V, and then solve the NP-hard subproblem with our G-VNE approach in Section V-C. We evaluate performance of our proposed algorithms through numerical experiments in Section VI, and conclude this paper in Section VII.

II. RELATED WORK

Due to the rise of deep learning and their intensive computation workload, scheduling optimization for DDL to expedite the training process has attracted increasing attention recently. To date, the PS-worker architecture [8] has been widely adopted and its scheduling design has been relatively well studied (e.g., [9]–[12]). However, as pointed out in Section I, the PS-worker architecture suffers from communication bottlenecks and reliability limitations. Thanks to its better scalability compared to the PS-worker architecture [13], the RAR architecture has received strong interest in the research community and has been recently adopted by modern DDL frameworks. So far, however, results on scheduling designs for the RAR architecture remain scarce. To our knowledge, PACE [14] is the only existing work in the literature designed for all-reduce tensors based on the RAR architecture, aiming at maximizing the overlap between communication and computation using DAG (directed acyclic graph) of DNN training. However, the goal of PACE is to speed up the training process of a *single job*, instead of optimizing the scheduling of multiple jobs to improve the system-wide performance (e.g., minimize the average completion time). In contrast, our goal in this work is to design the *first* scheduler tailored for the RAR-based DDL jobs in computing clusters. We propose a theoretical framework that enables rigorous RAR-based DDL training resource optimization in large-scale computing clusters (typically with a multi-layer hierarchical topology, e.g., fat-tree).

We note that there also exists other lines of research on resource scheduling for optimizing DDL training performance (e.g., latency, energy efficiency), but they are agnostic to the underlying distributed/parallel architectures. Here, we also provide a quick overview, although they are not directly comparable to our work. In [15], a GPU scheduler called Tiresias is proposed based on the assumption that DDL job performance could be estimated from historical job duration information. Meanwhile, a resource provisioning method called Cynthia was developed in [16] to study the impact of system heterogeneity, i.e., servers in the system may have different hardware configuration (e.g., CPU, amounts of memory), on both synchronous and asynchronous training, and to optimize the training performance. Also, due to the growing training workloads that incur huge energy consumption in the GPU clusters, the design of energy-efficient scheduling algorithms also receives significant interest recently. For example, in [17], [18], various scheduling schemes are proposed for CPU/GPU hybrid clusters, aiming at maximizing the energy efficiency without significantly sacrificing the system performance. We
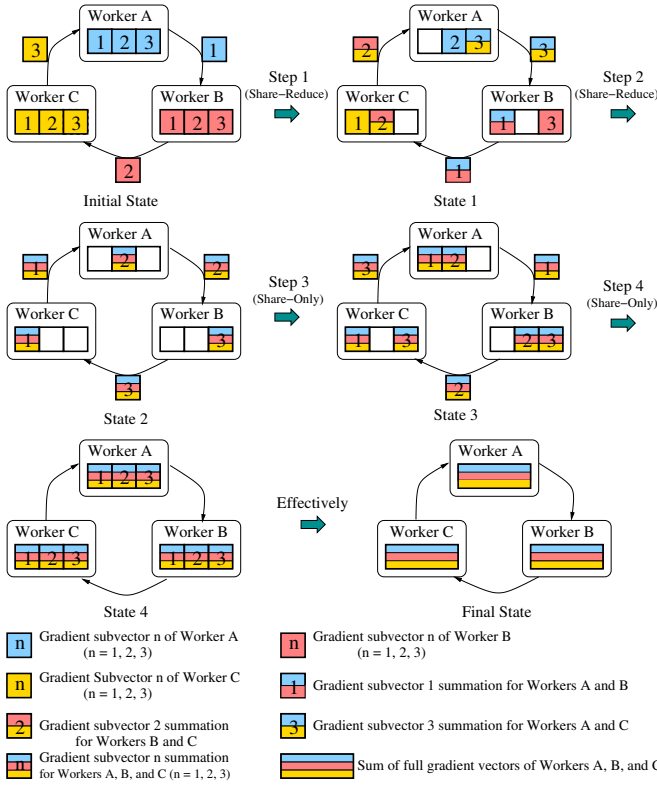
Fig. 1: A three-worker illustrative example of the ring-all-reduce (RAR) process.

note, however, that all of these DDL scheduling algorithms are heuristic methods that do not provide performance guarantees.

III. DISTRIBUTED LEARNING WITH RING-ALL-REDUCE: A PRIMER

In this section, we provide an overview on DDL training based on the RAR parallel computing architecture, to familiarize readers with necessary background and fix terminologies that will be used in this paper.

1) Distributed Stochastic Gradient Descent (SGD): At the heart of deep learning lies an optimization problem in the form of $\min_{\mathbf{w} \in \mathbb{R}^d} \bar{L}(\mathbf{w}) \triangleq \frac{1}{N} \sum_{i=1}^N L(\mathbf{w}, \xi_i)$, where $\mathbf{w}$ contains the model parameters to be learned, $L(\mathbf{w}, \xi_i)$ is a loss function, and N is the total number of samples. In an w-worker DDL system, the dataset is often partitioned and trained by each worker. Let $\mathcal{D}_j$ denote the j-th partition of the dataset trained by a worker $j \in \{1, \dots, w\}$. Then, the training problem can be decomposed as $\min_{\mathbf{w} \in \mathbb{R}^d} \bar{L}(\mathbf{w}) = \sum_{j=1}^w \frac{|\mathcal{D}_j|}{N} L_j(\mathbf{w})$, where $L_j(\mathbf{w}) \triangleq \frac{1}{|\mathcal{D}_j|} \sum_{i \in \mathcal{D}_j} L(\mathbf{w}, \xi_i)$. To date, most DDL systems in practice adopt the distributed stochastic gradient descent (SGD) method, where, in iteration k, the weight parameter vector $\mathbf{w}$ is updated as $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \left(\sum_{j=1}^n \frac{|\mathcal{D}_j|}{N} \mathbf{g}_j^k \right)$, $k = 1, 2, \dots$. Here η_k is the step-size (aka learning rate) in iteration k, and $\mathbf{g}_j^k$ denotes the stochastic gradient computed by worker j in iteration k.

2) The Ring-All-Reduce (RAR) Architecture: The above SGD update requires a weighted sum of all stochastic gradients, $\mathbf{g}_j$, $\forall j \in \{1, \dots, w\}$ (the iteration index k is omitted for notational simplicity). To compute this weighted sum in the PS-worker architecture, each worker j simply sends $\mathbf{g}_j$ to the PSs, which then perform the summation and return the result to each worker. However, this implies that a $2wd$ amount of data exchange per iteration is required under the PS-worker architecture, which scales *linearly* as the number of workers increases and is problematic in large-scale DDL training.

To address this limitation, the more sophisticated RAR architecture is proposed. The basic idea of RAR is to form a *ring* between the workers, where each worker performs gradient reduction (e.g., summation) and sharing by receiving gradients from its upstream worker and sending the local reduction result to its downstream worker. In general, for a w-worker RAR structure, each worker splits its gradients into w sub-vectors (see Fig. 1 for an example for $w = 3$). The RAR process is divided into two phases. The first is the *"Share-Reduce"* phase ($t = 1, \dots, w - 1$), where each worker sends its reduced sub-vector (i.e., the sub-vector sum) to its downstream worker, while receiving the reduced sub-vector from its upstream worker to compute a new reduced sub-vector. The second phase is the *"Share-Only"* phase ($t = w, \dots, 2w - 2$), where each node sends its newly received fully-reduced sub-vector to its downstream worker, while receiving a new fully-reduced sub-vector from its upstream worker. Since each worker sends $\frac{d}{w}$ amount of data for $2(w-1)$ times, the total amount of data each worker sends is $\frac{2d(w-1)}{w}$, which is asymptotically *independent* of w as w gets large.

