Non-Idle Machine-Aware Worker Placement for Efficient Distributed Training in GPU Clusters

Jin Fang[1,2] Gongming Zhao[1,2] Hongli Xu[1,2] Luyao Luo[1,2] Zhen Yao[3] An Xie[3]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Suzhou Institute for Advanced Research, University of Science and Technology of China
[3] Huawei Technologies Co., Ltd.

***Abstract*—Distributed training (DT) has emerged as a solution to address the growing computational resource demands of training large-scale machine learning models. To meet this need, major cloud providers typically build GPU clusters to accommodate DT jobs. Specifically, for an incoming DT job request, cloud providers need to determine in which GPUs place workers (*i.e.*, worker placement). Existing approaches usually place workers on as few idle machines as possible to minimize communication time. However, this scheme will lead to *resource fragmentation problem*, which degrades the efficiency of the GPU cluster and increases training costs for cloud providers.**

In this paper, we propose Titan, a novel worker placement scheme that mitigates the influence of resource fragmentation by enhancing the utilization of non-idle machines. Titan formulates a multi-objectives non-linear optimization problem that incorporates the collective communication constraint and proves its NP-hardness. To solve the problem, Titan presents an effective submodular-based greedy algorithm with a tight approximation ratio ($1 - \frac{1}{e}$). We evaluate Titan with a large-scale simulation employing real-world job traces and a small-scale testbed consisting of 8 servers with 32 logical GPUs. Experimental results show that Titan can achieve near-optimal training throughput while improving the efficiency of the cluster by 74.9% compared to the state-of-the-art solutions.

***Index Terms*—Distributed Training, Worker Placement, Datacenter Network, GPU Cluster**

I. Introduction

The demand for training machine learning (ML) models has increased dramatically in recent years due to the growing complexity of modern ML applications (*e.g.*, real-time translation [1], text summarization [2] and artificial intelligence agents [3]). To satisfy the massive computing requirements, distributed training (DT) has emerged as a promising solution. One DT job consists of multiple computing nodes (workers) to train part of models and synchronize the model iteratively [4]–[6], potentially consuming hundreds or even thousands of GPUs [7]. To this end, mainstream cloud providers build large-scale GPU clusters to serve DT jobs [7]–[9]. As a result, it is important for cloud providers to decide on which GPUs place workers of the DT job (*i.e.*, worker placement).

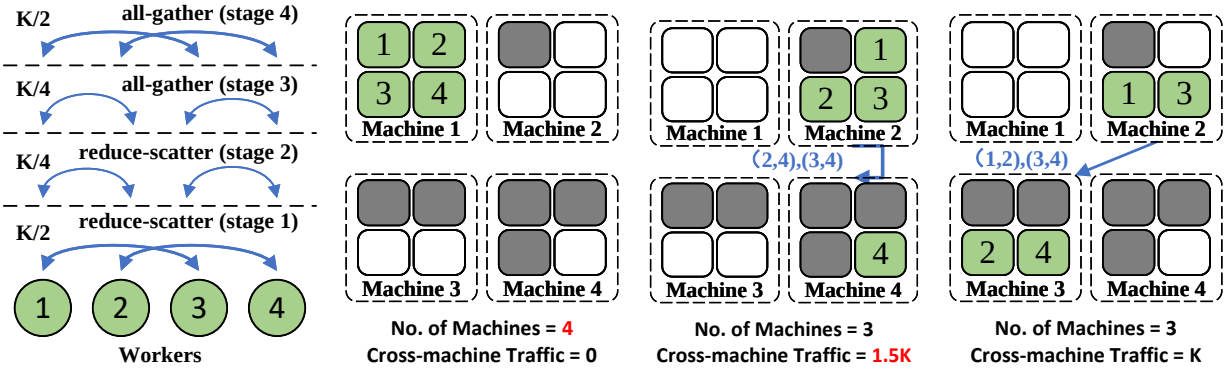
Existing worker placement solutions focus on minimizing job completion time (JCT) by reducing the amount of cross-machine traffic [10], [11], or bandwidth contention [12]–[14]. Firstly, bandwidth among machines is relatively limited (*e.g.*, 100Gbps link) compared to the bandwidth within a machine (*e.g.*, 1.8Tbps NVlink [15]). Therefore, some studies [10], [11] place workers on the least number of machines to minimize cross-machine traffic amount and improve DT performance. Secondly, workers need to perform collective communication operations during the training for parameter synchronization. These operations (*e.g.*, allreduce or allgather) usually involve workers communicating simultaneously, leading to traffic congestion. To address this, some works [12], [13], [16] place workers of a DT job based on their communication patterns and machine network topology, to effectively reduce bandwidth contention.

Due to the unpredictability of job arrival and completion, these schemes may increase the resource fragmentation rate in the GPU cluster. Similar to work [17], we define *fragmented resources* as machines that are not fully utilized (*i.e.*, the number of idle GPUs is less than the total number of GPUs). High resource fragmentation will incur the ability of serving large-scale DT jobs in the GPU cluster, since the manager can not allocate enough contiougus GPUs in idle machines. We give an example to illustrate the impact of fragmentation. Assuming the number of workers of DT jobs varies from 1 to 16, and each machine is equipped with 8 GPUs. The cluster has three machines where two of them have been placed 4 workers each. Under this scenario, when a 16-worker job comes, it can not be scheduled for worker placement with the objective of minimizing the cross-machine traffic. The reason is that there are not enough idle machines (*i.e.*, 2 idle machines) for worker placement. Due to resource fragmentation, this job may suffer long queueing delays, even if there are enough number of idle GPUs. From the above discussion, we can see that a worker placement scheme decreasing resource fragmentation is in urgent need.

To this end, we propose Titan, *which prioritizes non-idle machines for worker placement to reduce the resource fragmentation of the GPU cluster*. The core of Titan is two objectives. Our primary objective is to minimize the number of used idle machines when placing workers. By doing so, we intend to improve the utilization of non-idle machines while saving more idle machines. Our secondary objective is to minimize the total number of used machines (both idle and non-idle machines). In this way, we consolidate workers of a DT job to minimize the cross-machine traffic, so that we can guarantee high training speed for DT jobs.

However, it is non-trivial to realize Titan. First, the idle GPUs of machines are often highly fragmented in GPU

979-8-3503-0322-3/23/$31.00 ©2024 IEEE



(a) Communication Pattern Example (b) Elasticflow (c) HiveD (d) Titan (Ours)

Fig. 1: The first subplot shows the procedure of the halving-doubling (HD) algorithm under 4 workers. We denote the total size of the parameter as K. The bidirectional arrows represent the corresponding communication pairs and the thickness of the lines distinguishes the communication volume. From the second to fourth subplots, we present the placement results of Elasticflow, HiveD and Titan under a 4-machine cluster. Each machine is equipped with 4 GPUs, denoted by rectangles and the gray rectangles denote non-idle GPUs.

clusters [18], making it difficult to decide whether to use GPUs in idle or non-idle machines for efficient worker placement. Second, the communication pattern of collective communication operations varies depending on the corresponding communication algorithms (see Sec. II-B for details). We need to consider different communication patterns when deciding the worker placement to avoid traffic congestion. To overcome the above challenges, we formulate the resource-aware worker placement problem and present a submodular algorithm. The main contributions of this paper are summarized as follows:

1) We design Titan, which performs non-idle machine-aware worker placement to optimize the resource fragmentation problem and improve non-idle machine utilization for cloud providers.
2) We formulate the worker placement problem as a multi-objective non-linear optimization problem, named NIMP problem, and prove its NP-hardness.
3) We present a submodular-based solution. We prove that our algorithm is neal-optimal and bounded by a tight approximation factor of $(1 - \frac{1}{e})$.
4) We conduct a large-scale simulation using real-world job traces and a small-scale testbed based on 8 servers. Both experimental and simulation results show that Titan can achieve near-optimal training throughput and improve the average profit rate by up to 74.9% compared with the state-of-the-art solutions.

