spotDNN: Provisioning Spot Instances for Predictable Distributed DNN Training in the Cloud

***Abstract*—Distributed Deep Neural Network (DDNN) training on cloud spot instances is increasingly compelling as it can significantly save the cloud user budget. To cope with unexpected instance revocations, provisioning a *heterogeneous* cluster using the asynchronous parallel mechanism becomes the *dominant* method for DDNN training with spot instances. However, blindly provisioning a cluster of spot instances can easily lead to *unpredictable* DDNN training performance, mainly because bottlenecks occur on the parameter server network bandwidth and PCIe bandwidth resources. Moreover, while existing works mostly focus on modeling DDNN training speed, relatively little attention has been paid to *quantitatively* characterizing the DDNN training loss of heterogeneous clusters. In this paper, we propose *spotDNN*, a heterogeneity-aware spot instance provisioning framework to provide predictable performance for DDNN training workloads in the cloud. It first builds an *analytical* performance model of DDNN training in a heterogeneous cluster, by explicitly considering the contention on the bottleneck resources. With such a model and lightweight workload profiling, we further design a *cost-efficient* instance provisioning strategy to guarantee the training performance service level objectives (SLOs). We implement a prototype of *spotDNN* and conduct extensive experiments on Amazon EC2. Experiment results show that *spotDNN* can deliver predictable DDNN training performance while reducing the monetary cost by up to 68.1% in comparison to the state-of-the-art instance provisioning strategies, yet with practically acceptable runtime overhead.**

***Index Terms*—distributed DNN training, predictable performance, spot instance provisioning, heterogeneous clusters**

I. Introduction

As Deep Neural Network (DNN) models get deeper and the training datasets get larger, distributed DNN (DDNN) training in the cloud becomes increasingly compelling [1]. To improve resource utilization, cloud providers offer users with idle computing resources at a 60%-80% discount, such as AWS spot instances, Google Spot VMs, and Azure Spot VMs [2]. Though at highly reduced prices, deploying DDNN training workloads on spot resources can suffer from severe performance degradation [3], which is mainly caused by unexpected instance revocations [4] and insufficient spot capacity due to user quotas [5]. To alleviate such performance degradation, provisioning a *heterogeneous* cluster of spot instances using the asynchronous parallel (ASP) mechanism [6] is becoming the *first choice* for cloud users to train large DNN models.

However, how to provision the heterogeneous cluster cost-efficiently (*i.e.,* finding the optimal provisioning plan including the type and the number of spot instances) still remains challenging, even for sophisticated cloud users. They often rely on their own experience and intuition to provision spot instances for DDNN training [7]. Unfortunately, blindly provisioning

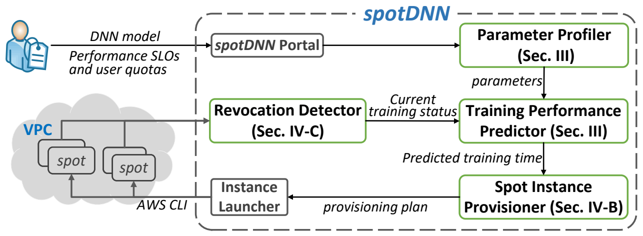
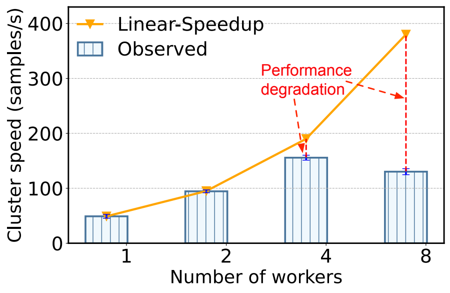

spotDNN
DNN model
Performance SLOs and user quotas
spotDNN Portal
Parameter Profiler (Sec. III)
parameters
VPC
spot
spot
Revocation Detector (Sec. IV-C)
Current training status
Training Performance Predictor (Sec. III)
Predicted training time
AWS CLI
Instance Launcher
provisioning plan
Spot Instance Provisioner (Sec. IV-B)

Fig. 1: Overview of *spotDNN*.

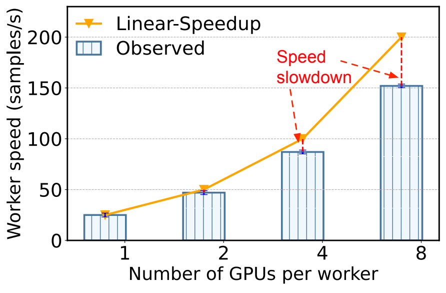
heterogeneous instances suffers from *unpredictable* DDNN training performance due to the following two facts. *First,* the network bandwidth of the parameter server (PS) [8] and the limited PCIe bandwidth inside the instance can easily become bottleneck resources. As evidenced by our motivation experiment in Sec. II-B, the DDNN training time of a VGG-19 model can be prolonged by up to 65.7% as more spot instances are provisioned. *Second,* the scale of heterogeneous clusters (with the ASP mechanism) can negatively impact the convergence rate of DDNN training workloads due to the gradient staleness. Our motivation experiment in Sec. II-B also reveals that the convergence rate of a VGG-19 model can be slowed down by up to 34.3% as more workers are provisioned. Accordingly, how to adequately provision spot instances to alleviate such unpredictable performance becomes the main *obstacle* to training DNN models in the cloud.

To tackle such performance issues above, many efforts have been devoted to scheduling training jobs (*e.g.,* Gavel [9]), tuning the batch size (*e.g.,* LB-BSP [10]), and optimizing the parameter synchronization mechanism (*e.g.,* Spotnik [11]) in a *homogeneous* DDNN training cluster. However, relatively little attention has been paid to how to adequately provision spot instances and guarantee the DDNN training performance in a *heterogeneous* cluster. There have recently been works (*e.g.,* CM-DARE [7]) on modeling the training performance of a heterogeneous cluster. Nevertheless, they imprecisely predict the training performance without considering the performance degradation caused by the bottleneck resources. Though a more recent work (*i.e.,* Srifty [12]) leverages exhaustive search to identify the optimal instance provisioning plan in a heterogeneous cluster, it is hard to be implemented in practice due to the heavy computation overhead. As a result, scant research has been devoted to providing predictable performance by appropriately provisioning a heterogeneous cluster of spot instances in a lightweight manner.

In this paper, we present *spotDNN* in Fig. 1, a heterogeneity-aware spot instance provisioning framework that guarantees



(a)



(b)

Fig. 2: Training speed of a VGG-19 model on ImageNet obtained by (a) an n-worker cluster which consists of $\frac{n}{2}$ g3.4xlarge instances and $\frac{n}{2}$ g4dn.4xlarge instances, and (b) a G4dn instance by varying the number of GPUs from 1 to 8.

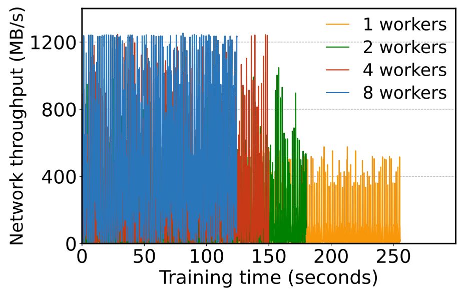


Fig. 3: Network throughput of the PS node during the training period of the VGG-19 model on ImageNet in the n-worker cluster.