3) Per-Iteration Training Time for RAR-Based DDL: Consider a w-worker RAR-based DDL training job. We use b to denote the bandwidth between workers. We use G to denote the computation speed of a worker. Since each worker sends a gradient sub-vector of size d/w at each time step for $2(w - 1)$ times, the transmission time can be computed as $\frac{2d(w-1)}{wb}$. Also, since it takes $w - 1$ times in total to perform gradient sub-vector summations, the computation time can be computed as $\frac{d(w-1)}{wG}$. Hence, the total time for a single RAR operation is $\frac{d(w-1)}{w}\left[\frac{2}{b} + \frac{1}{G}\right]$. In each iteration of RAR-based DDL training, in addition to the all-reduce operation time, each worker needs to perform a forward pass (FP) and a backward pass (BP) to compute the stochastic gradients. The FP time can be computed as $t^f M$, where t^f is the model-dependent per-sample FP time and M is mini-batch size. The BP time t^b is independent of the mini-batch size. In addition, there is extra latency γ caused by communication overhead (e.g., ACK time for message transmission, negotiation time among all workers before conducting RAR [13]). By putting all the above time consumption components together, we have the per-iteration training time τ for RAR-based DDL as:

$$\tau = \frac{d(w-1)}{w}\left[\frac{2}{b} + \frac{1}{G}\right] + t^f M + t^b + \gamma. \tag{1}$$

We can see that τ depends on the learning model size, worker

---

[1]In this paper, we focus on GPU computation and network communication times and neglect the data IO latency due to the use of high-speed solid state drives (SSDs), short pre-fetch time of pre-stored training programs, and overlapping of data chunks fetching and training through pipelining.

communication and computation speeds, batch size, communication overhead, and FP/BP times (in turn, the learning model), but is asymptotically upper bounded by $d(\frac{2}{b}+\frac{1}{G})+t^f M+t^b+\gamma$ as the number of workers w goes to infinity.

IV. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we will present a general analytical framework for resource scheduling and allocation for RAR-based DDL training performance optimization.

Consider a computing cluster with a set of physical servers denoted as $\mathcal{S}$. Each server has a certain amount of computing resources (e.g., GPUs, memory). The system is time-slotted with horizon $\mathcal{T} \triangleq \{1, 2, \cdots, T\}$. Over time, RAR-based DDL jobs dynamically arrive and compete for the shared resources with unfinished jobs. Let a_i denote the arrival time of job i, which is *unpredictable* to the scheduler. Let $\mathcal{I}$ denote the set of jobs arrived over time horizon $\mathcal{T}$. The jobs are *preemptive* if resources are insufficient and could be resumed later.[2] All jobs' workers are implemented as containers. Next, we will develop RAR-based DDL scheduling models in detail.

1) Resource Constraint Modeling: We let $y_{is}[t]$ denote the number of workers scheduled on server s for job i in time-slot t. Let N_i be the largest number of assigned concurrent workers in each time-slot, we have:

$$\sum_{s \in \mathcal{S}} y_{is}[t] \leq N_i, \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T}. \quad (2)$$

Let $\mathcal{R}$ denote the set of computing resources (e.g., GPUs, memory, training time budget). Let l_i^r be the amount of type-r resource consumed by each worker of job i. We use F_i^r to denote the maximum type-r resource demand requested by job i. To ensure that job i does not exceed type-r resource's limit, we have:

$$\sum_{t \in \mathcal{T}} l_i^r \sum_{s \in \mathcal{S}} y_{is}[t] \leq F_i^r, \quad \forall i \in \mathcal{I}, \forall r \in \mathcal{R}. \quad (3)$$

Let C_s^r denote the type-r resource capacity of server s. To ensure that server's type-r limit is not violated, we have:

$$\sum_{i \in \mathcal{I}} l_i^r y_{is}[t] \leq C_s^r, \quad \forall s \in \mathcal{S}, t \in \mathcal{T}, \forall r \in \mathcal{R}. \quad (4)$$

We use a binary variable $x_{is}[t] = 1$ to indicate that job i uses server s in each time-slot t and let $x_{is}[t] = 0$ if otherwise. Note that, when server s is used for job i, $y_{is}[t]$ should not exceed any maximum resource demand F_i^r, the resource capacity of server s, and the largest number of workers of job i in each time-slot t. Also, observe that $x_{is}[t] = 0$ forces $y_{is}[t] = 0$. Combining these facts yields:

$$y_{is}[t] \leq x_{is}[t] \min \left\{ N_i, \frac{C_s^r}{l_i^r}, \frac{F_i^r}{l_i^r}, \forall r \right\}, \forall i \in \mathcal{I}, s \in \mathcal{S}, t \in \mathcal{T}. (5)$$

To ensure no workers are allocated before jobs arrive, we have:

$$y_{is}[t] = 0, \quad \forall i \in \mathcal{I}, s \in \mathcal{S}, t < a_i. \quad (6)$$

---

[2]Flexible resource allocation enabled by containers can be exploited to suspend a job and reclaim its resources without losing the execution progress [19].
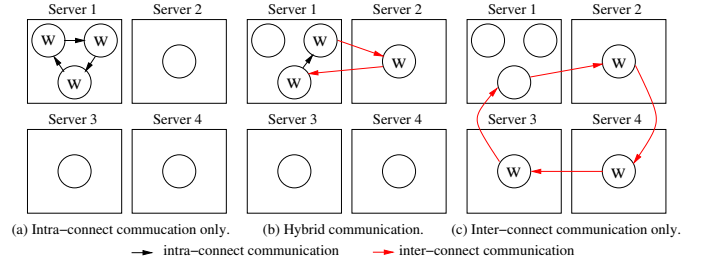

(a) Intra–connect communication only. (b) Hybrid communication. (c) Inter–connect communication only.

→ intra–connect communication   → inter–connect communication

Fig. 2: Various placement schemes with three workers.

2) RAR Topological Constraint Modeling: A key component in scheduling an RAR-based DDL training job is to guarantee that the *physical network topology* corresponding to the resource scheduling decisions should be *compatible* with the *logical topology* of the job's computational graph. Toward this end, we note that the computational graph of an RAR-based DDL training job i is a *directed ring graph* $G_i = (V_i, E_i)$, where V_i is the set of logical vertices representing workers, and E_i represents the set of logical edges denoting the RAR directions between the workers. Also, physical servers and network links in the cluster can be modeled as a directed substrate graph. Next, we model the topological constraints.

We use $\mathcal{P}_{ss'}[t]$ to denote the set of all possible paths between severs s and s' in the physical substrate graph in time-slot t. We use a binary variable $r_{ss'}^{(p,i)}[t] = 1$ to indicate that a path $p \in \mathcal{P}_{ss'}[t]$ is used by job i in time-slot t, and let $r_{ss'}^{(p,i)}[t] = 0$, otherwise. If neither server s nor s' hosts any worker of job i in time-slot t, then $r_{ss'}^{(p,i)}[t] = 0$, which implies:

$$r_{ss'}^{(p,i)}[t] \leq x_{is}[t] x_{is'}[t], \forall s, s' \in \mathcal{S}, p \in \mathcal{P}_{ss'}[t], i \in \mathcal{I}, t \in \mathcal{T}. \quad (7)$$

Also, we use b_i and $B_{\min}^p[t]$ to denote the reserved bandwidth requirement of job i and the bottleneck link capacity of path $p \in \mathcal{P}_{ss'}[t]$, respectively. To ensure that the bottleneck link capacity in any activated path p is not exceeded, we have:

$$\sum_{i \in \mathcal{I}} r_{ss'}^{(p,i)}[t] b_i \leq B_{\min}^p[t], \forall s, s' \in \mathcal{S}, p \in \mathcal{P}_{ss'}[t], t \in \mathcal{T}. \quad (8)$$