II. Background and Motivation

A. Current Defragmentation Solutions and Limitations

Resource fragmentation has been widely observed in production clusters that run diverse DT jobs [8], [19], [20]. For example, in ByteDance, training jobs experience long queuing time with an average of over 3000s, due to the resource fragmentation problem [20]. Moreover, fragmented resources also increase the management cost of the GPU cluster, since the cluster manager needs to migrate small jobs to free up enough continouse GPUs for large-scale DT jobs. This problem has become a thorny point in reducing the total cost of GPU clusters.

Currently, several DT worker placement solutions [10], [11], [21], [22] are proposed to address this problem. One intuitive solution is to prioritize placing workers in fragmented machines [21]. However, since the fragmented GPUs are typically spread across multiple machines, and workers need to communicate frequently, this approach will lead to substantial cross-machine traffic, resulting in communication bottlenecks. Alternatively, some schemes overprovision resources during DT worker placement to avoid resource fragmentation [11], [22]. For example, Elasticflow [11] restricts the number of workers of each job to be a power of two. If one job does not satisfy this constraint (*e.g.*, 5 workers), the cluster scheduler will allocate additional idle GPUs to avoid resource fragmentation (*e.g.*, allocate 8 GPUs). Though this scheme guarantees no resource fragmentation, it does not improve cluster utilization.

B. A Motivating Example

In this section, we give an example to show the advantages and disadvantages of the state-of-the-art solutions, which motivates our research. We first illustrate the procedure of communication operations, then show the performance of existing placement algorithms.

Collective communication algorithms. In practice, parameter synchronization are performed by collective communication operations (*e.g.*, allreduce or allgather). To mitigate the communciation overhead, different collective communication algorithms (*e.g.*, ring-allreduce [23] in NCCL [24], or halving-doubling [25] in ACCL [26]) are proposed for suiting problem scales and network topologies. Specifically, one collective communication algorithm usually takes several stages, where each stage consists of several disjoint worker sets for communication. The worker sets change in each stage so that all workers can communicate with each other after all stages and synchronize the parameters.

Existing placement schemes. Consider a distributed training task with 4 workers, where the size of parameters is K. The communication procedure of the halving-doubling (HD) algorithm [25] under 4 workers (W_1-W_4) is shown in Fig. 1(a). In this example, it takes 4 stages to finish, and the traffic amounts of communication pairs are $\frac{K}{2}$, $\frac{K}{4}$, $\frac{K}{4}$ and $\frac{K}{2}$, respectively. Supposing there are 4 machines in a GPU cluster, where the number of idle GPUs is 4, 3, 2, and 1, respectively. All machines are connected with 100Gbps links.

Firstly, Elasticflow [11] places workers in as few as possible machines, and the placement result is shown in Fig. 1(b). With this scheme, all workers are placed in Machine 1. The cluster launches 4 machines and there is no cross-machine traffic.

Secondly, HiveD [21] prioritizes utilizing scattered idle GPUs for worker placement. In this case, it will place workers W_1-W_3 in Machine 2 and worker W_4 in Machine 4. As a result, Machines 1 and 4 are fully utilized, and the number of launched machines is 3. However, since W_4 needs to communicate with W_2 and W_3, the amount of cross-machine traffic is $(\frac{K}{2} + \frac{K}{4}) \cdot 2 = \frac{3K}{2}$.

C. Our Intuition

From the above example, we observe that both worker placement schemes have advantages and disadvantages. Elasticflow minimizes cross-machine communication by placing workers in as few as possible machines while ignoring the resource fragmentation problem. HiveD utilizes fragmented GPUs for worker placement which incurs a massive cross-machine traffic and becomes the bottleneck of the DT job. A question immediately following the above discussion is *how to reduce the fragmentation of the GPU cluster by worker placement with the performance guarantee?*

We notice that during the allreduce operation, the traffic amount between workers is decided by the corresponding algorithm. Therefore, as shown in Fig. 1(c), we place W_1 and W_3 in Machine 2, and W_2 and W_4 in Machine 3. The total amount of cross-machine traffic is $(\frac{K}{4} + \frac{K}{4}) \cdot 2 = K$, produced by $\{W_1, W_2\}$ and $\{W_3, W_4\}$. In this way, we reduce the cross-machine traffic by 33.3%, compared with HiveD. Moreover, we reduce the number of launched machines by 25%, compared with Elasticflow. Motivated by this example, we design a novel worker placement scheme utilizing non-idle machines, called Titan.

III. Problem Formulation

A. Network Model

GPU cluster. In practice, one GPU cluster is composed of a set of machines $S = \{s_1, s_2, \ldots, s_{|S|}\}$, each of which is equipped with K GPUs. These machines are interconnected through the datacenter network. Let $P_{s,s'} \in \mathbb{Z}$ denote the available bandwidth between machines s and s'. Considering that there may be other distributed training tasks occupying GPUs in the cluster, we classify these machines into two categories: *non-idle machine* and *idle machine*, based on the number of idle GPUs. We use $S_f \subseteq S$ to denote the set of non-idle machines, where some GPUs in machine $s \in S_f$ are occupied (*i.e.*, $R_s < K$). We use $S_n \subseteq S$ to denote the set of idle machines, in which all GPUs are idle (*i.e.*, $R_s = K, \forall s \in S_n$). Obivously, $S_f \cup S_n = S$.

TABLE I: Important Notations

Notations	Semantics
N	the set of workers
S	the set of machines
S_f	the set of non-idle machines
S_n	the set of idle machines
T	the total number of phases of the allreduce communication
$C^t_{n,n'}$	the traffic amount between workers n and n' in phase t
$P_{s,s'}$	the bandwidth between machines s and s'
R_s	the number of idle GPUs of machine s
x^s_n	whether worker n is placed on machine s or not
y_s	whether machine s is used or not

Communication pattern. A distributed training task partitions the dataset into multiple sub-datasets to train by a set of workers $N = \{n_1, n_2, \ldots, n_{|N|}\}$ (*i.e.*, data parallel training). In each epoch, workers need to perform collective communication operations according to corresponding algorithms to synchronize the model. The communication operation usually requires T phases, with each phase consisting of multiple worker communication pairs. The traffic amount between worker communication pair (n, n') in phase $t \in T$ is represented by $C^t_{n,n'} \in \mathbb{Z}$. Based on this definition, we can simulate the communication pattern of different collective communication algorithms.

B. Problem Formulation

This section gives the formulation of the non-idle machine-aware worker placement (NIMP) problem. To launch a distributed training task, we need to place workers on the corresponding GPUs. Let $x^s_n \in \{0, 1\}$ represent whether worker n is placed on the GPU of machine s, or not. In our settings, we assume that one worker will occupy one GPU exclusively (*i.e.*, we do not consider the situation of GPU sharing). Let $y_s \in \{0, 1\}$ represent whether machine s is used by the DT job, or not. We summarize the notations in Table I.

Constraints. We mainly consider the following constraints.