DDNN training performance service level objectives (SLOs) in terms of training time and training loss, while minimizing the training monetary cost. By leveraging the parameters obtained through a lightweight workload profiling by the *parameter profiler*, we *first* devise an analytical performance model of DDNN training workloads in a heterogeneous cluster. The *training performance predictor* in our model explicitly considers the performance degradation caused by *bottleneck resources* including the PS network bandwidth and limited PCIe bandwidth. It also leverages the *normalized batch size* and the number of heterogeneous workers to characterize the training loss in a heterogeneous cluster. *Second,* we design a cost-efficient instance provisioning strategy in *spot instance provisioner* to guarantee the training performance SLOs while saving the user training budget. By periodically checking the status of DDNN training and spot instances in the *revocation detector*, *spotDNN* is able to consider the performance impact of unexpected instance revocations, and provisions *takeover* spot instances to guarantee the performance SLOs of DDNN training workloads.

Finally, we implement a prototype[1] of *spotDNN* on AWS EC2 [5] and conduct extensive prototype experiments using four representative DNN models and seven types of spot instances. Experimental results demonstrate that *spotDNN* can deliver predictable performance for DDNN training workloads while saving the user budget by up to 68.1% compared to the state-of-the-art instance provisioning strategies (*e.g.,* Srifty [12]), yet with practically acceptable runtime overhead.

The rest of the paper is organized as follows. Sec. II conducts an empirical study to analyze the key factors that affect DDNN training performance in heterogeneous environments, which motivates the design of our analytical performance model of DDNN training workloads in Sec. III. Sec. IV presents the design and implementation of our *spotDNN* instance provisioning strategy. Sec. V evaluates the effectiveness and runtime overhead of *spotDNN*. Sec. VI discusses related work and Sec.VII concludes this paper.

II. Background and Motivation

In this section, we first seek to explore the key factors that impact the DDNN training performance in a heterogeneous cluster. We then present a motivational example to show how to adequately provision spot instances to save user budget while guaranteeing the training performance SLOs.

A. DDNN Training with Cloud Spot Instances

Though DDNN training with spot instances can significantly save the user budget, it is challenging because spot instances can be revoked at any time and the number of spot requests has a stringent limit (*i.e.,* quota) [5]. To cope with such challenges above, we simply adopt a *heterogeneous* cluster of spot instances for DDNN training due to the following two facts. *First,* provisioning spot instances with one type is unlikely to meet workload performance requirements due to the user quotas. *Second,* heterogeneous instances can take over the training work of revoked instances as different instance types are usually not revoked simultaneously [4]. To efficiently train DNN models on heterogeneous spot instances, we focus on the ASP [13] mechanism under the PS architecture [8]. We have the following three benefits. *First,* ASP allows each worker[2] to communicate with the PS individually, and thus it breaks the synchronization barrier in the heterogeneous environment. *Second,* the training process cannot be interrupted in ASP even when several workers are revoked [7]. *Third,* the newly-launched *takeover* instances can pull the latest model parameters directly from the PS, which reduces the recovery time of instance revocations.

B. Characterizing DDNN Training Performance in Heterogeneous Clusters

To explore the key factors of DDNN training performance in a heterogeneous cluster, we conduct motivation experiments on EC2 spot instances [5], by adopting G3 instances and G4dn instances for workers. We also deploy an m5.xlarge on-demand instance for the PS. We run three representative DDNN training workloads including ResNet-50 and ResNet-110 [14] on the CIFAR-100 [15] dataset, as well as VGG-19 [16] on a part of the ImageNet [17] dataset. To accelerate DDNN training, we simply adopt the batch size which can fully utilize the GPU resource for each GPU type. We also enlarge the learning rate equally as the batch size to ensure the training quality of DNN models [10].

[1]https://github.com/spotDNN/spotDNN

[2]We use *workers* and *instances* interchangeably in this paper.

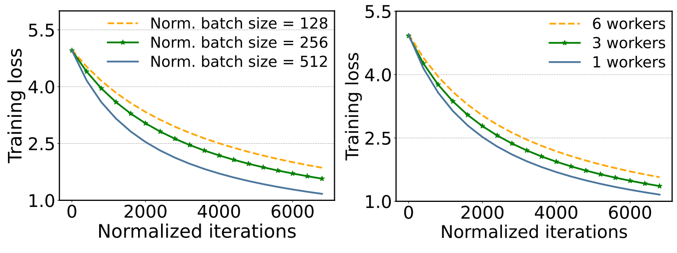


Fig. 4: Training loss of a ResNet-50 model on CIFAR-100 achieved by (a) different *normalized* batch sizes on heterogeneous clusters consisting of 6 workers, and (b) varying the scale of the heterogeneous cluster as 1, 3, and 6 workers with the *normalized* batch size set as 256.

Training Speed. As shown in Fig. 2(a), the observed cluster speed first increases and then surprisingly decreases by up to 65.7% as the number of workers increases from 2 to 8 compared with the linear speedup curve. This is because the PS network bandwidth can easily become a *bottleneck resource* as the number of workers increases when training even with the ASP mechanism. To validate our analysis above, we record the PS network throughput over time by varying the number of workers shown in Fig. 3. As expected, the PS network bandwidth becomes saturated (*i.e.,* 1,200 MB/s) as the number of workers grows to around 8. Meanwhile, the PS CPU utilization keeps at a low level (*i.e.,* below 30%). Such an observation excludes the impact of CPU resource and confirms our analysis of PS network bandwidth as a bottleneck resource when training with the ASP mechanism.

Moreover, the limited PCIe bandwidth within a worker is another key factor that affects the training speed. As shown in Fig. 2(b), the worker training speed slows down by up to 25.6% compared to the linear speedup curve, as the number of GPUs inside a worker varies from 2 to 8. Our result is consistent with the latest measurement study [18]. This is because multiple GPUs in a worker contend for the limited PCIe bandwidth resource, prolonging the gradient aggregation time (linearly) with the number of GPUs. Accordingly, PCIe bandwidth contention can cause a moderate speed slowdown, resulting in a *non-linear* acceleration of the training speed for a multiple-GPU instance.

Training Loss. To characterize DDNN training in a heterogeneous cluster, we define the *normalized batch size* as the amount of trained data samples per unit time divided by the number of iterations per unit time. Such a normalized batch size can be considered as the expectation of batch size of heterogeneous workers during asynchronous training. Accordingly, we define a *normalized iteration* as training a normalized batch in a heterogeneous cluster. In particular, a homogeneous cluster can be considered as a special case of a heterogeneous cluster, where the normalized batch size of a heterogeneous cluster is actually reduced to the batch size of a worker. As shown in Fig. 4(a), the DDNN training loss converges faster as the normalized batch size increases. This is because a larger normalized batch size generates a more accurate gradient, which can achieve an objective training loss value with fewer iterations [19]. Accordingly, we leverage the *normalized batch size* to generalize the relationship between the training loss and batch size in a homogeneous cluster [20] to the heterogeneous cluster.

Moreover, we examine the effect of the number of workers on training loss in a heterogeneous cluster. As illustrated in Fig. 4(b), we observe that the training loss converges more slowly as ResNet-50 is deployed on more workers, which is consistent with the findings observed in a homogeneous cluster [1]. The rationale is that workers can receive stale model parameters in the ASP mechanism and they inevitably miss more fresh parameter updates when the cluster scale increases [21]. Accordingly, an increased heterogeneous cluster scale can slow down the training convergence rate.

TABLE I: Comparison of the training time and monetary cost of a ResNet-110 model with different instance provisioning strategies. The training time in color red means SLO violations occur, while the training time in bold indicates that the time SLO is guaranteed.