Next, to ensure that the allocated workers of job i can indeed form a directed ring graph, we consider two cases. First, if only one server hosts all workers for job i, then the cyclic constraint is automatically satisfied since a server can support any computational graph topology for co-located workers (Fig. 2(a)).[3] Second, if workers are located on different servers, it can be observed that each server that hosts some of job i's workers (i.e., $x_{is}[t] = 1$) should have a degree of *exactly two* for job i's paths in order to form a cycle (Fig. 2(b)-(c)). These two cases can be compactly written in one constraint as:

$$\mathbb{1}\left\{ \sum_{s' \in \mathcal{S}} x_{is'}[t] - 1 > 0 \right\} \left(\sum_{s' \in \mathcal{S}_i[t]} \sum_{p \in \mathcal{P}_{ss'}[t]} r_{ss'}^{(p,i)}[t] - 2 \right) = 0,$$
$$\forall i \in \mathcal{I}, s \in \mathcal{S}, t \in \mathcal{T}. \quad (9)$$

where $\mathcal{S}_i[t]$ denotes the set of servers containing job i at time t. To see why Eq. (9) is a valid constraint, note that if all workers

---

[3]Servers in Fig. 2 are connected cyclically, i.e., there are communication links between servers 1 and 2, 2 and 4, 4 and 3, 3 and 1.

of job i are hosted by server s, we have $\mathbb{1}\{\sum_{s'\in\mathcal{S}} x_{is'}[t]-1 > 0\} = 0$, which implies Eq. (9) trivially holds. Note also that the first summation in this term is over the set $\mathcal{S}_i[t]$. This guarantees that there is only one big cycle instead of multiple small cycles.

3) Objective Function and Problem Statement: Let $\mu_i(\cdot)$ be the utility function associated with job i's resource allocation, which is *non-decreasing* and concave to represent the "diminishing return effect." In this paper, our goal is to maximize the overall utility of all jobs. Let ζ_i represent some general notion of "per-worker efficiency/cost" (see three concrete examples next). Putting all modeling together, the DDL job scheduling problem (DDLJS) can be formulated as:

$$\textbf{DDLJS}: \underset{\mathbf{x},\mathbf{y}}{\text{Maximize}} \sum_{i\in\mathcal{I}} \mu_i\Big(\zeta_i\sum_{t\in\mathcal{T}}\sum_{s\in\mathcal{S}} y_{is}[t]\Big)$$

$$\text{subject to Constraints } (2)-(9).$$

We note that Problem DDLJS is a general analytical framework that has many applications. Here, we provide three examples to highlight its practical relevance: **1) Excessive Training Avoidance [4]:** Here, ζ_i can represent the number of training iterations per-unit time of each worker of job i, which can be obtained by inverting Eq. (1). The utility function can be chosen as $\mu(k) = (O(1/\sqrt{k}))^{-1} = C\sqrt{k}$ for some $C > 0$, which is the typical convergence rate of SGD-type algorithms with respect to the iteration index k. **2) Energy-Efficiency Optimization [18]:** Here, ζ_i denotes the per-worker power consumption of job i. We can choose $\mu(\cdot) = -c(\zeta_i\sum_{t\in\mathcal{T}}\sum_{s\in\mathcal{S}} y_{is}[t])$, where $c(\cdot)$ is a quadratic function, which is often used in the power system literature to model energy consumption costs. **3) Resource Fairness in Training:** Here, we can let $\zeta_i = 1, \forall i$, and adopt the classical "proportional fairness" utility function [20], i.e., $\mu(\cdot) = \log(\cdot)$.

V. SOLUTION APPROACH

Problem DDLJS is a challenging *online* optimization problem (the scheduler does not have arrival information $\{a_i, \forall i\}$). What exacerbates the problem is the fact that even its offline planning version (assuming $\{a_i, \forall i\}$ are known beforehand) is a mixed-integer non-convex programming (MINCP) problem, which is NP-Hard in general. Moreover, Eq. (9) contains an indicator function that is not amenable to conventional optimization techniques. In light of all these challenges, our goal in this paper is to pursue an online approximation algorithmic design that offers theoretical competitive ratio guarantee.

A. Basic Idea

To overcome the above challenges, we propose an online algorithmic design called GADGET (<u>g</u>reedy ring-<u>a</u>ll-reduce <u>d</u>istributed <u>g</u>raph <u>e</u>mbedding <u>t</u>echnique). Our basic idea of GADGET contains two key steps: i) Through a careful examination, we show that Problem DDLJS is submodular with respect to scheduling decisions in the *temporal* domain. Thus, it is possible to design a temporally greedy scheduling algorithm with a provable competitive ratio; ii) For the resource allocation subproblem in each time-slot that remains NP-Hard,
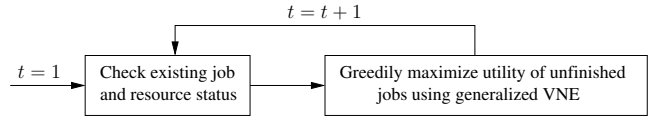

Fig. 3: Algorithmic idea overview.

we show that it can be viewed as a *generalized* virtual network embedding problem (G-VNE). As a result, it is also possible to (non-trivially) modify existing VNE techniques (see, e.g,. [5] and references therein) to adapt to our setting and solve each subproblem with provable approximation ratio guarantee. We illustrate the basic idea of our GADGET algorithm in Fig. 3. In what follows, we will discuss these two key steps in detail.

B. An Online Temporally Greedy Approach

In this subsection, we establish the submodularity of Problem DDLJS over the temporal domain. First, consider the following online *temporally greedy* algorithm. We let $z_{i,t} \triangleq \sum_{\tau=1}^{t}\sum_{s\in\mathcal{S}} y_{is}[\tau]$ be the *accumulative* number of worker-time allocated for job i up to time t. Define $\mathcal{I}[t] \triangleq \{i \in \mathcal{I} : t \geq a_i \text{ and } z_{i,t-1} < \min_{r\in\mathcal{R}} F_i^r/l_i^r\}$ as the set of jobs that are *active* (i.e., in-training and not resource-violated in time-slot t). Given the scheduling $\{z_{i,t-1}\}_{i\in\mathcal{I}[t]}$ in previous time slots, we greedily find the resource allocation in time-slot t by solving the following optimization problem:

$$\text{Maximize} \sum_{i\in\mathcal{I}[t]} \mu_i\Big(\zeta_i\sum_{s\in\mathcal{S}} y_{is}[t] + \zeta_i z_{i,t-1}\Big) \quad (10)$$

$$\text{subject to} \sum_{s\in\mathcal{S}} y_{is}[t] \leq \Big(\min_{r\in\mathcal{R}} \frac{F_i^r}{l_i^r}\Big) - z_{i,t-1}, \forall i\in\mathcal{I}[t], \quad (11)$$

$$\text{Constraints } (2),(4)-(9) \text{ only at } t, \forall i\in\mathcal{I}[t],$$

$$y_{is}[t] \in \mathbb{Z}_+, \ x_{is}[t] \in \{0,1\}, \forall i\in\mathcal{I}[t], s \in \mathcal{S}.$$

Constraint (11) ensures that the accumulated "worker-time product" in time slot t does not exceed the remaining worker-time-product limit (determined by the bottleneck resource type) for all active jobs. Note that Problem (10) remains a challenging NP-Hard packing problem, for which an approximation algorithm will be developed later in Section V-C. With Problem (10), our online temporally greedy algorithm is presented in Algorithm 1.

Algorithm 1: Online Temporally Greedy Approach.

1 Initialization: Set $z_{i,t} \leftarrow 0, \forall t \in \mathcal{T}, i \in \mathcal{I}$;

2 for $t \in \mathcal{T}$ **do**

3 $\mathcal{I}[t] \triangleq \{i \in \mathcal{I} : t \geq a_i \text{ and } z_i[t-1] < \min_{r\in\mathcal{R}} F_i^r/l_i^r\}$;

4 $y_{is}[t] \leftarrow$ Solutions of Problem (10) using Algorithm 2 developed in Section V-C;

5   **return** $y_{is}[t]$;

6 $z_{i,t} \leftarrow z_{i,t-1} + \sum_{s\in\mathcal{S}} y_{is}[t]$;

Next, we show that Algorithm 1 provides competitive ratio guarantee by proving that Problem DDLJS is submodular over the temporal domain. For the paper to be self-contained, we restate some necessary basics of submodular optimization and

matroid theory here, and refer readers to standard sources of submodular optimization (e.g., [21]) for further details.