1) *Placement Constraint*: Each worker should be placed on one and only machine. That is, $\sum_{s \in S} x^s_n = 1, \forall n \in N$.
2) *Resource Constraint*: For each machine, the number of placed workers should not exceed the number of its idle GPUs. It follows $\sum_{n \in N} x^s_n \leq R_s, \forall s \in S$.
3) *Bandwidth Constraint*: To avoid network congestion, the traffic amount of each worker communication pair should not exceed the bandwidth between placed machines, which means $\sum_{n \in N} \sum_{n' \in N} x^s_n \cdot x^{s'}_{n'} \cdot C^t_{n,n'} \leq P_{s,s'}, \forall s, s' \in S, t \in T$.

Objectives. We have two objectives. The primary objective is to minimize the number of idle machines used to reduce the resource fragmentation of the GPU cluster. While achieving the first objective, the second objective (*i.e.*, minimizing the total number of used machines) should be pursued to avoid cross-machine traffic as well as improve the training performance. The problem can be formulated as follows.

$$\min \ O_1 = \sum_{s \in S_n} y_s$$

$$\min \ O_2 = \sum_{s \in S} y_s$$

$$S.t. \begin{cases} \sum\limits_{s \in S} x_n^s = 1, & \forall n \in N \\ \sum\limits_{n \in N} x_n^s \leq R_s, & \forall s \in S \\ \sum\limits_{n \in N} \sum\limits_{n' \in N} x_n^s \cdot x_{n'}^{s'} \cdot C_{n,n'}^t \leq P_{s,s'}, & \forall s, s' \in S, t \in T \\ y_s \geq x_n^s, & \forall s \in S, n \in N \\ x_n^s \in \{0,1\}, & \forall s \in S, n \in N \\ y_s \in \{0,1\}, & \forall s \in S \end{cases} \tag{1}$$

The first set of equations represents the placement constraint. The second set of inequalities indicates the resource constraint. The third set of inequalities shows the bandwidth constraint. The fourth set of inequalities means that one machine will be used if one worker is placed on its GPU. Our first optimization goal is to minimize the number of used idle machines, and the second objective is to minimize the total number of used machines.

Discussion. Ideally, the cloud providers want to achieve the least job completion time given the fixed number of machines. Due to the complexity of DT jobs and physical environment of the cluster, it is diffcult to model the job completion time accurately. Therefore, we turn to focus on the communication ovehead constraint in Eq. 1. We believe it is rational for the following reasons.

One DT job can be splitted into local training phase and parameter synchronize phase. For the local training phase, since we consider the scenario that one worker exclusively occupies one GPU, the training time is relatively stable, regradless of the selection of GPUs. In the parameter synchronize phase, the main communication traffic is incurred by communication operations. Since the bandwidth inside a machine is usually seen as sufficient, we can reduce the parameter synchronize time by restricting the cross-machine traffic. In this way, we reduce the total job completion time.

Theorem 1: The NIMP problem is NP-hard.

Proof: We prove the NP-hardness by reducing it to the Multiple Knapsack Problem (MKP) [27], a well-known NP-hard problem. If we ignore the bandwidth constraint and the first objective, our NIMP problem can be seen as Multiple Knapsack Problem, where each machine s can be seen as a knapsack with resource capacity R_s and each worker n can be viewed as an item with a weight of 1. Under this circumstance, the goal of the problem is to place workers in different machines to maximize machine utilization while satisfying the resource constraints of all the machines. Since the Multiple Knapsack Problem is a special case of our problem, we can conclude that the NIMP problem is NP-hard. ■

The optimization formulation of worker placement with various constraints in Eq. (1) results in a complex multi-objectives non-linear optimization problem that is computationally hard. Despite using a state-of-the-art LP solver (*e.g.*, Gurobi [28]), it still needs an order of hours to solve even for relatively small scales [29]. Thus, designing an efficient algorithm for this problem is challenging.

IV. Algorithm Design

A. Preliminaries

In general, deploying workers in $K = |S|$ machines can be seen as dividing the worker into K disjoint sets, denoted as $\{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_{|S|}\}$, where workers in $\mathcal{N}_s$ are deployed in machine s. We define the set of *remaining machines* $\mathcal{S}_r$ for machines that are not deployed by any workers (*i.e.*, $\mathcal{N}_s = \emptyset, \forall s \in \mathcal{S}_r$). The total number of used machines can be calculated as $\sum_{s \in S} C(\mathcal{N}_s)$, where

$$C(\mathcal{N}_s) = \begin{cases} 1 & s \in S - \mathcal{S}_r \\ 0 & s \in \mathcal{S}_r \end{cases} \tag{2}$$

Obviously, the maximum number of used machines of the total cluster is $C_{\max} = \max\{|S|, |N|\}$. With these notations, we remove the variable y_s and convert the objective O_1 into maximizing the number of remaining machines, as shown in Eq. (3)

$$\max \ O_1 = C_{\max} - \sum_{s \in S_n} C(\mathcal{N}_s)$$

$$\max \ O_2 = C_{\max} - \sum_{s \in S} C(\mathcal{N}_s)$$

$$S.t. \begin{cases} \sum\limits_{s \in S} x_n^s = 1, & \forall n \in N \\ \sum\limits_{n \in N} x_n^s \leq R_s, & \forall s \in S \\ \sum\limits_{n \in N} \sum\limits_{n' \in N} x_n^s \cdot x_{n'}^{s'} \cdot C_{n,n'}^t \leq P_{s,s'}, & \forall s, s' \in S, t \in T \\ x_n^s \in \{0,1\}, & \forall s \in S, n \in N \end{cases} \tag{3}$$

To efficiently solve Eq. (3), we propose a submodular-based algorithm in the following.

B. Algorithm Description

The core idea of our algorithm is through efficient computations of a submodular set function H, which defines the maximum number reduction by merging the workers from several disjoint sets. We define the submodular set function H as follows, and we will prove that the function H is submodular in Section IV-C.

Definition 1: Given the set $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k\}$, which contains k disjoint subsets of worker set N, the reduction of

Algorithm 1 Search for Feasible Worker Set

Step 1: Initialization
Let feasible worker set $A(s) = \emptyset$ for machine s.
Let available worker set $N_c = N - \mathcal{N}$.
Step 2: Iterative update feasible worker sets according to collective communication
Use $\mathcal{C}_{\int,\int'}$ to denote the existing communication overhead between machines s and s'.
for $n \in N_c$ **do**
  for $n' \in \mathcal{N}$ **do**
    for $t \in T$ **do**
      if $C^t_{n,n'} + \mathcal{C}_{\int,\int'} \leq P_{s,s'}$ **then**
        $A(s) \leftarrow A(s) + n$
      end if
    end for
  end for
end for
Output the feasible worker set $A(s)$ for machine s.

deployed machine number achieved by merging the worker subset from $\mathcal{N}$ is defined as:

$$H(\mathcal{N}) = \sum_{\mathcal{N}_i \in \mathcal{N}} C(\mathcal{N}_i) + C(N - R) - C(N) \tag{4}$$

where R is the set of workers that can cover all the sets in $\mathcal{N}$ (*i.e.*, $R = \bigcup_{\mathcal{N}_i \in \mathcal{N}} \mathcal{N}_i$).

To maintain the bandwidth constraints of the machine s, we only focus on the worker set $\mathcal{N}_s \subseteq N$ satisfying Eq. (5).

$$\sum_{n \in \mathcal{N}_s} \sum_{n' \in \mathcal{N}_{s'}} C^t_{n,n'} \leq P_{s,s'}, \forall s' \in S, t \in T \tag{5}$$

Searching the feasible worker sets. We call the worker sets satisfying Eq. (5) as *feasible worker sets* for machine s. The feasible worker sets of machine s can be constructed efficiently by performing a depth-first search [30], [31] on used machines. However, since the depth-first search will incur high time complexity (*i.e.*, $O(N!)$), we propose a polynomial time complexity algorithm to search the feasible worker set for machine s (Alg. 1) in this paper.