Strategies	Provisioning plans	Time (secs)	Cost ($)
Random	5×g4dn.4xlarge, 4×g3.8xlarge	2,912.16	3.67
On-demand	5×g3.8xlarge, 3×g3.16xlarge	**2,549.02**	17.75
CM-DARE [7] + Srifty [12]	4×g3.8xlarge, 3×g3.16xlarge	2,710.85	5.59
spotDNN	5×g3.8xlarge, 3×g3.16xlarge	**2,549.02**	5.75

C. A Motivation Example

Blindly configuring heterogeneous clusters can either under-provision or over-provision instance resources, which can adversely affect the training performance. We design *spotDNN* in Sec. IV to identify a cost-effective instance provisioning plan with predictable DNN training performance. We conduct a motivation experiment on the ResNet-110 model to illustrate its effectiveness. Specifically, we use g4dn.4xlarge, g3.8xlarge, g3.16xlarge and p2.8xlarge instances each with a user quota of 5. The objective training loss is set at 0.8 and the objective training time is set to 2,600 seconds.

As shown in Table I, the Random strategy configures 9 workers with the lowest price, but fails to meet the objective time due to insufficient resource allocation. In addition, the CM-DARE+Srifty strategy, which combines the performance model of CM-DARE [7] and the instance provisioning strategy (*i.e.,* exhaustive search) of Srifty [12], produces a provisioning plan that exceeds the objective time by 110.85 seconds. This is because CM-DARE neglects the PS network bandwidth bottleneck and PCIe bandwidth limitation during the training process. It then over-estimates the training performance and under-provisions spot instances. In contrast, *spotDNN* meets the training performance SLOs and saves the monetary cost by up to 67.6% compared to the cluster provisioned with on-demand instances (*i.e.,* the On-demand strategy with *spotDNN* instance provisioning plan).

Summary. *First,* the PS network bandwidth and the PCIe bandwidth can easily become bottleneck resources for DDNN training in a heterogeneous cluster. *Second,* DDNN training

TABLE II: Key notations in our analytical DDNN training performance model in a heterogeneous cluster.

Notation	Definition
$\mathcal{N}$	Set of provisioned heterogeneous workers
j	Number of normalized iterations for a DNN model
T^i	Iteration time of a worker i
b^i	Batch size of a worker i
T_{norm}	Normalized iteration time of a heterogeneous cluster
b_{norm}	Normalized batch size of a heterogeneous cluster
v	Training speed of a heterogeneous cluster
T^i_{comm}	Communication time in each iteration of a worker i
S_{parm}	Parameter size of a DNN model
g^i	Number of GPUs in a worker i
B^i_{wk}	Available network bandwidth between a worker i and PS
B_{pcie}	Available PCIe bandwidth in workers
B_{ps}	Available bandwidth of a PS node

loss essentially depends on the normalized batch size and the number of workers in a heterogeneous cluster. *Finally,* judiciously provisioning spot instances can significantly save monetary cost while guaranteeing training performance SLOs for DDNN training workloads.

III. Modeling DDNN Training Performance in Heterogeneous Clusters

In this section, we proceed to devise a simple yet effective analytical model to capture the training performance of DNN models. Based on our analysis in Sec. II-B, we first model the training loss using the normalized batch size and number of workers. We then predict the DDNN training performance by explicitly considering the bottlenecks on the PS network bandwidth and the PCIe bandwidth. The notations in our performance model are summarized in Table II.

DDNN training requires a number of iterations to achieve convergence. However, the iteration time can be different for heterogeneous workers. As elaborated in Sec. II, we characterize the DDNN training process in a heterogeneous cluster with j normalized iterations and each iteration requires the normalized iteration time T_{norm}. Accordingly, we formulate the DDNN training time T as

$$T = j \cdot T_{norm}, \tag{1}$$

where a normalized iteration indicates a heterogeneous cluster trains a normalized batch with the size as b_{norm}. As discussed in Sec. II-B, the normalized iteration time T_{norm} can be considered as the expectation of the iteration time of heterogeneous workers, which is formulated as

$$T_{norm} = \frac{1}{\sum_{i \in \mathcal{N}} \frac{1}{T^i}}, \tag{2}$$

where $\frac{1}{T^i}$ denotes the number of iterations of a worker i per unit time.

Modeling DDNN Training Loss. As discussed in Sec. II-B, DDNN training loss converges faster as the normalized batch size b_{norm} gets larger, and the convergence rate slows down as more workers are provisioned. Moreover, DDNN training loss is inversely proportional to the normalized iterations j and converges at a rate of $\mathcal{O}(\frac{1}{j})$ [22]. The relationship between training loss and the number of provisioned workers $|\mathcal{N}|$ is as follows: $f_{loss} \propto \sqrt{|\mathcal{N}|}$. Accordingly, we empirically model the training loss in a heterogeneous cluster as

$$f_{loss}\left(b_{norm}, \mathcal{N}, j\right) = \frac{\left(\gamma_2 \cdot b_{norm} + \gamma_3\right)\sqrt{|\mathcal{N}|}}{j + \gamma_1} + \gamma_4, \tag{3}$$

where γ_1, γ_2, γ_3 and γ_4 are the model coefficients, and in general γ_2 <0.

We proceed to model the normalized batch size b_{norm}, which can be defined as the amount of data trained in a cluster per unit time divided by the total number of cluster iterations per unit time. Specifically, workers communicate with the PS without a barrier in the ASP mechanism, allowing the amount of data trained per unit time in a cluster be represented as the cluster training speed (*i.e.,* v). Similarly, the total number of cluster iterations per unit time can be identified as the sum of the iterations per worker per unit time (*i.e.,* the *inverse* of the normalized iteration time T_{norm}). Accordingly, we formulate the normalized batch size b_{norm} as

$$b_{norm} = v \cdot T_{norm}. \tag{4}$$

We calculate $v = \sum_{i \in n} \frac{b^i}{T^i}$, where b^i and T^i denote the batch size and the iteration time of a worker i , respectively.

Modeling Iteration Time of a Worker. Each iteration of DDNN training can be split into two phases: gradient computation and parameter communication, which are generally processed in sequential for the ASP mechanism. Accordingly, we formulate the iteration time T^i for a worker i as

$$T^i = T^i_{comm} + T^i_{comp}, \tag{5}$$

where T^i_{comm} and T^i_{comp} denote the communication time and GPU computation time in an iteration for worker i. Among them, the GPU computation time T^i_{comp} is strongly related to the GPU type and batch size. Accordingly, we can treat it as a constant value as long as the instances' GPU types stay the same as a result of the same batch size for comparable GPUs. We can obtain it by $T^i - T^i_{comm}$ during workload profiling.

The communication phase consists of the gradient aggregation through PCIe and the parameter communication through the network. In general, the pushing and pulling of parameters can be considered as equal, and the size of model gradients is the same as that of model parameters (*i.e.,* S_{parm}). Accordingly, the communication time T^i_{comm} for a worker i can be calculated as

$$T^i_{comm} = \frac{2 \cdot S_{parm}}{B^i_{wk}} + \frac{2 \cdot g^i \cdot S_{parm}}{B_{pcie}}, \tag{6}$$

where B^i_{wk} denotes the available network bandwidth between a worker i and PS, and B_{pcie} denotes the available PCIe bandwidth in a worker with g^i GPUs.