Definition 1 (Submodularity). *A set function $f(\cdot) : 2^{\mathcal{V}} \to \mathbb{R}$ is submodular if for every $\mathcal{B} \subseteq \mathcal{V}$, and $\mathcal{A}' \subseteq \mathcal{A} \subseteq \mathcal{V}$, we have $f(\mathcal{A} \cup \mathcal{B}) - f(\mathcal{A}) \le f(\mathcal{A}' \cup \mathcal{B}) - f(\mathcal{A}').$*

An important subclass of submodular functions are those that have the monotone property defined as follows:

Definition 2 (Monotonicity). *A set function $f(\cdot) : 2^{\mathcal{V}} \to \mathbb{R}$ is monotone if for every $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$, we have $f(\mathcal{A}) \le f(\mathcal{B})$.*

In this paper, we focus on non-negative monotone submodular functions. Submodular optimization is also closely related to the notion of matroid, which is defined as follows:

Definition 3 (Matroid). *A matroid is a pair $(\mathcal{V}, \mathcal{J})$ such that $\mathcal{V}$ is a finite set, and $\mathcal{J} \subseteq 2^{\mathcal{V}}$ is a collection of independent subsets of $\mathcal{V}$ satisfying: 1) $\emptyset \in \mathcal{J}$; 2) for $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$, if $\mathcal{B} \in \mathcal{J}$, then it implies $\mathcal{A} \in \mathcal{J}$; 3) if $\mathcal{A}, \mathcal{B} \in \mathcal{J}$, and $|\mathcal{A}| < |\mathcal{B}|$, then $\exists\, v \in \mathcal{B} \setminus \mathcal{A}$ such that $\mathcal{A} \cup \{v\} \in \mathcal{J}$.*

Matroids have many interesting properties and subclasses. One important subclass of matroids that is useful in this paper is the partition matroid, which is defined as follows:

Definition 4 (Partition Matroid). *Partition $\mathcal{V}$ into disjoint subsets $\{\mathcal{V}_j\}$. Let $0 \le \nu_j \le |\mathcal{V}_j|$, $\forall j$ be integers associated with the $\mathcal{V}_j$'s. Define a collection of subsets $\mathcal{J} = \{\mathcal{H} \in \mathcal{V} : |\mathcal{H} \cap \mathcal{V}_j| \le \nu_j, \forall j\}$. Then, the pair $(\mathcal{V}, \mathcal{J})$ is a partition matroid.*

We now show that Problem DDLJS is a submodular optimization problem in the temporal domain. To avoid ambiguity, we will use the term *"schedule"* to refer to an $(\mathbf{x}, \mathbf{y})$-decision over the entire time horizon $\mathcal{T}$; and we use the term *"allocation"* to refer to an $(\mathbf{x}, \mathbf{y})$-decision in a particular time-slot (i.e., a "snapshot" in time). We let $\mathcal{H} \triangleq \{\mathcal{H}[t], t \in \mathcal{T}\}$ be the space of all schedules, where $\mathcal{H}[t]$ denotes the space of all resource allocation in time-slot t (which may or may not be feasible). Define the ground set $\mathcal{V} \triangleq \{y_{is}[t], \forall i, s, t \,|\, y_{is}[t] \in \mathcal{H}[t], \forall t\} \subset \mathbb{Z}_+^{|\mathcal{I}| \times |\mathcal{S}| \times |\mathcal{T}|}$ to be the space of all possible schedules of the y-components. Also, let $\mathcal{V}[t] = \{y_{is}[t], \forall i, s \,|\, y_{is}[t] \in \mathcal{H}[t]\} \subset \mathbb{Z}_+^{|\mathcal{I}| \times |\mathcal{S}|}$ denote the feasible y-allocation space in time-slot t. Clearly, $\{\mathcal{V}[t], t \in \mathcal{T}\}$ is a partition of $\mathcal{V}$ since $\mathcal{V}[t_1] \cap \mathcal{V}[t_2] = \emptyset$. Now, we choose $\nu_t = 1$, $\forall t \in \mathcal{T}$. Consider any schedule $\mathcal{E} \in \mathcal{V}$. Since at most one resource allocation decision can be chosen in each time slot in $\mathcal{E}$, we have $|\mathcal{E} \cap \mathcal{V}[t]| \le 1 = \nu_t$, $\forall t$. Let $\mathcal{J}$ be the collection of all such feasible schedules $\{\mathcal{E}\}$. Then, the pair $(\mathcal{V}, \mathcal{J})$ containing all schedules forms a partition matroid, and finding an optimal feasible schedule is equivalent to finding an optimal independent set of this partition matroid.

Next, we let $z_i[t] = \sum_{s \in \mathcal{S}} y_{is}[t]$ denote the total number of workers of job i in time-slot t. Clearly, $z_i[t] = 0, \forall t < a_i$. Recalling the definition of $z_{i,t}$, we have $z_{i,t} = z_i[a_i] + z_i[a_i + 1] + \cdots + z_i[t]$. For any schedule $\mathcal{E}$ (could be infeasible), we define $z_i^{\mathcal{E}}[t] \triangleq \sum_{\{y_{is}[t] \in \mathcal{E} \cap \mathcal{V}[t]\}} y_{is}[t]$. Accordingly, we define $z_{i,t}^{\mathcal{E}} \triangleq z_i^{\mathcal{E}}[a_i] + \cdots + z_i^{\mathcal{E}}[t]$. Then, we can rewrite the objective

function in (10) as $F(\mathcal{E}) \triangleq \sum_{i \in \mathcal{I}} \mu_i(\zeta_i z_{i,T}^{\mathcal{E}})$. Since $\mu_i(\cdot)$ is concave and increasing, $F(\mathcal{E})$ is monotonically increasing as t increases. Next, we show that $F(\mathcal{E})$ is also submodular.

Lemma 5. *$F(\mathcal{E})$ is a submodular function.*

Proof. We pick two schedules $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$. Now, for a resource allocation decision $v \in \mathcal{V}[t]$ in time-slot t. Consider the following two cases:

- Case 1) $v \in \mathcal{B}$: In this case, we have $F(\mathcal{A} \cup \{v\}) - F(\mathcal{A}) \ge F(\mathcal{B} \cup \{v\}) - F(\mathcal{B}) = 0$, which holds trivially.
- Case 2) $v \notin \mathcal{B}$: In this case, we have

$$F(\mathcal{B} \cup \{v\}) - F(\mathcal{B}) = \sum_{i \in \mathcal{I}} \left(\mu_i(\zeta_i z_{i,T}^{\mathcal{B}} + \zeta_i z_i^v[t]) - \mu_i(\zeta_i z_{i,T}^{\mathcal{B}}) \right)$$

$$\overset{(a)}{\le} \sum_{i \in \mathcal{I}} \left(\mu_i(\zeta_i z_{i,T}^{\mathcal{A}} + \zeta_i z_i^v[t]) - \mu_i(\zeta_i z_{i,T}^{\mathcal{A}}) \right) = F(\mathcal{A} \cup \{v\}) - F(\mathcal{A}),$$

where (a) follows from $\mu_i(\cdot)$ being concave, monotone, and increasing. Then, the proof is complete by Definition 1. $\square$

The main competitive ratio result of GADGET is stated in the following theorem. Due to space limitation, we provide a proof sketch in this paper.