At the start of the algorithm, we initiate the feasible worker sets of machine s as $A(s) = \emptyset$, and the available worker set with the workers that are not placed yet (*i.e.*, $N_c = N - \mathcal{N}$. Then, we iteratively compute the communication overhead over the available worker set to check if they will exceed the bandwidth constraint of the corresponding machines. Specifically, for each available worker $n \in N_c$, we check if it performs communication at any stages of the collective communication operation to workers that have been placed. We stop searching if there exists one communication pair that exceeds the bandwidth constraint. If all communication pairs of worker n satisfy the bandwidth constraint, we will add it to the feasible worker set.

Given feasible worker sets, we can generate the placement scheme with a maximum number of remaining machines using

Algorithm 2 Submodular-based Algorithm

Step 1: Initialization
Initiate $\mathcal{N}_s, \forall s \in S$ by randomly distributing workers.
Initiate $\Phi \leftarrow \emptyset$.
Step 2: Iterative Merging Worker Subsets
while $|\Phi| \leq K - 1$ **do**
  Set $tmp \leftarrow 0, opt \leftarrow 0$
  for $s \in S_n$ **do**
    for $n \in \mathcal{N}_s - \Phi$ **do**
      $tmp \leftarrow H(\Phi \cup \{n\})$
      if $tmp > opt$ **then**
        $opt \leftarrow tmp$, $\mathcal{N}^* \leftarrow \mathcal{N}^* + \{n\}$
      end if
    end for
  end for
  $\Phi \leftarrow \Phi + \mathcal{N}^*$
  Update the feasible worker sets based on the bandwidth constraint of Eq. (5) with Alg. 1.
end while
Deploy the remaining workers on one machine (*i.e.*, $\Phi \leftarrow \Phi + \{N - \bigcup_{\mathcal{N} \in \Phi} \mathcal{N}\}$).

a submodular-based algorithm (Alg. 2). At last, we invoke the Alg. 2 twice over idle and non-idle machine sets to achieve two objectives. The algorithm is formally described in Alg. 3.

Minimizing the number of used machines. We show how to minimize the number of used machines for a given machine set. The algorithm consists of two steps. In the first step, we initiate feasible worker sets for each machine, and start with an empty set Φ. In the second step, we try to find a disjoint worker subset to maximize the number of remaining machines. To this end, we loop through the feasible worker set $\mathcal{N}_s$ for machine $s \in S_n$ and find the machine s with maximum function value (*i.e.*, $\arg\max_{s \in S_n} H(\Phi \cup \mathcal{N}_s)$). We add the worker set $\mathcal{N}_s$ with the maximum submodular function value into Φ. After that, we update the feasible worker sets for each machine based on the current worker deployment. The algorithm performs $K-1$ iterations so the set Φ will contain $K - 1$ worker sets. The remaining workers will be placed on one idle machine.

Generating the final scheme. We adopt Alg. 2 in different machine sets to satisfy two objectives of Eq. (1) in Alg. 3. The algorithm consists of three steps. If we only consider deploying workers on idle machines (*i.e.*, $s \in S_n$), the optimal deployment scheme will use a minimum number of machines. To this end, in the first step, we run Alg. 2 over the idle machine set S_n, and determine the set of used machines Φ_n. In the second step, we try to replace idle machines deployed with workers with the non-idle machine, to achieve the second objective (*i.e.*, minimize the number of used idle machines). We construct the machine set $S' = S_f \cup \Phi$. Then we run Alg. 2 over machine set S' to determine the deployed machine set Φ. In the third, we output the deployment decisions of workers according to the machine set Φ.

Algorithm 3 The Overall Algorithm

Step 1: Minimizing the Number of Deployed New Racks
Initiate $\mathcal{N}_s, \forall s \in S_n$ by randomly distributing workers.
Initiate $\Phi_n \leftarrow \emptyset$.
Calculate Φ_n on idle machine set S_n with Alg. 2.
Step 2: Minimizing the Total Number of Deployed Racks
Initiate $\mathcal{N}_s, \forall s \in S_f \cup \Phi_n$ by randomly distributing workers.
Initiate $\Phi \leftarrow \emptyset$.
Calculate Φ on machine set $S_f \cup \Phi_n$ with Alg. 2.
Step 3: Determining the Deployment of Workers
for $\mathcal{N}_s \in \Phi$ **do**
  Set $x_n^s = 1, \forall n \in \mathcal{N}_s, s \in S$.
end for

C. Performance Analysis

We give the definition of submodular as follows and prove that Eq. (4) is submodular.

Definition 2: (Submodular [32]): Given a finite set E, a real-valued function z on the set of subsets of E is called *submodular* if $z(S \cup \{e\}) - z(S) \leq z(S' \cup \{e\}) - z(S')$ for all $S' \subseteq S \subseteq E$ and $e \in E - S$.

Lemma 2: Given the set U as the power set of N, the function H defined in Eq. (4) is submodular on U.

Proof: Without loss of generality, we consider an arbitrary set $\Phi \subseteq U$ and an arbitrary set $M \subseteq N$. Suppose that M does not intersect with other sets in Φ (*i.e.*, $M \cap \Phi' = \emptyset, \forall \Phi' \in \Phi$). Then, we have

$$H(\Phi \cup M) - H(\Phi) = C(M) + C(N-M-R) - C(N-R) \quad (6)$$

Given an arbitrary subset $\Phi' \subseteq \Phi$, it also follows

$$H(\Phi' \cup M) - H(\Phi') = C(M) + C(N-M-R') - C(N-R') \quad (7)$$

Note that, $C(M) + C(N-M-R) - C(N-R)$ represents the reduction of deployed machine number by merging two subsets M and $N-M-R$ into set $N-R$. Since Φ' is the subset of Φ, $N-R$ is also the subset of $N-R'$ accordingly, we obtain $C(N-R) \leq C(N-R')$. Combining Eqs. (6) and (7), we know that

$$H(\Phi \cup M) - H(\Phi) \leq H(\Phi' \cup M) - H(\Phi') \quad (8)$$

According to Definition 2, we show that the set function H is submodular. ■

We give a well-known conclusion of the approximation ratio of the submodular function as a lemma to help the analysis.

Lemma 3: For a real-valued non-decreasing submodular function $z(S)$ on U, the optimization problem $\max_{S \subseteq U} \{z(S) : S \subseteq U\}$ can reach an approximation factor of $(1 - \frac{1}{e})$ if the algorithm performs greedily [32].

Now we analyze the approximation performance of our proposed algorithm based on the above lemmas.

Theorem 4: For the first objective, our proposed algorithm can achieve a $(1 - \frac{1}{e})$ approximation factor.