As evidenced in Sec. II-B, B^i_{wk} is restricted by the PS network bandwidth B_{ps}, which becomes a resource bottleneck as the number of provisioned workers increases. As shown

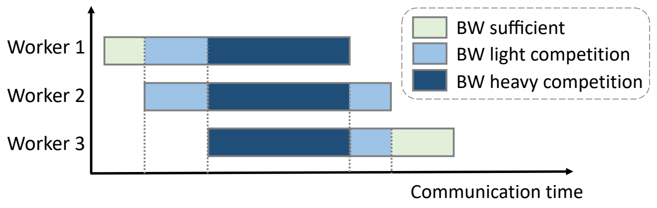


Fig. 5: Competition of PS network bandwidth (BW) over time as the communication between the worker and PS starts or ends.

in Fig. 5, the contention of PS network bandwidth only occurs during part of the communication phase, as workers communicate with the PS at different times. Accordingly, we formulate the available network bandwidth B_{wk}^i for a worker i as

$$B_{wk}^i = \begin{cases} P \cdot \frac{B_{ps}}{|\mathcal{N}|} + (1-P) \cdot B_{req} & B_{req} > \frac{B_{ps}}{|\mathcal{N}|}, \\ B_{req} & B_{req} < \frac{B_{ps}}{|\mathcal{N}|}, \end{cases} \tag{7}$$

where B_{req} denotes the network bandwidth requirement between a worker and PS when there is only one worker in the cluster. Moreover, P denotes the probability of PS network bandwidth bottleneck, which is positive correlation with the number of provisioned workers. We empirically fit it as

$$P = \min\left(\alpha_1 \cdot \sqrt{|\mathcal{N}|} + \beta_1,\ 1\right), \tag{8}$$

where α_1 and β_1 are model coefficients. In more detail, P has a maximum value of 1 when the bandwidth competition reaches its peak.

Identifying Batch Sizes for Different GPUs. As mentioned in Sec. II-B, we assign a GPU of type k with its largest batch size b^k, which is proportional to the GPU memory usage mem^k and is limited by the GPU memory size. Accordingly, we model the relationship between mem^k and b^k as follows:

$$mem^k = \alpha_2 \cdot b^k + \beta_2, \tag{9}$$

where α_2 and β_2 are model coefficients, which can be obtained by workload profiling. Based on the model above, we can calculate the batch size for worker i with g^i GPUs of type k as $b^i = g^i \cdot b^k$.

Obtaining Model Parameters. Based on the above, we have 5 *workload-specific* parameters (*i.e.,* S_{parm}, B_{req}, α_i, β_i, γ_i) and 4 *instance-specific* parameters (*i.e.,* T_{comp}^i, g^i, B_{pcie}, B_{ps}). Among them, the number of GPUs g^i in a worker i, the available PCIe bandwidth B_{pcie} and the available bandwidth of the PS B_{ps} can be found in the instance configuration documentation of cloud providers. Specifically, we fit the model coefficients α_2 and β_2 by running the DNN model on a single worker with a small number of iterations (*i.e.,* 30) and recording the GPU memory usage under different batch sizes. At the same time, the model parameter size S_{parm} can be measured as the network data traffic of the PS divided by the number of iterations, and the ideal network bandwidth B_{req} of a worker can be obtained using the `nethogs` tool. Then, we perform a small number of epochs (*i.e.,* 3) on three heterogeneous clusters with different numbers of workers (*i.e.,* 2, 4, 7) in parallel, from which we can get the iteration time of different GPUs as well as the model coefficients α_1, β_1 and γ_i (fitted by regression method [23]). Moreover, the computation time T_{comp}^i of different GPUs can be obtained by Eq. (5) and Eq. (6).

IV. Guaranteeing DDNN Training Performance with Cloud Spot Instances

In this section, we proceed to define the instance provisioning optimization problem of spot instances. We then design and implement our *spotDNN* instance provisioning strategy to deliver predictable DDNN training performance.

A. Optimizing Spot Instance Provisioning

Given the available instance types $\{0 \ldots m\}$, the user quotas Lim (*i.e.,* the available number of each type of instances) and the performance SLOs (*i.e.,* training time T_{obj} and loss value L_{obj}) for a DNN model, how can we appropriately provision spot instances in $\mathcal{N}$ to guarantee the DDNN training performance while minimizing the monetary cost. The optimization problem can be formulated as

$$\min_{\mathcal{N}} \quad C = T \cdot \sum_{k \in m} n_k \cdot p_k \tag{10}$$

$$\text{s.t.} \quad f_{loss}(b_{norm}, \mathcal{N}, j) = L_{obj}, \tag{11}$$

$$T \le T_{obj}, \tag{12}$$

$$n_k \le Lim_k, \quad \forall k \in m, n_k \in \mathcal{Z} \tag{13}$$

where n_k and p_k denote the provisioned number and the unit price of an instance type k, respectively. The output $\mathcal{N}$ is a list that stores the provisioned spot instances, which has a length of $\sum_{k \in m} n_k$. T_{obj} is considered as the remaining objective training time, which requires re-calculation when an instance revocation occurs. Constraint (11) and (12) guarantee the training performance SLOs in terms of L_{obj} and T_{obj}. Constraint (13) denotes the number of provisioned instances in each type k, which is below the user quotas.

Problem Analysis. According to Eq. (10), the monetary cost C is actually impacted by the number of provisioned workers $|\mathcal{N}|$ as well as the training time T. By submitting Eq. (2)-(9) into Constraint (12), we can deduce that the constraint on the training time T is *non-linear*. As a result, our optimization problem in Eq. (10) turns out to be in form of *non-linear integer programming*, which is an NP-hard problem [24]. To solve such a problem, we adopt a heuristic algorithm to acquire an appropriate solution for our instance provisioning problem.

We use the following two methods to narrow down the solution search space. *First,* we analyze the lower bound n_{lower} and upper bound n_{upper} of the number of provisioned workers $|\mathcal{N}|$. As evidenced in Sec. II-B, the resource contention of the PS network bandwidth gradually has a negative impact on the cluster speed as more workers are provisioned. To avoid severe performance degradation, we let $P < 1$. Consequently, we calculate the upper bound n_{upper} according to Eq. (8), as

$$n_{upper} = \left\lfloor \left(\frac{1-\beta_1}{\alpha_1}\right)^2 \right\rfloor. \tag{14}$$

According to Eq. (3), we can calculate the minimum data size needed to be trained to meet the objective loss value is $b_{min} \cdot \left(\frac{\gamma_2 \cdot b_{min}+\gamma_3}{L_{obj}-\gamma_4} - \gamma_1\right)$ when there is only one worker, where b_{min} is the minimum value of the batch size of the available instances. Therefore, to meet Constraint (12), we have the lower bound n_{lower} of the number of provisioned workers $|\mathcal{N}|$ as

$$n_{lower} = \left\lceil \frac{b_{\min} \cdot \left(\frac{\gamma_2 \cdot b_{\min}+\gamma_3}{L_{obj}-\gamma_4} - \gamma_1\right)}{v_{\max} \cdot T_{obj}} \right\rceil, \tag{15}$$

where v_{max} is the maximum value of training speed in all available instances.

Second, we provision spot instances with the similar iteration time. We make the decision for two reasons: on the one hand, workers with similar iteration time introduce a lower degree of staleness [25], which makes the training loss converge more stable; on the other hand, we can evaluate from Eq. (4) that the normalized batch size b_{norm} decreases as the difference in iteration time gets larger, which inevitably slows down the convergence rate. Accordingly, we sort all the available instances by iteration time and use a *sliding window* to choose adjacent instances with similar iteration time.