Theorem 6. *Algorithm 1 produces a schedule that is $\frac{\alpha}{\alpha+1}$-competitive, where $\alpha \in (0, 1]$ is the approximation ratio of solving Problem (10) in each time-slot.*

Proof Sketch. The main idea of our proof is to leverage the p-system result for greedy algorithms with approximation, where p denotes the ratio between the maximum and minimum cardinalities of maximal independent sets in a family of independent sets. It has been shown (e.g., [22]) that applying an online greedy algorithm with an approximation ratio of $\alpha \in (0, 1]$ in each round for a p-system yields a competitive ratio of $\frac{\alpha}{\alpha+p}$. Hence, the result in Theorem 6 is proved once we show that Problem DDLJS is a 1-system ($p = 1$).

Toward this end, recall that the space of all possible schedules $\mathcal{V} = \{y_{is}[t], \forall i, s, t\}$ is partitioned into a collection of disjoint "allocations" $\{\mathcal{V}[t]\}$, $t \in \mathcal{T}$, where $\mathcal{V}[t] = \{y_{is}[t], \forall i, s\}$ and the independence family is $\mathcal{J} = \{\mathcal{H} \in \mathcal{V} : |\mathcal{H} \cap \mathcal{V}[t]| \le 1, \forall t\}$. For any $Y = \{\mathcal{U}[t] \subseteq \mathcal{V}[t], \forall t\} \subseteq \mathcal{V}$, let $\mathcal{B}(Y)$ be the set of maximal independent sets of $\mathcal{V}$ included in Y, which implies that $|\mathcal{H} \cap \mathcal{U}[t]| \le 1$. Then, we can show that $\mathcal{B}(Y) = \{\{u_k[t] \in \mathcal{U}[t], \forall t\}, \forall k\}$, where k is the element (scheduling) index in the set $\mathcal{U}[t]$. Thus for any $A \in \mathcal{B}(Y)$, the cardinality of A is the same (i.e., $|A| = |Y|$), which immediately implies that $p = 1$ and the proof is complete. $\square$

C. Solving the Per-Time-Slot Problem in (10)

Under the temporally greedy approach in Algorithm 1, it remains to solve an NP-Hard problem in (10). Due to the challenges in directly handling the ring topological constraint in (9) that is mixed-integer and highly unstructured, we take an "indirect" approach by noting that Problem (10) is a *generalized* virtual network embedding (VNE) problem (embedding a virtual computational graph onto a substrate physical graph while respecting all capacity constraints). Notably, VNE with

request graphs of cactus-type topologies has been solved in [5], which includes the ring topology as a special case. The basic idea of the solution is based on randomized rounding the routing graphs over the underlying multi-commodity flow (MCF) problem (see, e.g., [5], [23] for details).

However, our work differs from standard VNE [5] in the following two key aspects: 1) Unlike in standard VNE where the number of nodes in each request graph is given, the number of nodes (i.e., workers) is part of the problem in (10); 2) Unlike standard VNE with a one-dimensional resource capacity constraint, Problem (10) has *multi-dimensional* resource capacity constraints. Due to these differences, we refer to Problem (10) in this paper as the *generalized* VNE (G-VNE).

To address these challenges, we again take a *"divide-and-conquer"* approach: i) We observe that, although the number of workers is unknown, its upper bound $q_i[t]$ can be obtained efficiently by solving Problem (10) with continuous relaxations of Constraints (2), (4), and (11); ii) Once the upper bound of the number of workers is known, we can reformulate the G-VNE problem with a "one-hot" worker number constraint.

To reformulate the G-VNE problem, we let $\rho_i[t] \in \{0, 1\}$ be the binary variable to indicate whether job i is embedded in time slot t. Let $\mathcal{Q}_i[t] \triangleq \{1, \ldots, \lceil q_i[t] \rceil\}$ be the set of all possible numbers of workers at time t for job i. Let binary variable $\chi_{i,\kappa}[t] \in \{0, 1\}$ denote whether a ring of size $\kappa \in \mathcal{Q}_i[t]$ is chosen at time t. We use V_i^κ and E_i^κ to represent the sets of nodes and edges of the chosen κ-ring, respectively. Let $\varrho_{i,\kappa,a}^u[t]$ and $\theta_{i,\kappa,a,b}^{u,v}[t]$ denote whether node $a \in V_i^\kappa$ and edge $(a, b) \in E_i^\kappa$ are mapped to a physical node $u \in V_s$ and a physical link $(u, v) \in E_s$ at time t, respectively.

Next, we use $d_{i,\kappa}^{a,r}$ and $d_{i,\kappa}^{a,b}$ to denote the type-r resource demand for node $a \in V_i^\kappa$ and resource demand for edge $(a, b) \in E_i^\kappa$ in the chosen κ-ring of job i, respectively. Let $g_s^{u,r}$ and $e_s^{u,v}$ be the type-r resource capacities of node $u \in V_s$ and edge $(u, v) \in E_s$ in the substrate network, respectively. Also, let $h_i^{u,r}[t]$ and $o_i^{u,v}[t]$ denote the cumulative type-r resource allocation on node u and the cumulative and aggregated resource allocation on edge (u, v) of job i in the substrate network at time t, respectively. Let $\pi_{i,\kappa}[t]$ be the incremental utility with the κ-ring chosen at time t, i.e., $\pi_{i,\kappa}[t] = \mu_i(\zeta_i z_{i,t-1} + \zeta_i \sum_{\kappa \in \mathcal{Q}_i[t]} \kappa \chi_{i,\kappa}[t]) - \mu_i(\zeta_i z_{i,t-1})$. We let $\delta^+(u)$ and $\delta^-(u)$ denote the sets of outgoing and incoming edges of a node u, respectively. Then, the G-VNE problem in time-slot t can be reformulated as an integer linear program (ILP) (omitting time index "$[t]$" for notational simplicity):

$$\text{Maximize} \sum_{i \in \mathcal{I}} \sum_{\kappa \in \mathcal{Q}_i} \pi_{i,\kappa} \chi_{i,\kappa} \tag{12}$$

$$\sum_{\kappa \in \mathcal{Q}_i} \chi_{i,\kappa} = \rho_i, \quad \forall i \in \mathcal{I}, \tag{13}$$

$$\sum_{u \in V_s} \varrho_{i,\kappa,a}^u = \chi_{i,\kappa}, \quad \forall i \in \mathcal{I}, \kappa \in \mathcal{Q}_i, a \in V_i^\kappa, \tag{14}$$

$$\sum_{(u,v) \in \delta^+(u)} \theta_{i,\kappa,a,b}^{u,v} - \sum_{(v,u) \in \delta^-(u)} \theta_{i,\kappa,a,b}^{v,u} = \varrho_{i,\kappa,a}^u - \varrho_{i,\kappa,b}^u,$$
$$\forall i \in \mathcal{I}, \kappa \in \mathcal{Q}_i, (a, b) \in E_i^\kappa, u \in V_s, \tag{15}$$

$$\sum_{\kappa \in \mathcal{Q}_i} \sum_{a \in V_i^\kappa} d_{i,\kappa}^{a,r} \varrho_{i,\kappa,a}^u = h_i^{u,r}, \quad \forall i \in \mathcal{I}, u \in V_s, r \in \mathcal{R}, \tag{16}$$

$$\sum_{\kappa \in \mathcal{Q}_i} \sum_{(a,b) \in E_i^\kappa} d_{i,\kappa}^{a,b} \theta_{i,\kappa,a,b}^{u,v} = o_i^{u,v}, \forall i \in \mathcal{I}, (u, v) \in E_s, \tag{17}$$

$$\sum_{i \in \mathcal{I}} h_i^{u,r} \leq g_s^{u,r}, \quad \forall u \in V_s, \forall r \in \mathcal{R}, \tag{18}$$

$$\sum_{i \in \mathcal{I}} o_i^{u,v} \leq e_s^{u,v}, \quad \forall (u, v) \in E_s. \tag{19}$$

Here, Constraint (13) ensures that at most one of the κ-ring graphs ($\kappa = 1, \ldots, \lceil q_i \rceil$) from $\mathcal{Q}_i$ can be selected. Constraint (14) ensures that if job i is embedded ($\rho_i = 1$) and if a κ-ring graph is selected ($\chi_{i,\kappa} = 1$), then each node $a \in V_i^\kappa$ must be mapped to some physical node in V_s, i.e., there exists a node $u \in V_s$ such that $\varrho_{i,\kappa,a}^u = 1$. Constraint (15) induces a non-splittable unit flow for each edge $(a, b) \in E_i^\kappa$ from the physical location that a is mapped to the physical location that b is mapped. Constraints (16)–(17) compute the cumulative resource on physical nodes and edges, respectively. Constraints (18)–(19) ensure no violation of the physical nodes and edges resource capacities, respectively. Note that Constraints (13), (16) and (18) are the *major differences* from the MCF formulation of the standard VNE problem (see, e.g., [5]), where each job has only one request graph (i.e., $|\mathcal{Q}_i| = 1$) and each node has only one resource type.