Proof: We first prove that the function H is non-decreasing. Supposing that there are two machines s_1 and s_2 and workers are split into two subsets $\mathcal{M}_1$ and $\mathcal{M}_2$, indicating that workers are deployed in machines s_1 or s_2 respectively. There are three conditions: 1) Workers are all deployed in machine s_1, then $C(\mathcal{M}_1) = 1, C(\mathcal{M}_2) = 0$; 2) Workers are all deployed in machine s_2, then $C(\mathcal{M}_1) = 0, C(\mathcal{M}_2) = 1$; 3) Workers are deployed in machines s_1 and s_2, we have $C(\mathcal{M}_1) = 1, C(\mathcal{M}_2) = 1$. If we merge two worker subsets into one worker set (*i.e.*, deploying all workers in one machine), we have $C(\mathcal{M}_1 + \mathcal{M}_2) = 1$. According to Eq. (6), by applying $\mathcal{M}_1 = M$ and $\mathcal{M}_2 = N - M - R$, we have

$$H(\Phi \cup M) - H(\Phi) \geq 0 \quad (9)$$

Thus, the function H is non-decreasing. In Lemma 2, we have proved that the function H is submodular. Given the fact that H is non-decreasing and submodular, according to Lemma 3, our proposed algorithm can reach a $(1-1/e)$ approximation factor for the problem of Eq. (3). ■

Theorem 5: For the second objective, our proposed algorithm can achieve a $(1 - \frac{1}{e})$ approximation factor.

Proof: In the second step of Alg. 3, we construct the machine set with non-idle machines and part of idle machines (*i.e.*, Φ_n). In Theorem 4, we have proved that function H is a real-valued non-decreasing submodular function. According to Lemma 2, we can find a machine set with a $(1 - 1/e)$ approximation factor for the second objective of Eq. (1). ■

Nemhauser [33] proved that any algorithm evaluating the submodular function at a polynomial number of sets will not obtain an approximation guarantee better than $(1-1/e)$, unless $NP = P$. Thus, we obtain the following Theorem.

Theorem 6: The problem in Eq. (3) does not admit a polynomial-time algorithm with approximation ratio $1-1/e+\epsilon$ unless $NP = P$, where ϵ is an arbitrary positive constant.

Theorem 7: The time complexity of Alg. 3 is $O(2 \cdot |S|^2 \cdot |N| \cdot T)$.

Proof: The main time complexity is contributed by the second step of Alg. 2. We run $K = |S|$ iterations to merge the worker subsets. In each iteration, we need to update the feasible worker set using Alg. 1. We should note that the number of feasible sets for each machine may be exponential. However, the work [31] has shown that a polynomial number of feasible sets is enough for performance optimization. To achieve the trade-off optimization between algorithm complexity and network performance, we only construct the polynomial number of feasible workers for each machine with Alg. 1. In the worst case, we iterate all workers at one time and check all stages of collective communication algorithm. So the time complexity of Alg. 1 is $O(|N|T)$. Under this condition, the function H is calculated $O(K|N|T)$ times in each iteration. As a result, the time complexity of Alg. 2 reaches $O(|S|^2 \cdot |N| \cdot T)$. Since we run Alg. 2 twice, the final time complexity of Alg. 3 is $O(2 \cdot |S|^2 \cdot |N| \cdot T)$. ■

V. Performance Evaluation

In this section, we evaluate Titan with testbed and simulation experiments and highlight our findings as follows:

- By utilizing the non-idle machines, Titan reduces the number of used machines by up to 47.9% (Exp#1) and the machine fragmentation rate by 38.1% (Exp#2), compared with benchmarks.
- By considering collective communication pattern, Titan reduces the communication overhead by 40.6%-76.4% (Exp#3) and increase the average profit rate by 22.7%-$12\times$ (Exp#4), compared with the existing solutions.
- By performing utilization-aware worker placement, Titan can reduce the machine hour by 29.7% compared with other alternatives (Exp#5) .
- Titan can achieve the near-optimal training performance (Exp#6) and communication time (Exp#7) compared with state-of-the-art solutions.

A. Experimental Setup

Metrics. We adopt the following metrics for performance comparison: (1) number of used machines; (2) total machine hour; (3) average machine fragmentation rate; (4) average profit rate (5) training throughput; (6) time-to-accuracy.

In the simulation, we calculate the number of machines deployed by at least one worker as *number of used machines*. We measure the number of launched machines multiplied by training time as the *total machine hour*. The shorter the total machine hour, the better. We define the *machine fragmentation rate* of as the ratio of the number of idle GPUs to the number of total GPUs in a machine and calculate the *average machine fragmentation rate* with the average value of machine fragmentation rate of machines at a given time. At last, we define the *average profit rate* as the $\frac{K \cdot P}{N \cdot T}$ at a given time, where K, P, N and T denote the number of jobs, price of the job, number of used machines and duration of jobs, separately. In practice, tenants pay cloud providers for the DT jobs, and the price of a job is usually related to its complexity (*e.g.*, number of workers, training duration).

In terms of the training performance, we measure the average number of processed samples (*e.g.*, images) per second as *training throughput* and record the time and test accuracy of each epoch as *time-to-accuracy* in the testbed.

Benchmarks. We compare Titan with three benchmarks. The first benchmark is Elasticflow [11], which performs best-fit placement scheme. Specifically, Elasticflow will search servers in which the number of idle GPUs is closest to the number of workers of the DT job. This scheme will consolidate the job's GPUs so that the job is allocated with the highest possible bandwidth between its workers. However, it will increase the machine fragmentation rate [11]. The second one, called HiveD [21], tries to utilize fragment GPUs in the cluster first, so the workers of a job will distributed among multiple servers. It reduces the machine fragmentation rate of the cluster, however, may introduce a large amount of communication overheads. The third solution is Tiresias [10]. The goal of Tiresias is to minimize the total network traffic and balance the network load across machines in the cluster. Specifically, Tiresias profiles the popular models and identifies whether their training performance is sensitive to consolidation. As a result, it performs consolidation placement or distributes workers to multiple servers according to model sensitivity.

B. Simulation Settings and Results

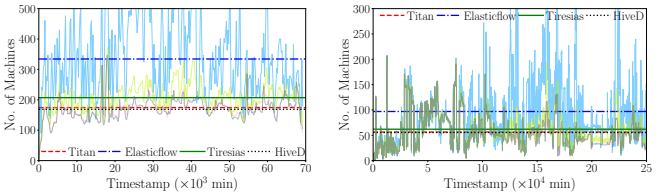
Settings. Our simulations are implemented on a physical server equipped with an Intel Core i9-10900 processor and 64GB RAM. We simulate a cluster with fat-tree topology [16], which is widely used in modern datacenter networks. Specifically, the topology contains 64 racks, where each rack contains 8 machines. In total, there are 256 edge switches, 256 aggregation switches and 64 core switches. All components are connected with 100Gbps links. We assume each machine is equipped with 8 GPUs. After placing workers in machines, we simulate the cross-machine traffic based on the communication patterns of the HD allreduce algorithm.

Workloads. We adopt two real-world DT job arrival traces in the simulation. The first is a two-month trace [11] containing 69742 jobs collected from 10 Microsoft clusters. The cluster size ranges from 164 GPUs to 2783 GPUs. The second is a six-month trace [34] containing 880740 jobs collected from Shanghai AI lab, where 470497 jobs are deployed in GPUs. This trace is a mixure of short-term and long-term jobs. Each record has information about job id, submission time, number of GPUs, and duration. Since Tiresias needs the model type for worker placement, similar to work [11] we randomly choose a DNN model for each job record. Although some traces [35] contain the model type of jobs, they are limited in the scale and number of jobs. Besides, considering that the arrival jobs is unpredictable in practice, we eventually choose the above traces and randomly assign a model type for each job.

Simulation workflow. At the start, we let all machines in the cluster be idle. Then, for each arrived job, we will decide the GPU allocation for worker placement and update the simulated timestamp. We maintain a list of current jobs and check if there are jobs finished at the latest timestamp. If there are, we will release its occupied GPUs for the incoming jobs. Since different placement schemes may influence the communication time and job duration, we calculate the communication time of each job according to placement results and add the bias to the corresponding job durations. Specifically, we obtain the initial job duration from the trace and add the communication time of different algorithms to it.