B. Design of spotDNN Instance Provisioning Strategy

As shown in Alg. 1, given a DNN model with the performance SLOs (*i.e.,* training time T_{obj} and loss value L_{obj}), the available instance types $\{0 \ldots m\}$, the user quotas Lim and the existing instances list E, *spotDNN* first obtain the *instance-specific* parameters (*i.e.,* T^k_{comp}, g^i, B_{pcie}, B_{ps}) and the *workload-specific* parameters (*i.e.,* S_{parm}, B_{req}, α_i, β_i, γ_i) using a *lightweight* profiling method mentioned in Sec. III (line 1). It then calculates the batch size b^k and the computation time T^k_{comp} for an instance type k (line 2). After initializing the monetary cost C, the provisioned plan $\mathcal{N}$ and the available instances list I, *spotDNN* calculates the upper bound n_{upper} and the lower bound n_{lower} of the number of provisioned workers $|\mathcal{N}|$ (lines 3-4). For each possible number n_i of $|\mathcal{N}|$, *spotDNN* considers the existing instances list E and uses a *sliding window* with the size n_i to choose possible provisioning plans $\widehat{\mathcal{N}}$ from the sorted instance list I (lines 5-8). *spotDNN* further calculates the number of normalized iterations j to meet the objective training loss value based on the normalized batch size b_{norm}. It then obtains the estimated training time and monetary cost (lines 9-10). When a revocation occurs, *spotDNN* adds an instance revocation overhead T_{ohd} (*e.g.,* less than 30 seconds) to the estimated training time $\widehat{T}$ (lines 11-13). Finally, *spotDNN* finds a cost-efficient instance provisioning plan $\mathcal{N}$ that guarantees the objective time T_{obj} and minimizes the monetary cost C (lines 14-18).

Remark. The complexity of *spotDNN* is in the order of $\mathcal{O}(x \cdot y)$, where $x = n_{upper} - n_{lower} + 1$ denotes the possible search space of the number of provisioned workers $|\mathcal{N}|$, and $y = |I| - n_i + 1$ denotes the candidate provisioning plan under each possible numbers n_i. As a result, the runtime overhead of *spotDNN* is well controlled and will be validated in Sec. V-D.

Algorithm 1: *spotDNN*: Heterogeneity-aware instance provisioning strategy for predictable performance of DDNN training with cloud spot instances.

```
Input: Performance SLOs (i.e., training time $T_{obj}$ and loss value
    $L_{obj}$) for a DNN model, the available instance types $\{0 \ldots m\}$,
    the user quotas $Lim$ as well as the existing instances list $E$.
Output: Instance provisioning plan $\mathcal{N}$ and the monetary cost $C$.
1: Acquire instance-specific parameters (i.e., $T^k_{comp}$, $g^i$, $B_{pcie}$,
   $B_{ps}$), and obtain workload-specific parameters (i.e., $S_{parm}$,
   $B_{req}$, $\alpha_i$, $\beta_i$, $\gamma_i$) by lightweight workload profiling;
2: Calculate the batch size $b^k \leftarrow Eq.$ (9) and the computation
   time $T^k_{comp} \leftarrow Eq.$ (5) for a single-GPU instance of type $k$;
3: Initialize: $C \leftarrow \infty$; $\mathcal{N} \leftarrow \emptyset$; Put every instance available into
   the list $I$ without deleting duplicate instances;
4: Calculate the upper bound $n_{upper} \leftarrow Eq.$ (14), the lower
   bound $n_{lower} \leftarrow Eq.$ (15), the iteration time $T^k \leftarrow Eq.$ (5) of
   a instances type $k$ with the required network bandwidth $B_{req}$;
5: Sort the available instances list $I$ according to $T^k$ in a
   descending order;
6: for all $n_i \in [n_{lower},\ n_{upper}]$ do
7:     for all $idx \in [0,\ |I| - n_i]$ do
8:         Acquire $\widehat{\mathcal{N}} \leftarrow E + I\,[idx,\ idx + n_i - 1]$ by sequentially
           selecting $n_i$ instances from the $idx$-th instance in the
           available instances list $I$;            // sliding window
9:         Calculate the normalized batch size $b_{norm} \leftarrow Eq.$ (4),
           and the number of normalized iterations $j$ to meet the
           objective loss value $f_{loss}(b_{norm},\ \widehat{\mathcal{N}}, j) \leftarrow L_{obj}$;
10:        Calculate $\widehat{T} \leftarrow Eq.$ (1), $\widehat{C} \leftarrow Eq.$ (10);
11:        if a revocation occurs then
12:            $\widehat{T} \leftarrow \widehat{T} + T_{ohd}$;      // add revocation overhead
13:        end if
14:        if $\widehat{T} \le T_{obj}$ && $\hat{C} < C$ then
15:            Record the monetary cost $C \leftarrow \widehat{C}$, and the instance
               provisioning plan $\mathcal{N} \leftarrow \widehat{\mathcal{N}}$;
16:        end if
17:    end for
18: end for
```

C. Implementation of our spotDNN Prototype

We implement a prototype of *spotDNN* on Amazon EC2 spot instances [5] based on TensorFlow [26] v1.15.0 with over $1,000$ lines of Python and Linux Shell codes. Our prototype can be deployed on an EC2 instance (*i.e,* t2.micro). To avoid network traffic across different Availability Zones, we place all instances within one Amazon Virtual Private Cloud (VPC) and determine the range of the VPC IPv4 address (*i.e.,* CIDR block) by the number of available instances. The instance launcher uses the AWS CLI command `aws ec2 request-spot-instances` to request the corresponding instances in the provisioning plan, particularly setting the `subnetID` according to the VPC information.

To timely capture revocation events, our revocation detector periodically checks the instance status (*i.e.,* 15 seconds) using the Amazon *instance metadata service*[3]. When a coming revocation is detected, the revocation detector sends the current training status (*i.e.,* remaining instances and unfinished

[3]https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2-instance-metadata.html

TABLE III: Configurations of EC2 instances provisioned in our experiment.

Instance type	GPU devices	Max quota	Unit price ($)	
			spot[4]	on-demand
p3.2xlarge	1× V100	10	0.92	3.06
p2.8xlarge	8× K80	10	2.16	7.20
g4dn.12xlarge	4× T4	10	1.18	3.91
g4dn.4xlarge	1× T4	10	0.36	1.20
g3.16xlarge	4× M60	10	1.57	4.56
g3.8xlarge	2× M60	10	0.69	2.28
g3.4xlarge	1× M60	10	0.37	1.14

performance SLOs) to the training performance predictor, which re-predicts the performance and then relaunches a series of instances added to the cluster to guarantee the training performance. To implement dynamic worker addition without interrupting the training process, *spotDNN* uses the *sparse mapping* [6] technique to put all the available IPv4 addresses of the VPC into the cluster configuration at the start of training, and then gives the newly added workers with IPv4 addresses within the same VPC to enable communication. In particular, the worker relaunch process is lightweight as the new workers can be online within three minutes in our experiments.

V. Performance Evaluation

In this section, we evaluate *spotDNN* by carrying out a set of prototype experiments with four representative DNN models (as listed in Table. IV) on Amazon EC2 [5]. Specifically, we answer the following questions:

- **Accuracy:** Can our performance model in *spotDNN* accurately predict the DDNN training performance in a heterogeneous environment? (Sec. V-B)
- **Effectiveness:** Can our instance provisioning strategy in *spotDNN* provide predictable training performance while saving monetary cost? (Sec. V-C)
- **Overhead:** How much runtime overhead of workload profiling and algorithm computation does *spotDNN* practically bring? (Sec. V-D)

A. Experimental Setup

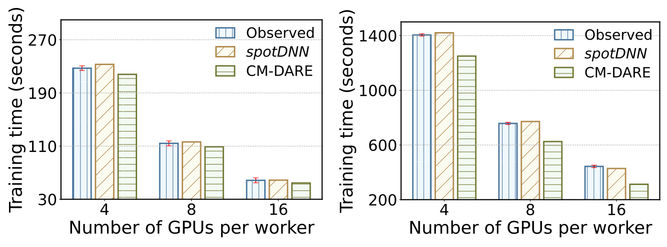
Training Cluster Configurations. We provision the instances in the same VPC in the us-east-1b region of AWS EC2. Specifically, we use an m5.xlarge on-demand instance to serve as the PS with workers on seven representative types of instances as listed in Table III. Besides, the available bandwidth for the PS B_{ps} and the available PCIe bandwidth B_{pcie} inside a worker are set as $1,200$ MBps and 10 GBps, respectively.