Now, consider a linear program (LP)-based ring selection approach: Let $\{\bar{\chi}_{i,\kappa}, \bar{\rho}_i\}$ be the LP-relaxation solution of Problem (12) and $\bar{\pi}_{i,k}$ be the its utility value. Select $\kappa_i = \max_{\kappa \in \mathcal{Q}_i : \bar{\chi}_{i,\kappa} > 0} \{\bar{\pi}_{i,\kappa} \bar{\chi}_{i,\kappa}\}$ as the request ring-graph of job i for $\bar{\rho}_i > 0$. That is, we set $\chi_{i,\kappa_i} = 1$ and $\chi_{i,\kappa} = 0$ if $\kappa \neq \kappa_i$, for each job with $\bar{\rho}_i > 0$. Then, we can show that this LP-based ring selection scheme has the following approximation ratio:

Lemma 7 (LP-based Ring Selection). *Let* $\Gamma = \max_i \lceil q_i \rceil$. *The LP-based ring selection scheme achieves at least a $1/\Gamma$-fraction of the utility obtained by an offline optimal approach.*

Proof. Let χ_{i,κ^*}^*, π_{i,κ^*}^* be the optimal solution and objective value with κ^* being the optimal ring size, respectively. Let $\hat{\chi}_{i,\hat{\kappa}}$, $\hat{\pi}_{i,\hat{\kappa}}$ be the solution and objective value of our LP-based scheme, respectively, with $\hat{\kappa}$ being the ring size of the LP-based scheme. Let $\Pi_i \triangleq \min\{N_i, \frac{C_s^r}{l_i^r}, \frac{F_i^r}{l_i^r}, \forall r, s\}, \forall i$. Then,

$$\frac{\sum_i \pi_{i,\kappa^*}^* \chi_{i,\kappa^*}^*}{\sum_i \hat{\pi}_{i,\hat{\kappa}} \hat{\chi}_{i,\hat{\kappa}}} \leq \frac{\sum_i \sum_\kappa \bar{\pi}_{i,\kappa} \bar{\chi}_{i,\kappa}}{\sum_i \hat{\pi}_{i,\hat{\kappa}} \hat{\chi}_{i,\hat{\kappa}}} \overset{(a)}{\leq} \max_i \frac{\sum_\kappa \bar{\pi}_{i,\kappa} \bar{\chi}_{i,\kappa}}{\hat{\pi}_{i,\hat{\kappa}} \hat{\chi}_{i,\hat{\kappa}}} \overset{(b)}{\leq} \Gamma.$$

To see why (a) holds, let $\bar{\iota}_i \triangleq \sum_\kappa \bar{\pi}_{i,\kappa} \bar{\chi}_{i,\kappa}$ and $\hat{\iota}_i \triangleq \hat{\pi}_{i,\hat{\kappa}} \hat{\chi}_{i,\hat{\kappa}}$. Let $i^* = \arg\max_i \left\{ \frac{\sum_\kappa \bar{\pi}_{i,\kappa} \bar{\chi}_{i,\kappa}}{\hat{\pi}_{i,\hat{\kappa}} \hat{\chi}_{i,\hat{\kappa}}} \right\}$. Thus, we have $\frac{\bar{\iota}_{i^*}}{\hat{\iota}_{i^*}} - \frac{\sum_i \bar{\iota}_i}{\sum_i \hat{\iota}_i} = \frac{\bar{\iota}_{i^*} \sum_i \hat{\iota}_i - \hat{\iota}_{i^*} \sum_i \bar{\iota}_i}{\hat{\iota}_{i^*} \sum_i \hat{\iota}_i} = \frac{\sum_i (\hat{\iota}_{i^*} \bar{\iota}_i - \bar{\iota}_i \hat{\iota}_{i^*})}{\hat{\iota}_{i^*} \sum_i \hat{\iota}_i} \geq 0$. Also, (b) follows from i) the LP relaxation: $\frac{\bar{\pi}_{i,\kappa} \bar{\chi}_{i,\kappa}}{\hat{\pi}_{i,\hat{\kappa}} \hat{\chi}_{i,\hat{\kappa}}} \leq 1, \forall i, \forall \kappa \in \mathcal{Q}_i)$; and ii) $|\mathcal{Q}_i| = \lceil q_i \rceil \leq \Pi_i$. This completes the proof. $\square$

Next, upon determining the ring size κ, we perform virtual network embedding with multi-dimensional resource constraints. First, similar to [5, Formulation 2], we resolve the uncertainty that may occur in embedding cyclic graphs by creating an augmented LP [5] that binds multiple copies of a

"reduced version" of Problem (12), where the set of $\hat{\kappa}$-rings has been chosen following the LP-based ring selection. Then, we solve the augmented LP to obtain a relaxation solution, which can be used to recover a set of mapping-selection tuples $\mathcal{M}_i = \{(\varphi_i^k, \omega_i^k), k = 1, \ldots, |\mathcal{M}_i| : \varphi_i^k > 0, \sum_k \varphi_i^k \leq 1\}$ for each job i, where the mapping ω_i^k (a candidate of embedding) is chosen with probability φ_i^k, and rejected with probability $1 - \sum_k \varphi_i^k$. Note that to address the *multi-dimensional resource* challenge in computing $\mathcal{M}_i$, our key idea is to conduct mapping search to determine $\omega_i^k(r)$ for each $r \in \mathcal{R}$ by leveraging techniques in [5, Sec. III-C]. Then, we choose $\omega_i^k = \cap_{r \in \mathcal{R}} \omega_i^k(r)$. After the set $\{\mathcal{M}_i, \forall i\}$ is calculated, we perform randomized rounding with probabilities based on φ_i^k-values to obtain the embedding. Putting all these together, we summarize our LP-based ring-selection and multi-dimensional resource embedding (LP-RS-MDE) method in Algorithm 2.

Algorithm 2: LP-based Ring-Selection and Multi-Dimensional Resource Embedding (LP-RS-MDE).

---

1 **Initialization:** Choose u_b as the maximum number of roundings, and set $iter \leftarrow 1$;
2 Set the values of constants α, β^r, γ (see Theorem 8);
3 Solve the LP relaxation of Problem (12) and set ring sizes $\kappa_i = \max_{\kappa \in \mathcal{Q}_i : \bar{\chi}_{i,\kappa} > 0} \{\bar{\pi}_{i,\kappa} \bar{\chi}_{i,\kappa}\}$ for all job i;
4 Solve an augmented LP with ring sizes fixed in Step 3;
5 **for** $i \in \mathcal{I}[t]$ **do**
6   Compute $\mathcal{M}_i = \{(\varphi_i^k, \omega_i^k)\}$ based on the solution of the augmented LP, where φ_i^k, ω_i^k are determined by [5, Sec. III-C] and the bottleneck resource;
7 **while** *solution is not* $(\alpha, \beta^r, \gamma)$-*approx.* & $iter < u_b$ **do**
8   **foreach** $i \in \mathcal{I}[t]$ choose ω_i^k with probability φ_i^k;
9     $iter \leftarrow iter + 1$;