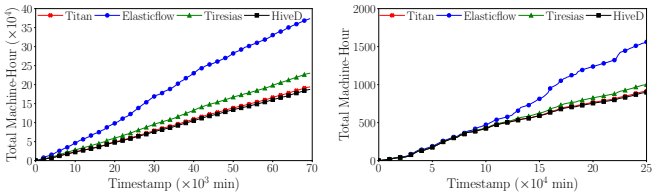
We iteratively calculate the metrics, such as the number of used machines, and the results are shown in Figs. 2-7. For ease of reading, we add dotted horizontal lines in some figures to show the average value of the corresponding metric.

(Exp#1) Comparison on number of used machines. We consider machines placed with at least one worker as used machines. We monitor the number of used machines as the job arrives, and the results are shown in Figs. 2-3. Fig. 2 shows that Titan utilizes a smaller number of machines compared with



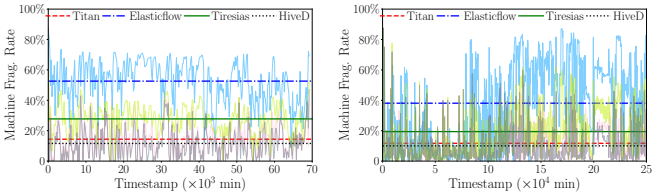
(a) Microsoft Trace (b) Shanghai AI Lab Trace

Fig. 2: No. of Used Machines vs. Timestamp



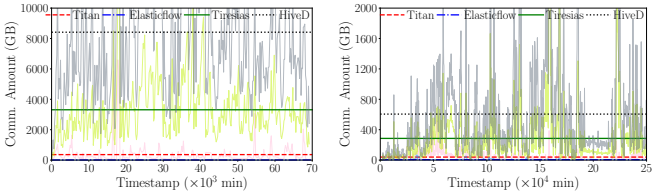
(a) Microsoft Trace (b) Shanghai AI Lab Trace

Fig. 3: Machine Hour vs. Timestamp



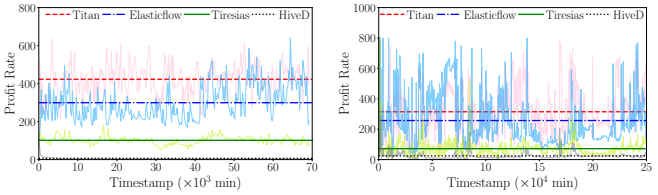
(a) Microsoft Trace (b) Shanghai AI Lab Trace

Fig. 4: Machine Fragmentation Rate vs. Timestamp



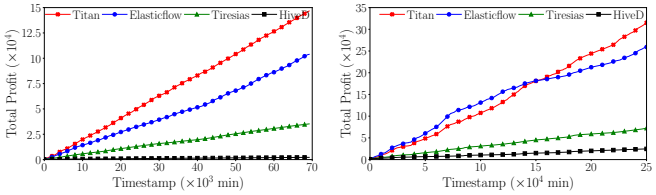
(a) Microsoft Trace (b) Shanghai AI Lab Trace

Fig. 5: Communication Overhead vs. Timestamp



(a) Microsoft Trace (b) Shanghai AI Lab Trace

Fig. 6: Profit Rate vs. Timestamp



(a) Microsoft Trace (b) Shanghai AI Lab Trace

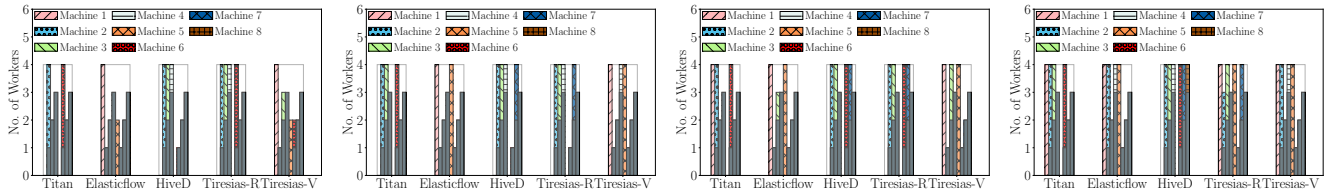
Fig. 7: Total Profit vs. Timestamp

Elasticflow and Tiresias and a similar number of machines with HiveD. In Fig. 3, we further record the machine hour in the cluster, and Titan costs the least machine hour. For example, in Fig. 2(a), the average number of used machines of Titan, HiveD, Elasticflow, and Tiresias are 174, 168, 334, and 207, respectively. Titan, HiveD, Elasticflow, and Tiresias are 174, 168, 334, and 207, respectively. Titan reduces the number of used machines by 47.9% and 15.9%, compared with Elasticflow and Tiresias, respectively. Compared with HiveD, Titan uses 4% more machines, since HiveD prefers split and place workers on fragmented GPUs.

(Exp#2) Comparison on machine fragmentation rate. This set of evaluations investigates the resource utilization of the GPU cluster. Specifically, for machines placed more than one worker, we calculate the machine fragmentation rate as $\frac{\text{No. of occupied GPUs}}{\text{No. of equipped GPUs}}$. We calculate the average machine fragmentation rate of all used machines at a given timestamp and plot the results in Fig. 4. From Fig. 4, we can see that, Elasticflow always achieves the highest machine fragmentation rate, and Titan keeps a relatively low machine fragmentation rate. For instance, in Fig. 4(b), the average machine fragmentation rate of Titan, HiveD, Elasticflow and Tiresias is 11.7%, 10%, 38.1% and 19.4%, respectively. The reason is that Titan will try to place workers in machines with fragmented GPUs rather than idle machines, therefore keeping the machine fragmentation rate low.

(Exp#3) Comparison on communication overhead. In this set of experiments, we measure the total communication overhead of workers at the given time of four benchmarks. Specifically, we assume workers of a DT job perform allreduce via the HD algorithm. Since the communication pattern is known, we can obtain the set of worker communication pairs. Considering that, in practice, GPUs inside a machine are connected with sufficient bandwidth, we mainly focus on cross-machine communication here, and the communication overhead is shown in Fig. 5. We can see that, Elasticflow keeps the least communication overhead and Titan maintains the second least overhead. In Fig. 5(a), the communication overhead of Titan, Tiresias, and HiveD are 1.65GB, 2.78GB, and 7.03GB, respectively. The results show that Titan can reduce the communication overhead by 40.6% and 76.4%, compared with Tiresias and HiveD, respectively. The reason is that, when placing the workers, Titan considers the communication pattern among workers, and makes sure the communication amount will not exceed the bandwidth constraint.

(Exp#4) Comparison on profit rate. This set of experiments is conducted to illustrate the profit rate enhancement of Titan. Once a job arrives, we decide its worker placement and calculate the profit rate at this timestamp. We also accumulate



(a) 6 Workers (b) 8 Workers (c) 10 Workers (d) 12 Workers

Fig. 8: Worker Placement Results of All Solutions

the profit rate of all timestamps as the total profit. The results are shown in Figs. 6-7. It shows that Titan achieves the highest average profit rate among other alternatives. For example, in Fig. 6(b), Titan achieves a profit rate of 314, while that of Elasticflow, Tiresias and HiveD are 256, 71 and 24, respectively. Titan improves the profit rate by 22.7%, 3.42× and 12×, compared with Elasticflow, Tiresias and HiveD, respectively. In terms of the total profit, when simulating Trace 1, Titan can improve the total profit by 41.21%, 316%, and 65×, compared with other alternatives. The reason is that, Titan uses as little number of idle machines as possible, and reduces the communication overhead by considering the allreduce communication constraint. Therefore, Titan reduces the value of N and T, to improve the average profit rate.