Configurations of DNN Training Workloads. We select four representative DNN models as listed in Table. IV. Due to the budget limit, we adopt CIFAR-100 [15] as the training dataset for VGG-19 [16], Inception-v3 [27] and ResNet-110 [14] models for image classification. We also include

[4]We list the spot price during the period of our experiments (Sep. 2022).

TABLE IV: Workload-specific parameters of four representative DNN models.

Workload	ResNet-110	EsperBERTo	VGG-19	Inception-v3
Dataset	CIFAR-100	OSCAR	CIFAR-100	CIFAR-100
S_{parm} (MB)	11.54	336	574	98
B_{req} (MBps)	142	543	620	562
α_1, β_1	0.81, -1.92	0.42, -0.27	1.00, -1.41	0.41, -0.40



(a) ResNet-110 (b) VGG-19

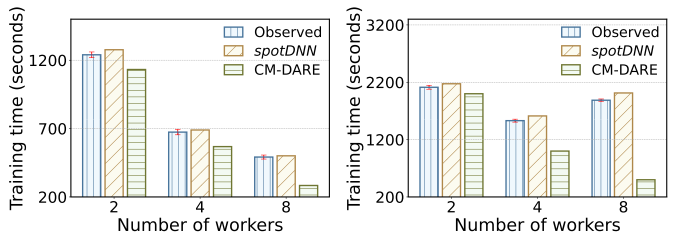
Fig. 6: Comparison of the observed and predicted training time with *spotDNN* and CM-DARE, using a P2 instance by varying the number of GPUs per instance from 4 to 16.

a DNN model from NLP (*i.e.,* EsperBERTo [28]) trained on the Esperanto portion of OSCAR dataset [29]. Through workload profiling according to Sec. III, we can obtain the key *workload-specific* parameters as elaborated in Table IV.

Baselines and Metrics. We compare *spotDNN* with the following three strategies: (1) On-demand strategy, which uses on-demand instances to provision workers based on the instance provisioning strategy of *spotDNN*; (2) CM-DARE$^+$, which considers the cluster speed as the sum of ideal GPU training speeds [7] and adopts the instance provisioning strategy in *spotDNN*; (3) Srifty$^+$, which patches the performance model in *spotDNN* and can be considered as the optimal solution due to exhaustive search [12]. We focus on three key metrics including the DDNN training time, the monetary cost and the computation overhead.

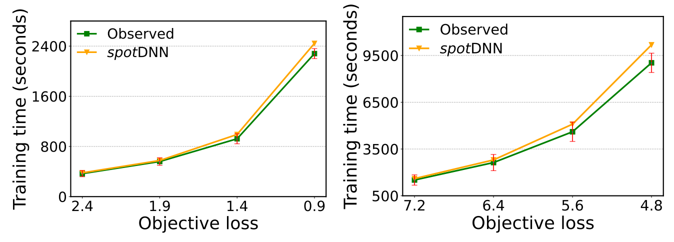
B. Validating Prediction Accuracy of DDNN Training Performance

Can *spotDNN* predict the DDNN training time well? As shown in Fig. 6, *spotDNN* can well predict the DDNN training time with a prediction error from 0.2% to 2.5%, in comparison of 4.0% to 28.5% achieved by CM-DARE. Specifically, the training time of the ResNet-110 model predicated by CM-DARE is comparatively accurate as shown in Fig. 6(a). However, CM-DARE poorly predicts the DDNN training time as the proportion of time spent on gradient aggregation increases. As shown in Fig. 6(b), the prediction error of the VGG-19 achieved by CM-DARE gets larger from 13.7% to 28.5% when the number of GPUs within a worker changes from 4 to 16. The rationale is that the limited PCIe bandwidth increases the time for gradient aggregation within a worker and leads to non-linear speedup in training speed even when scaling up the number of GPUs.



(a) Inception-v3 (b) EsperBERTo

Fig. 7: Comparison of the observed and predicted training time with *spotDNN* and CM-DARE in an n-worker heterogeneous cluster which consists of $\frac{n}{2}$ g4dn.4xlarge and $\frac{n}{2}$ g3.16xlarge workers.

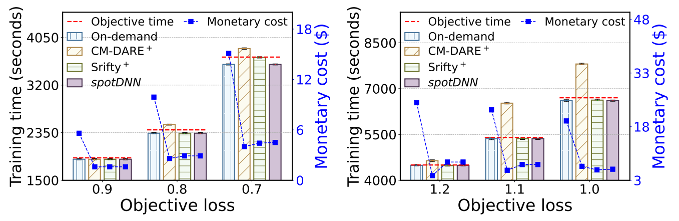


(a) ResNet-110 (b) EsperBERTo

Fig. 8: Comparison of the observed and predicted training time under different objective loss values with a 6-worker heterogeneous cluster which consists of 3 g4dn.4xlarge and 3 g3.16xlarge workers.

In addition, we examine the DDNN training time of Inception-v3 and EsperBERTo models by varying the number of workers in heterogeneous clusters. As shown in Fig. 7, we observe that *spotDNN* can accurately predict the DDNN training performance with a prediction error of 1.8% to 6.9%. On the contrary, CM-DARE shows terrible prediction results with an error of up to 73.5%. Moreover, *spotDNN* shows an advantage in the prediction of models with a larger number of parameters (*i.e.,* EsperBERTo) compared with CM-DARE, as shown in Fig. 7(b). The rationale is that there is steadily increasing competition for PS network bandwidth as the number of workers increases, which is ignored by CM-DARE. When the number of workers changes from 4 to 8, the speedup from adding workers is smaller than the slowdown due to the resource bottleneck of the PS network bandwidth, and thus the training time increases. *spotDNN* is able to predict the competition of PS network bandwidth well for different numbers of workers, while CM-DARE simply considers the cluster speed as the sum of GPU speeds, which leads to inaccurate training time prediction.

Can *spotDNN* predict the DDNN training time given an objective loss value? We further evaluate the accuracy of our DDNN training loss model with training loss values as 3.0, 2.2, 1.4, 0.6 for the ResNet-110 model and 7.2, 6.4, 5.6, 4.8 for the EsperBERTo model respectively. As shown in Fig. 8, we observe that *spotDNN* can basically predict the training time under different objective loss values. Specifically, *spotDNN* predicts the training time of the ResNet-110 with a prediction error from 2.9% to 7.1%, and the EsperBERTo with a prediction error from 5.9% to 12.7%. We observe a lower prediction accuracy for the EsperBERTo compared to the ResNet-110. This is because: first, the contention of the PS network bandwidth becomes evenly severe as the number of workers increases to 8, which makes the training performance become unstable with a large standard deviation (*i.e.,* up to 620.2 seconds); second, we slightly underestimate the training speed of the cluster (as shown in Fig. 7(b)) due to the complex communication overlap in the network (discussed in Sec. III).



(a) ResNet-110 (b) Inception-v3

Fig. 9: Comparison of DDNN training time and monetary cost achieved by various instance provisioning strategies under different performance SLOs.