To analyze LP-RS-MDE's performance, we let $d_{max}^r(i, u)$ and $d_{max}(i, u, v)$ denote the maximal type-r resource demands that job i imposes on node u and edge (u, v) in the substrate network, respectively. Let $C_{max}^r(i, u)$ and $C_{max}(i, u, v)$ denote the maximal type-r resource allocation that a valid mapping of job i may impose on the substrate network's nodes and edges, respectively. We define two constants as follows:

$$\Delta^r(V_s) \triangleq \max_u \sum_{i \in \mathcal{I}[t]: d_{max(i,u)}^r > 0} (C_{max}^r(i, u)/d_{max}^r(i, u))^2, \forall r,$$

$$\Delta(E_s) \triangleq \max_{(u,v)} \sum_{i \in \mathcal{I}[t]: d_{max(i,u,v)} > 0} (C_{max}(i, u, v)/d_{max}(i, u, v))^2.$$

Also, we let $A^{u,r}[t]$ and $A^{u,v}[t]$ represent the overall type-r resource allocation on node $u \in V_s$ and on edge $(u, v) \in E_s$ after randomized rounding at time t, respectively. Then we have following analytical results:

Theorem 8 (Performance of LP-RS-MDE). *Assume that the substrate network has at least three servers ($|V_s| \geq 3$). Then, LP-RS-MDE achieves at least* (1/3)-*fraction of the optimal value of Problem* (12), *with probabilities of resource constraint violations satisfying* $\mathbb{P}\{A^{u,r}[t] \geq \beta^r g_s^{u,r}\} \leq |V_s|^{-4}, \forall r, u, t,$

and $\mathbb{P}\{A^{u,v}[t] \geq \gamma e_s^{u,v}\} \leq |E_s|^{-4}, \forall (u, v), t,$ *where* $\beta^r = 1 + \epsilon\sqrt{2\Delta^r(V_s) \log(|V_s|)}, \forall r,$ *and* $\gamma = 1 + \epsilon\sqrt{2\Delta(E_s) \log(|E_s|)}.$

To prove Theorem 8, note that unlike the single-dimensional resource capacity in standard VNE [5], the analysis of Algorithm 2 needs to consider multi-dimensional resource capacity violation (see Eqs. (16) and (18)). Toward this end, we observe that the mapping computation in Step 6 of Algorithm 2 implies that a similar approach as in [5] can be used to analyze the final mapping for each type-r resource in each job, which in turn leads to the results stated in Theorem 8. We omit the proof details here due to space limitation. Next, we analyze the running time complexity of LP-RS-MDE.

Theorem 9 (Time Complexity of LP-RS-MDE). *LP-RS-MDE has polynomial time complexity* $O((|\mathcal{I}[t]| \cdot |G_s|)^3 + u_b \cdot \mathcal{I}[t])$.

Proof. The main components of the running time include solving the LP relaxations to obtain fractional solutions (Lines 3-4 in Algorithm 2), computing valid mappings using decomposition approach (Lines 5-6 in Algorithm 2), and then performing randomized rounding on these computed mappings (Lines 7-9 in Algorithm 2). The time complexity of solving the LP relaxations is $O((|\mathcal{I}[t]| \cdot |G_s|)^3)$ [24]. The mapping search step terminates when $\varrho_i \leq 0$, and in each termination, at least one variable becomes 0. The number of variables for each job i is bound by $O(|G_i| \cdot |G_s|)$ for each copy of Problem (12). Specifically, the mapping search visits all nodes and edges in the request graph. Also, in each visit, in order to find the set of mapping-selection tuples $\{\mathcal{M}_i, \forall i\}$, the substrate network will be traversed. Thus, the total number of variables is upper bounded by $O(|G_i| \cdot |G_s|)$. Further, there are at most $O(|V_s|)$ copies of Problem (12), which implies $O(|G_i| \cdot |G_s| \cdot |V_s| \cdot |\mathcal{I}[t]|)$ complexity. Finally, the rounding time of each job i is bounded by u_b, and thus the running time is bounded by $O(u_b \cdot \mathcal{I}[t])$. Hence, the overall time complexity of the LP-RS-MDE method is $O((|\mathcal{I}[t]| \cdot |G_s|)^3 + u_b \cdot \mathcal{I}[t])$, and the proof is complete. $\square$

By combining results in Theorems 6 and 8, we have:

Theorem 10 (Competitive Ratio of GADGET). *Algorithm 1 produces a schedule that yields a utility value at least* $(\frac{1}{3\Gamma+1})$-*fraction of the maximum utility value of Problem DDLJS, with probabilities of resource violations satisfying* $\mathbb{P}\{A^{u,r}[t] \geq \beta^r g_s^{u,r}, \exists t \in \mathcal{T}\} \leq 1 - (1 - |V_s|^{-4})^T, \forall u, r,$ *and* $\mathbb{P}\{A^{u,v}[t] \geq \gamma e_s^{u,v}, \exists t \in \mathcal{T}\} \leq 1 - (1 - |E_s|^{-4})^T, \forall (u, v),$ *where* Γ *and* β^r, γ *are as defined in Lemma 7 and Theorem 8.*

VI. EMPIRICAL STUDIES

In this section, we conduct simulations to evaluate our GADGET algorithm. We use "excessive training avoidance" as an application example (cf. Section IV, Page 5), which aims to maximize the overall utility in a GPU computing cluster. Here, Eq. (3) is specialized to $\sum_t \sum_s y_{is}[t] \leq F_i, \forall i,$ where F_i is the maximum number of iterations specified by users upon their job submissions. Eq. (4) is specialized to $\sum_i y_{is}[t] \leq C_s, \forall s, t,$ where C_s is the GPU capacity of server s.
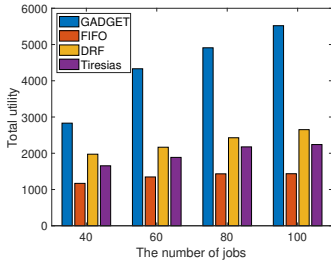
Fig. 4: Total utility comparisons.

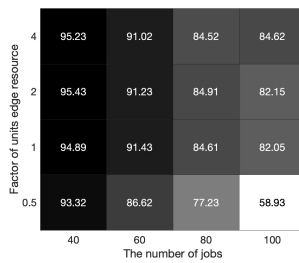Fig. 5: Embedded ratio overview w.r.t node resource.

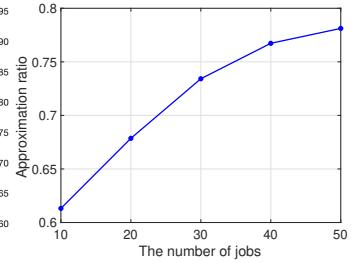Fig. 6: Embedded ratio overview w.r.t edge resource.

Fig. 7: Approximation ratio.

1) Experiment Settings: We use a "fat-tree" GPU computing cluster running for $T = 200$, with $S = 50$ servers. The servers are randomly divided into "racks" and the number of racks is randomly chosen between $[2, 5]$. Following similar settings as in [25], we configure each server with the number of GPUs randomly chosen from the discrete set $\{1, 2, 4, 8\}$. We adopt the job arrival pattern from the Google Cluster data [26]. The job parameters are integers generated uniformly at random from the following intervals: $N_i \in [1, 5]$, $F_i \in [1000, 6000]$, $\zeta_i \in [50, 500]$, and $b_i \in [100 \text{ Mbps}, 5 \text{ Gbps}]$. The bandwidths between racks and servers are chosen from $[200 \text{ Gbps}, 3200 \text{ Gbps}]$ [27] and $[10 \text{ Gbps}, 100 \text{ Gbps}]$ uniformly at random, respectively. We adopt the Sigmoid utility function [28]: $\mu_i(y_{is}[t]) = \frac{\lambda_{i,1}}{1 + e^{-\lambda_{i,2}(\zeta_i \sum_{s \in S} y_{is}[t] + \zeta_i z_{i,t-1} - \lambda_{i,3})}}$, where $\lambda_{i,1} \in [1, 100]$ represents the priority of job i, $\lambda_{i,2} \in (0, 1)$ is to represent the sensitivity of the job to the number of iterations, and $\lambda_{i,3} \in [300, 3000]$ represents the expected number of iterations that should be trained.