Discussion about the impact of characteristics of job traces. In general, the effectiveness of job placement algorithms will be decided by the characteristics of job traces. Specifically, if jobs arrive at the cluster in a relatively stable pattern (similar job durations and arrival intervals), the machine fragmentation rate will remain stable. And different algorithms only affect the value of the machine fragmentation rate. If the job trace contains a mixture of short-term and long-term jobs, consolidation placement algorithms such as Elasticflow may cause machine fragmentation rates to fluctuate severely. By considering the communication pattern of workers, Titan can utilize non-idle machines to reduce the machine fragmentation rate when placing workers of jobs.

Summary. By trying to utilize the non-idle machines and carefully consider the collective communication overhead, Titan reduces the number of used idle machines, the machine fragmentation rate and communication overhead, as a result, improves the total profit of the cluster.

C. Testbed Settings and Results

Settings. We build the testbed with 8 servers and 1 switch. Each server has one NVIDIA GeForce RTX 3090, a 22-core Intel Xeon 6152 processor, and a Mellanox ConnectX-6 100G dual-port NIC. All the servers run Ubuntu 18.04 with CUDA 11.6. The NIC driver of all servers is Mellanox driver OFED 5.51.0.3.2. These servers and the switch are connected by 100 Gbps links. Due to the limitation of our machines, we use docker to simulate multiple GPUs in one server, similar to [37]. Specifically, we divide one physical server into 4 docker containers, each with 1 GPU. These docker containers share one physical GPU in reality. Ultimately, we get a testbed with 32 devices (virtualized GPUs) across 8 machines.

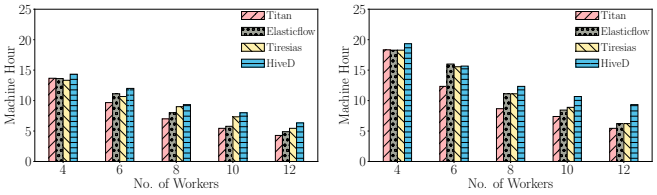
To simulate the resource fragmentation of machines, we run several background training jobs on these machines. Specifically, The number of background jobs ranges from 0-3, where one background job trains ResNet-50 [38] with one device.

Workloads. We use two image classification models and the Cifar-100 dataset [39] in our experiments. The models include: ResNet-50 [38] and VGG-16 [40]. Referring to the settings in [41], during the experiments, we set the learning rate to 0.1 for ResNet-50 and 0.01 for VGG-16. The Cifar-100 dataset contains 60000 images (50000 for training and 10000 for testing), labeled in 100 classes. By default, we set the batch size to 512 and perform 200 training epochs for each DT job. We implement HD allreduce algorithm based on gloo [42], and invoke the corresponding backend in `pytorch.distributed`.

Worker placement result. In the testbed, we calculate the worker placement solution of the DT job with different models and scales (*i.e.*, number of workers), then specify the map of logical workers to physical devices and launch the DT job. We illustrate the detailed placement results of DT jobs with 4 workers as follows and present the result of placing 6-12 workers in Fig. 8. We use Tiresias-R to denote the placement result of Tiresias for ResNet-50 and Tiresias-V to denote that of Tiresias for VGG-16. The gray rectangles in Fig. 8 represent the background training jobs, and colored rectangles represent workers placed in the corresponding machines. We can see that, as the number of workers increases, Titan can remain one idle machine even there places 12 workers.

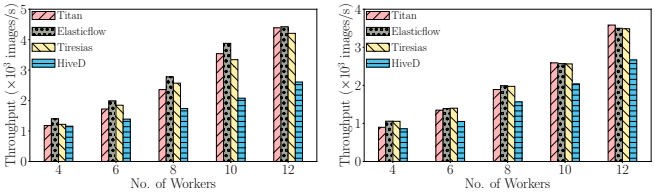
(Exp#5) Comparison on total machine hour. In this set of experiments, we calculate the total machine launching time in the testbed and record the machine hour in Fig. 9. We can see that Titan can always achieve the least machine hour. For example, in Fig. 9(b), when the number of workers is 8, the total machine hour of Titan, Elasticflow, Tiresias and HiveD is 8.67, 11.11, 11.12, and 12.33, respectively. Titan reduces the total machine hour by 21.9%, 22% and 29.7%, compared with other benchmarks. The reason is that we try not to launch idle machines, and reduce the allreduce communication overhead to guarantee the training performance over non-idle machines.

(Exp#6) Comparison on training throughput. In this set of experiments, we modify the number of workers and evaluate the training throughput performance. The evaluation results are



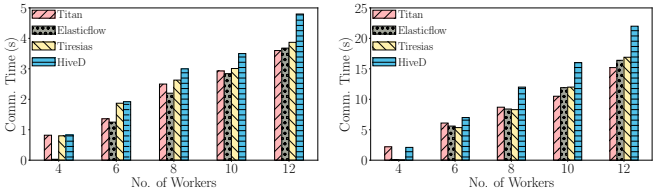
(a) ResNet-50 (b) VGG-16

Fig. 9: Machine Hour vs. No. of Workers



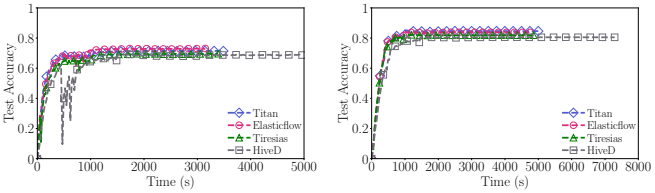
(a) ResNet-50 (b) VGG-16

Fig. 10: Training Throughput vs. No. of Workers



(a) ResNet-50 (b) VGG-16

Fig. 11: Communication Time vs. No. of Workers



(a) ResNet-50 (b) VGG-16

Fig. 12: Test Accuracy vs. Training Time

shown in Fig. 10. We increase the number of workers from 4 to 12 with an increment of 2. The experimental results show that Elasticflow always achieves the highest training throughput as the number of workers increases and Titan can achieve a similar performance compared with Elasticflow. For example, given 12 workers in Fig. 10(a), the training throughputs of Elasticflow, Titan, Tiresias, and HiveD are 4422, 4389, 4208, and 2610 images/s, respectively. Titan can achieve the performance of Elasticflow by 99.3%. The reason is that Elasticflow tries to place workers in as few idle servers as possible to minimize cross-server communication. Titan firstly try to use as few number of idle servers to save the cost, then use as few number of servers to reduce cross-machine communication. Moreover, Titan carefully considers the worker placement according to the allreduce communication pattern to further reduce communication overhead. Therefore Titan can achieve similar performance with benchmarks.

(Exp#7) Comparison on communication time. This set of evaluations varies the number of workers and compares the per-epoch communication time performance of different solutions. Note that, each epoch consists of three phases: local training phase, allreduce communication phase, and parameter synchronization phase. Since different worker placement strategies only influence the allreduce communication phase, we only record the communication time as shown in Fig. 11. Fig. 11 shows that, as the number of workers increases, the communication time increases too. Under the fixed number of workers, Elasticflow obtains the least per-epoch communication time among all solutions and Titan can achieve a similar performance with Elasticflow. Given 8 workers in Fig. 11(b), the per-epoch communication time of Elasticflow, Titan, Tiresias and HiveD are 8.4s, 8.7s, 8.3s and 12s, respectively. Note that, for VGG-16, Tiresias performs the same worker placement strategy with Elasticflow (*i.e.*, use 2 idle servers to place 8 workers), therefore achieving similar performance. Titan only increases the communication time by 3.5% and 4.8% compared with Elasticflow and Tiresias, respectively.