TABLE V: Comparison of instance provisioning plans achieved by on-demand, CM-DARE$^+$, Srifty$^+$, and *spotDNN* for ResNet-110 and Inception-v3 under three performance SLOs. The provisioning plan in the array is defined as [#p3.2xl, #g4dn.4xl, #g3.8xl, #g3.16xl, #p2.8xl] with the user quotas of 2, 5, 5, 8 and 5, respectively.

Workloads/		**Performance SLOs** (loss value, training time)		
	Strategies	(0.9, 1, 900)	(0.8, 2, 400)	(0.7, 3700)
ResNet-110	On-demand	[2, 0, 2, 0, 0]	[2, 0, 4, 0, 0]	[2, 1, 4, 0,0]
	CM-DARE$^+$	[2, 0, 2, 0, 0]	[2, 0, 3, 0, 0]	[2, 0, 3, 0, 0]
	Srifty$^+$	[2, 0, 2, 0, 0]	[2, 0, 4, 0, 0]	[2, 1, 3, 0, 0]
	spotDNN	[2, 0, 2, 0, 0]	[2, 0, 4, 0, 0]	[2, 1, 4, 0, 0]
		(1.2, 4, 500)	(1.1, 5, 400)	(1.0, 6, 700)
Inception-v3	On-demand	[2, 0, 0, 3, 0]	[2, 0, 0, 2, 0]	[2, 0, 1, 0, 0]
	CM-DARE$^+$	[2, 0, 0, 1, 0]	[0, 5, 2, 0, 0]	[0, 5, 2, 0, 0]
	Srifty$^+$	[2, 0, 0, 3, 0]	[2, 0, 0, 2, 0]	[2, 0, 2, 0, 0]
	spotDNN	[2, 0, 0, 3, 0]	[2, 0, 0, 2, 0]	[2, 0, 1, 0, 0]

C. *Effectiveness of spotDNN Instance Provisioning Strategy*

Can *spotDNN* guarantee the DDNN training performance while minimizing the monetary cost? To examine the efficiency of our instance provision strategy, we set different training performance SLOs listed in Table V for ResNet-110 and Inception-v3 models. As shown in Fig. 9, we observe that *spotDNN* can well meet the objective training time under different objective loss values, and save the monetary cost by up to 68.1% compared with On-demand strategy. Specifically, CM-DARE$^+$ can hardly provide predictable training performance except for the ResNet-110 model with the objective loss value of 0.9 in Fig. 9(a), since it ignores the resource bottleneck of the PS network bandwidth. Besides, both *spotDNN* and Srifty$^+$ can guarantee training performance. *spotDNN* generally returns the optimal plans with Srifty$^+$ in Table V, and can achieve similar monetary cost with an increase of at most 4.6% (*i.e.,* objective loss of 1.0 in Fig. 9(b)). However, *spotDNN* can identify provisioning plans 84.6% faster than

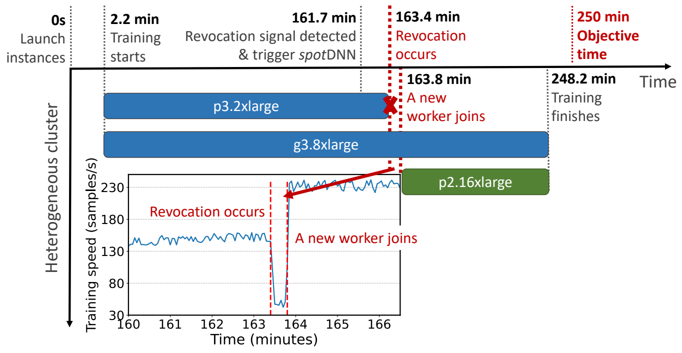


Fig. 10: Timeline of *spotDNN* dealing with a spot instance revocation.

Srifty$^+$ (*i.e.,* 0.2 seconds *vs.* 1.3 seconds) in computation overhead, which we will explain in detail in Sec. V-D.

Can *spotDNN* guarantee DDNN training performance when a revocation occurs? *spotDNN* leverages the current training status to update constraints and adds new instances to satisfy the original performance SLOs with acceptable runtime overhead. We train EsperBERTo model with an objective loss of 4.0 and an objective time of 250 minutes. As shown in Fig. 10, *spotDNN* starts training with 1 p3.2xlarge instance and 1 g3.8xlarge instance. The revocation detector catches the revocation signal at 161.7 minutes and finds that the AWS platform will withdraw the p3.2xlarge instance in 1.7 minutes. To guarantee the training performance, the performance predictor works with the spot instance provisioner to generate a new provisioning plan based on the current training status (*i.e.,* remaining instances and unfinished performance SLOs). The newly added worker p2.16xlarge spends 2.1 minutes to join the training, which is slightly longer than the remaining running time of the p3.2xlarge instance (*i.e.,* from the 161.7-th to 163.4-th minute). Due to the fast response of *spotDNN*, the training speed degradation lasts only for 24 seconds (*i.e.,* from the 163.4-th to 163.8-th minute), as evidenced in Fig. 10. At the 248.2-th minute, DDNN training finishes and successfully achieves the performance SLOs.

D. Runtime Overhead of spotDNN

We further evaluate the runtime overhead of *spotDNN* in terms of the workload profiling overhead and the computation overhead of the instance provisioning strategy (*i.e.,* Alg. 1). Specifically, we launch a p2.xlarge EC2 instance to profile the *workload-specific* coefficients (*i.e.,* S_{param}, B_{req}, α_2, β_2) for DDNN training workloads. By training the DNN models for 30 iterations, the profiling time of ResNet-110 [14], Inception-v3 [27], VGG- 19 [16], and EsperBERTo [28] models are 114, 75, 141, and 21.9 seconds, respectively. Besides, the profiles of the remaining parameters (*i.e.,* α_1, β_1, γ_i) are carried out in parallel on 3 heterogeneous clusters with the number of provisioned workers of 2, 4, 7, respectively. We train CIFAR-100 for 3 epochs and Esperanto dataset for 1 epoch, and the profiling time are 0.6, 5.8, 12.8 and 11.7 minutes, respectively. Such job profiling overhead is negligible compared to the hours of training time required by a typical DNN workload.

To clarify the computation overhead of *spotDNN*, we conduct another experiment on a ResNet-110 model with all available instances in Table III and apply the maximum user quotas. As a result, *spotDNN* returns the same optimal provisioning plan as Srifty$^+$ with a negligible increase in computation overhead (*i.e.,* from 0.2 seconds to 0.3 seconds). Consequently, the runtime overhead of *spotDNN* is practically acceptable. However, the computation overhead of Srifty$^+$ increases by $1.0e5$ times (*i.e.,* from 1.3 seconds to $1,503.7$ seconds). The rationale is that the exhaustive search method used by Srifty$^+$ has a complexity of $\mathcal{O}\left(\prod_{i\in m} \mathrm{Lim}_i\right)$, so the computation overhead grows exponentially as more instances are available. Accordingly, Srifty$^+$ is difficult to be deployed in real-world scenarios with a large number of available instances.

VI. Related Work

Resource Provisioning of DDNN Training. To improve the training performance, Proteus [30] elastically scales the cluster resources with low-price spot instances. To minimize the training time with a limited budget, FC^2 [31] provisions a cluster of the most cost-effective instances for DDNN training workloads. λDNN [32] and Cynthia [1] provision a cluster of homogeneous workers with serverless functions and EC2 instances, respectively. Different from prior works above, we focus on DDNN training in heterogeneous environments while taking unexpected revocations of spot instances into account.A more recent work Srifty [12] leverages the exhaustive search method to provision heterogeneous spot instances for DDNN training with Ring-AllReduce. In contrast, *spotDNN* develops a heuristic algorithm to dramatically reduce the runtime overhead as evidenced in Sec. V-D. Moreover, *spotDNN* explicitly constructs an analytical performance model to predict training speed, while Srifty relies on trained regression models which highly depend on the quality of training data.