2) Baselines: We compare GADGET with the following representative job scheduling policies for computing clusters:

- *FIFO in Hadoop and Spark [29]:* Jobs are scheduled in the order of their arrivals with a fixed number of workers.
- *Dominant Resource Fairness Scheduling (DRF) in Yarn [30] and Mesos [31]:* Jobs are scheduled based on their dominant resource share.
- *Least Attained Service (LAS) in Tiresias [15]:* Resource scheduling among active users in the cluster is conducted in a round-robin fashion across jobs, according to the total number of accelerator hours consumed by each job.

Since the above scheduling policies do not consider the underlying topology constraints, we place workers based on the simple heuristic that greedily allocates workers to servers. where a cycle can be attained. For FIFO and Tiresias, the number of workers is fixed to a number within $[1, 10]$.

3) Experiment Results: Fig. 4 illustrates the comparisons of our GADGET algorithm to the above baselines. We can see that GADGET algorithm significantly outperforms the baseline algorithms, and the gains in total utility value over the baselines become more pronounced as the number of jobs increases. This shows that the dynamic resource allocation in GADGET achieves higher resource utilization than those of the static resource allocations (i.e., the number of workers remains fixed throughout the training process) in the baselines.

Next, we examine the impacts of node GPU and edge bandwidth resource capacities in the substrate network on the performance of GADGET. The results are shown in Figs. 5 and 6, where one unit of node and edge resource in the y-axes represents (GPU=100, bandwidth=200Gbps). We evaluate the ratio between the numbers of embedded jobs and active jobs in each time slot, and the results in Figs. 5 and 6 are the average ratio over three trials. We can see that increasing the node and edge capacities in the cluster has a positive impact, which allows more jobs to be embedded in each time slot. Intuitively, as the GPU capacity of each node increases, jobs with high bandwidth demands have a higher probability to be allocated using intra-server communication, which typically has a much larger bandwidth capacity than that of inter-server communication. Also, the larger the GPUs number, the more jobs can be trained simultaneously. Similarly, the larger the edge capacity, the more jobs with inter-server communications can be scheduled.

Lastly, we investigate the performance of our proposed G-VNE technique, which is a major component of our GADGET algorithm. We evaluate the embedding performance in terms of the ratio between the total utility obtained by our algorithm and the optimal total utility. The optimal utility of Problem (12) at each time slot is computed using the global optimization solver Gurobi based on the branch-and-bound approach (of exponential complexity) [32]. The results are shown in Fig. 7. We can see that the actual performance ratio is better than our theoretical bound in Theorem 8, which achieves 60%–80% of the optimal utility obtained by the global optimization solver.

VII. CONCLUSION

In this paper, we studied online resource scheduling for the training of RAR-based DDL jobs in computing clusters. We first developed an analytical optimization framework and then developed an online scheduling algorithm called GADGET In GADGET, by showing the temporal submodularity of the online scheduling problem, we developed a greedy scheduling approach with competitive ratio guarantee. Then, for the NP-Hard subproblem in each time-slot, we proposed a generalized virtual network embedding technique with approximation ratio guarantee. Trace-driven simulations confirmed the superior performance of GADGET over existing schemes. [4]

---

[4]The authors have provided public access to their code or data at https://zenodo.org/record/5847644#.YervbxNKhTZ.

REFERENCES

[1] M. Abadi, P. Barham *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. of USENIX OSDI*, 2016.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[3] "Amazon EC2 P3 Instances," 2021. [Online]. Available: https://aws.amazon.com/ec2/instance-types/p3/

[4] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Multi-tenant gpu clusters for deep learning workloads: Analysis and implications," Microsoft Research, Tech. Rep., 2018.

[5] M. Rost and S. Schmid, "Virtual network embedding approximations: Leveraging randomized rounding," in *Proc. IFIP Networking*, 2018.

[6] C. Avin and S. Schmid, "Toward demand-aware networking: A theory for self-adjusting networks," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1807.02935

[7] M. Rost, E. Döhne, and S. Schmid, "Parametrized complexity of virtual network embeddings: Dynamic & linear programming approximations," in *SIGCOMM Comput. Commun.*, vol. 49, no. 1, 2019, pp. 3–10.

[8] M. Li, D. G. Andersen *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. of USENIX OSDI*, 2014.

[9] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. of USENIX OSDI*, 2014.

[10] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proc. of ACM KDD*, 2015.

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

[12] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. of IEEE INFOCOM*, 2018.

[13] A. Sergeev and M. D. Balso, "Horovod: Fast and easy distributed deep learning in tensorflow," in *arXiv preprint arXiv:1802.05799*, 2018.

[14] Y. Bao, Y. Peng, Y. Chen, and C. Wu, "Preemptive all-reduce scheduling for expediting distributed DNN training," in *IEEE INFOCOM*, 2020.

[15] 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 *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 485–500.

[16] H. Zheng, F. Xu, L. C. andZ. Zhou, and F. Liu, "Cynthia: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training," in *2019 48th International Conference on Parallel Processing (ICPP)*, 2019.

[17] X. Mei, X. Chu, H. Liu, Y. Leung, and Z. Li, "Energy efficient real-time task scheduling on cpu-gpu hybrid clusters," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[18] V. Chau, X. Chu, H. Liu, and Y.-W. Leung, "Energy efficient job scheduling with dvfs for cpu-gpu heterogeneous systems," in *Proceedings of the Eighth International Conference on Future Energy Systems, ser. e-Energy '17*, 2017, pp. 1–11.

[19] W. Chen, J. Rao, and X. Zhou, "Preemptive and low latency datacenter scheduling via lightweight containers," in *2017 USENIX Annumal Technical Conference*, 2017.

[20] I. Hou and P. Gupta, "Proportionally fair distributed resource allocation in multiband wireless systems," in *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, 2014, pp. 1819–1830.

[21] A. Schrijver, "Combinatorial optimization: polyhedra and efficiency," in *Springer-Verlag*, Berlin, 2003.

[22] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a monotone submodular function subject to a matroid constraint," in *SIAM Journal on Computing*, vol. 40, no. 6, 2011, pp. 1740–1766.

[23] P. Raghavan and C. D. Thompson, "Provably good routing in graphs: Regular arrays," in *Proc. 17th ACM STOC*, 1985, pp. 79–87.

[24] P. Vaidya, "An algorithm for linear programming which requires $o(((m + n)n^2 + (m + n)^{1.5}n)L)$ arithmetic operations," in *Mathematical Programming*, 1990, pp. 175–201.

[25] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient gpu cluster scheduling," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 289–304.

[26] C. Reiss, A. Tumanov *et al.*, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. of ACM SoCC*, 2012.

[27] S. Kipp, "Data center bandwidth scenarios." [Online]. Available: https://www.ieee802.org/3/ad_hoc/bwa/public/may11/kipp_01_0511.pdf

[28] Z. Huang, B. Balasubramanian, M. Wang, T. Lan, M. Chiang, and D. H. Tsang, "Need for speed: Cora scheduler for optimizing completiontimes in the cloud," in *Proc. of IEEE INFOCOM*, 2015.

[29] M. Zaharia, M. Chowdhury, M. J. Franklin, and et al, "Spark: Cluster computing with working sets," in *n Proc. of USENIX HotCloud*, 2010.

[30] V. K. Vavilapalli, A. C. Murthy *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proc. of ACM SoCC*, 2013.

[31] B. Hindman, A. Konwinski *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. of USENIX NSDI*, 2011.

[32] K. Kobayashi and Y. Takano, "A branch-and-cut algorithm for solving mixed-integer semidefinite optimization problems," in *Computational Optimization & Applications*, vol. 2, no. 75, 2020, pp. 493–513.