In Fig. 12, we further record the test accuracy versus training time for the DT jobs containing 8 workers. For example, given ResNet-50 in Fig. 12(a), Titan reaches an accuracy of 0.7140 in 3486s while the time of Elasticflow, Tiresias, and HiveD are 3147s, 3390s, and 4964s, respectively. It shows that Titan achieves a similar test accuracy compared with other alternatives.

Summary. By minimizing the number of used idle machines when placing workers, Titan spends the least machine hours while achieving near-optimal training performance. Compared with fragmentation-aware worker placement solution, Titan can reduce the communication overhead by considering the allreduce communication constraint.

VI. Conclusion

In this paper, we present Titan, a novel non-idle machine-aware worker placement scheme to improve the resource utilization rate of GPU clusters. We formulate a multi-objectives non-linear problem and propose a submodular-based algorithm to solve it. Extensive large-scale simulation and small-scale testbed experiment results show that Titan can achieve near-optimal distributed training performance while improving the cluster profit rate.

Acknowledgement

We thank the anonymous reviewers and our shepherd Prof. Xiaoqi Chen for their helpful suggestions. This research received partial support from several funding sources, including the National Science Foundation of China (NSFC) under Grants 62372426 and 62102392; the Fundamental Research Funds for the Central Universities; and the Youth Innovation Promotion Association of the Chinese Academy of Science (2023481). The corresponding authors of this paper are Gongming Zhao and Hongli Xu.

REFERENCES

[1] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[2] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed, "Automatic text summarization: A comprehensive survey," *Expert systems with applications*, vol. 165, p. 113679, 2021.

[3] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, "Lamda: Language models for dialog applications," *arXiv preprint arXiv:2201.08239*, 2022.

[4] M. Langer, Z. He, W. Rahayu, and Y. Xue, "Distributed training of deep learning models: A taxonomic perspective," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2802–2818, 2020.

[5] S. Li and T. Hoefler, "Near-optimal sparse allreduce for distributed deep learning," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 135–149.

[6] Z. Zhang, C. Wu, and Z. Li, "Near-optimal topology-adaptive parameter synchronization in distributed dnn training," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[7] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong *et al.*, "Megascale: Scaling large language model training to more than 10,000 gpus," *arXiv*, 2024.

[8] 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 *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 945–960.

[9] 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 *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021.

[10] 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.

[11] D. Gu, Y. Zhao, Y. Zhong, Y. Xiong, Z. Han, P. Cheng, F. Yang, G. Huang, X. Jin, and X. Liu, "Elasticflow: An elastic serverless training platform for distributed deep learning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 266–280.

[12] Y. Zhao, C. Yang, G. Zhao, Y. Hou, T. Wang, and C. Qiao, "Jointps: Joint parameter server placement and flow scheduling for machine learning clusters," *IEEE Transactions on Computers*, 2023.

[13] Z. Luo, Y. Bao, and C. Wu, "Optimizing task placement and online scheduling for distributed gnn training acceleration," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 890–899.

[14] J. Dong, Z. Cao, T. Zhang, J. Ye, S. Wang, F. Feng, L. Zhao, X. Liu, L. Song, L. Peng *et al.*, "Eflops: Algorithm and system co-design for a high performance distributed training platform," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 610–622.

[15] "Maximize system throughput with nvidia nvlink," https://www.nvidia.com/en-us/data-center/nvlink/.

[16] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, pp. 63–74, 2008.

[17] Q. Weng, L. Yang, Y. Yu, W. Wang, X. Tang, G. Yang, and L. Zhang, "Beware of fragmentation: Scheduling {GPU-Sharing} workloads with fragmentation gradient descent," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 995–1008.

[18] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond {GPUs} for {DNN} scheduling on {Multi-Tenant} clusters," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 579–596.

[19] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. ACM, Nov. 2021. [Online]. Available: http://dx.doi.org/10.1145/3458817.3476223

[20] J. Li, H. Xu, Y. Zhu, Z. Liu, C. Guo, and C. Wang, "Lyra: Elastic scheduling for deep learning clusters," in *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023, pp. 835–850.

[21] H. Zhao, Z. Han, Z. Yang, Q. Zhang, F. Yang, L. Zhou, M. Yang, F. C. Lau, Y. Wang, Y. Xiong *et al.*, "{HiveD}: Sharing a {GPU} cluster for deep learning with guarantees," in *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, 2020, pp. 515–532.

[22] C. Hwang, T. Kim, S. Kim, J. Shin, and K. Park, "Elastic resource sharing for distributed deep learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 721–739.

[23] Z. Cheng and Z. Xu, "Bandwidth reduction using importance weighted pruning on ring allreduce," 2019. [Online]. Available: https://arxiv.org/abs/1901.01544

[24] "Nvidia collective communications library (nccl)," https://developer.nvidia.com/nccl.

[25] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[26] J. Dong, S. Wang, F. Feng, Z. Cao, H. Pan, L. Tang, P. Li, H. Li, Q. Ran, Y. Guo, S. Gao, X. Long, J. Zhang, Y. Li, Z. Xia, L. Song, Y. Zhang, P. Pan, G. Wang, and X. Jiang, "Accl: Architecting highly scalable distributed training systems with highly efficient collective communication library," *IEEE Micro*, vol. 41, no. 5, pp. 85–92, 2021.

[27] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 713–728, 2005.

[28] B. Bixby, "The gurobi optimizer," *Transp. Re-search Part B*, vol. 41, no. 2, pp. 159–178, 2007.

[29] A. Agarwal, Z. Liu, and S. Seshan, "{HeteroSketch}: Coordinating network-wide monitoring in heterogeneous and dynamic networks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 719–741.

[30] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[31] L. Luo, G. Zhao, H. Xu, L. Xie, and Y. Xiong, "Vita: Virtual network topology-aware southbound message delivery in clouds," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 630–639.

[32] A. Krause and D. Golovin, "Submodular function maximization." *Tractability*, vol. 3, pp. 71–104, 2014.

[33] G. L. Nemhauser and L. A. Wolsey, "Best algorithms for approximating the maximum of a submodular set function," *Mathematics of operations research*, vol. 3, no. 3, pp. 177–188, 1978.

[34] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo *et al.*, "Characterization of large language model development in the datacenter," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 709–729.

[35] J. Cao, Y. Guan, K. Qian, J. Gao, W. Xiao, J. Dong, B. Fu, D. Cai, and E. Zhai, "Crux: Gpu-efficient communication scheduling for deep learning training," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 1–15.

[36] "Mininet: An instant virtual network on your laptop (or other pc)," https://mininet.org/.

[37] H. Wang, H. Tian, J. Chen, X. Wan, J. Xia, G. Zeng, W. Bai, J. Jiang, Y. Wang, and K. Chen, "Towards {Domain-Specific} network transport for distributed {DNN} training," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1421–1443.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[39] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-100 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[41] J. Fang, G. Zhao, H. Xu, C. Wu, and Z. Yu, "Grid: Gradient routing with in-network aggregation for distributed training," *IEEE/ACM Transactions on Networking*, 2023.

[42] "Index of algorithms provided by gloo and their semantics.)," https://github.com/facebookincubator/gloo/blob/main/docs/algorithms.md.