Performance Modeling of DDNN Training. Habitat [33] leverages wave scaling to predict the training time of different training operations across different GPUs. Li et al. [34] predict ASP training throughput by simulating the interaction and transmission scheduling between PS and workers. However, the two works above fail to consider the performance objective of training loss. Optimus [22] builds a performance model by exploiting the pattern of computation and communication of DDNN training without considering the instance heterogeneity. Jiang et al. [35] *qualitatively* analyze the impact of heterogeneity on the training loss of DNN models. CM-DARE [7] performs regression fitting on the training speed of a heterogeneous cluster without considering the performance degradation caused by bottleneck resources. Different from the prior works, *spotDNN quantitatively* models the training loss by introducing *normalized* batch size and the number of workers. It also models the DDNN training performance of a heterogeneous cluster by explicitly considering the contention of PS network bandwidth and PCIe bandwidth in ASP.

Fault Tolerance of Cloud Spot Instances. Several recent works adopt checkpointing to resume DDNN training. For example, CheckFreq [3] profiles the checkpointing overhead online to adjust the checkpoint frequency at the iteration-level granularity. Kadupitiya et al. [36] build a three-stage revocation probability model to determine the checkpoint frequency. iSpot [2] designs a critical data checkpointing mechanism for Spark jobs. To deal with training data loss, Spotnik [11] proposes an adaptive collective communication and synchronization to recover from instance revocations in Ring-AllReduce. Orthogonal to the works above, we focus on asynchronous training under the PS architecture, which can *intrinsically* maintain DDNN training and avoid training data loss even when instance revocations occur.

VII. Conclusion and Future Work

To provide predictable performance and save the user budget for DDNN training workloads, this paper presents *spotDNN*, a heterogeneity-aware spot instance provisioning framework in the cloud. By explicitly considering the severe contention of the bottleneck resources (*i.e.,* the PS network bandwidth and PCIe bandwidth), *spotDNN* devises a lightweight analytical performance model for DDNN training in a heterogeneous cluster. Based on such a performance model, it further provisions an adequate number and type of spot instances to train the DNN model within an objective training time and loss value while minimizing the monetary cost. Extensive prototype experiments on AWS EC2 demonstrate that *spotDNN* can deliver predictable DDNN training performance and save the monetary cost by up to 68.1%, in comparison to the state-of-the-art instance provisioning strategies.

As our future work, we plan to extend *spotDNN* to extremely large DNN models and large training datasets (*e.g.,* ImageNet), and examine the effectiveness of *spotDNN* for real-world DDNN training workloads in enterprises.

References

[1] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "Cynthia: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training," in *Proc. of ICPP*, Aug. 2019, pp. 1–11.

[2] F. Xu, H. Zheng, H. Jiang, W. Shao, H. Liu, and Z. Zhou, "Cost-effective cloud server provisioning for predictable performance of big data analytics," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1036–1051, 2018.

[3] J. Mohan, A. Phanishayee, and V. Chidambaram, "CheckFreq: Frequent,Fine-Grained DNN Checkpointing," in *Proc. of USENIX FAST*, Feb. 2021, pp. 203–216.

[4] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Analysis and exploitation of dynamic pricing in the public cloud for ml training," in *Proc. of VLDB Workshop*, Aug. 2020, pp. 1–8.

[5] Amazon. (2022, Sep.) Amazon EC2 Spot Instances. [Online]. Available: https://aws.amazon.com/cn/ec2/spot/

[6] S. Li, R. J. Walls, L. Xu, and T. Guo, "Speeding up deep learning with transient servers," in *Proc. of USENIX ICAC*, Jun. 2019, pp. 125–135.

[7] S. Li, R. J. Walls, and T. Guo, "Characterizing and Modeling Distributed Training with Transient Cloud GPU Servers," in *Proc. of IEEE ICDCS*, Nov. 2020, pp. 943–953.

[8] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. of NIPS*, pp. 1–9.

[9] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads," in *Proc. of USENIX OSDI*, Nov. 2020, pp. 481–498.

[10] C. Chen, Q. Weng, W. Wang, B. Li, and B. Li, "Accelerating Distributed Learning in Non-Dedicated Environments," *IEEE Transactions on Cloud Computing*, vol. preprint, pp. 1–17, 2021.

[11] M. Wagenlander, L. Mai, G. Li, and P. Pietzuch, "Spotnik: Designing distributed machine learning for transient cloud resources," in *Proc. of USENIX HotCloud*, Jul. 2020, pp. 1–8.

[12] L. Luo, P. West, P. Patel, A. Krishnamurthy, and L. Ceze, "SRIFTY: Swift and Thrifty Distributed Neural Network Training on the Cloud," in *Proc. of MLSys*, Aug. 2022, pp. 833–847.

[13] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," in *Proc. of NIPS*, Dec. 2012, pp. 1–9.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of IEEE CVPR*, Jun. 2016, pp. 770–778.

[15] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Report, 2009.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. of ICLR*, May 2015, pp. 1–14.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[18] A. Sharma, V. M. Bhasi, S. Singh, R. Jain, J. R. Gunasekaran, S. Mitra, M. T. Kandemir, G. Kesidis, and C. R. Das, "Analysis of Distributed Deep Learning in the Cloud," *arXiv preprint arXiv:2208.14344*, 2022.

[19] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proc. of NIPS*, Dec. 2015, pp. 2737–2745.

[20] 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.

[21] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," in *Proc. of ICLR Workshop*, May 2016, pp. 1–10.

[22] Y. eng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proc. of EuroSys*, Apr. 2018, pp. 1–14.

[23] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012.

[24] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[25] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *Proc. of IEEE ICDM*, Feb. 2016, pp. 171–180.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: a system for Large-Scale machine learning," in *Proc. of USENIX OSDI*, Aug. 2016, pp. 265–283.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. of IEEE CVPR*, Jun. 2016, pp. 2818–2826.

[28] Huggingface. (2022, Sep.) EsperBERTo. [Online]. Available: https://huggingface.co/blog/how-to-train

[29] A. Chagu. (2022, Sep.) OSCAR dataset. [Online]. Available: https://oscar-corpus.com/

[30] A. Harlap, A. Tumanov, A. Chung, G. R. Ganger, and P. B. Gibbons, "Proteus: agile ml elasticity through tiered reliability in dynamic resource markets," in *Proc. of EuroSys*, Apr. 2017, pp. 589–604.

[31] N. B. D. Ta, "FC^2: cloud-based cluster provisioning for distributed machine learning," *Cluster Computing*, vol. 22, no. 4, pp. 1299–1315, 2019.

[32] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "λDNN: Achieving Predictable Distributed DNN Training With Serverless Architectures," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 450–463, 2022.

[33] X. Y. Geoffrey, Y. Gao, P. Golikov, and G. Pekhimenko, "Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training," in *Proc. of USENIX ATC*, Jul. 2021, pp. 503–521.

[34] Z. Li, W. Yan, M. Paolieri, and L. Golubchik, "Throughput prediction of asynchronous sgd in tensorflow," in *Proc. of ACM/SPEC ICPE*, Apr. 2020, pp. 76–87.

[35] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proc. of ACM SIGMOD*, May 2017, pp. 463–478.

[36] J. Kadupitige, V. Jadhao, and P. Sharma, "Modeling the temporally constrained preemptions of transient cloud vms," in *Proc. of ACM HPDC*, Jun. 2020, pp. 41–52.