

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

Ruitao Shang<sup>Y</sup>, Fei Xu<sup>Y</sup>, Zhuoyan Bai<sup>Y</sup>, Li Chen<sup>Z</sup>, Zhi Zhou<sup>X</sup>, Fangming Liu<sup>Y</sup> Shanghai Key Laboratory of Multidimensional Information Processing, East China Normal University.

<sup>Z</sup>University of Louisiana at Lafayette. <sup>X</sup>Sun Yat-sen University.

<sup>Y</sup>Huazhong University of Science and Technology, and Peng Cheng Laboratory.

**Abstract**—Distributed Deep Neural Network (DDNN) training on cloud spot instances is increasingly compelling as it can significantly save the user budget. To handle 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 result in unpredictable DDNN training performance, mainly because bottlenecks occur on the parameter server network bandwidth and PCIe bandwidth resources, as well as the inadequate cluster heterogeneity. To address the challenges above, we propose spotDNN, a heterogeneity-aware spot instance provisioning framework that provides predictable performance for DDNN training in the cloud. By explicitly considering the contention for bottleneck resources, we first build an analytical performance model of DDNN training in heterogeneous clusters. It leverages the weighted average batch size and convergence coefficient to quantify the DDNN training loss in heterogeneous clusters. Through a lightweight workload profiling, we further design a cost-efficient instance provisioning strategy which incorporates the bounds calculation and sliding window techniques to effectively guarantee the training performance service level objectives (SLOs). We have implemented a prototype of spotDNN and conducted 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% compared to the existing solutions, yet with 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 [2], 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 [3]. Though at highly reduced prices, deploying DDNN training workloads on spot resources can suffer from

Corresponding author: Fei Xu (fxu@cs.ecnu.edu.cn). This work was supported in part by the NSFC under Grant 61972158, the Science and Technology Commission of Shanghai Municipality under Grants 20511102802 and 22DZ2229004, the BoRSF under Grants LEQSF(2019-22)-RD-A-21 and LEQSF(2021-22)-RD-D-07, the NSF under Grant OIA-2019511, the Guangdong Basic and Applied Basic Research Foundation under Grant 2023B1515020120, the National Key Research & Development (R&D) Plan under Grant 2022YFB4501703, and the Major Key Project of PCL under Grant PCL2022A05.

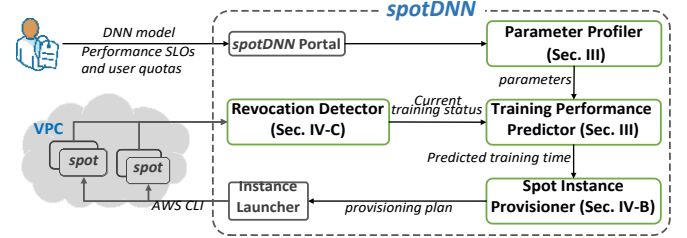


Fig. 1: Overview of spotDNN.

severe performance degradation [4], which is mainly caused by unexpected instance revocations [5] and insufficient spot capacity due to user quotas [6]. To alleviate such performance degradation for DDNN training, provisioning a heterogeneous cluster of spot instances using the asynchronous parallel (ASP) mechanism [7] is becoming the first choice for cloud users.

However, finding the cost-efficient provisioning plan of heterogeneous clusters (i.e., identifying 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 [8]. Unfortunately, blindly provisioning heterogeneous instances can result in unpredictable DDNN training performance due to the following two facts. First, the network bandwidth of the parameter server (PS) [9] and the limited PCIe bandwidth inside the instance can easily become bottleneck resources. As evidenced by our motivation experiment in Sec. II-B, the training time of VGG-19 can be prolonged by up to 65:7% as more spot instances are provisioned. Second, the inadequate cluster heterogeneity can impact the convergence rate of DDNN training due to the gradient staleness. Another motivation experiment in Sec. II-B shows that the convergence rate of ResNet-50 can vary by up to 39:8% as tuning the configuration of heterogeneous clusters. Accordingly, how to adequately provision spot instances to alleviate such unpredictable performance becomes the main obstacle to training DDNN models cost-efficiently in the cloud.

To tackle such performance issues above, many efforts have been devoted to scheduling training jobs (e.g., Gavel [10]), tuning the batch size (e.g., LB-BSP [11]), migrating tasks with spot price prediction (e.g., FarSpot [12]), and online batching (e.g., DOLL [13]) in homogeneous DDNN training clusters. However, relatively little attention has been paid to adequately provision spot instances and guaranteeing the DDNN training performance in heterogeneous clusters. There have recently

been works (e.g., CM-DARE [8]) on modeling the training performance of heterogeneous clusters. Nevertheless, they imprecisely predict the training time due to neglecting the performance degradation caused by bottleneck resources. Though a more recent work (i.e., Siftly [14]) leverages regression models and exhaustive search to identify the optimal instance provisioning plan in a heterogeneous cluster, it is hard to be applied in practice due to the heavy algorithm computation overhead. Moreover, characterizing the DDNN training loss in heterogeneous clusters has surprisingly been received little attention. As a result, scant research has been devoted to providing predictable DDNN training performance (i.e., time and loss) by adequately 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 DDNN training performance service level objectives (SLOs) in terms of objective training time and loss, while saving the training budget. By obtaining the parameters through a lightweight workload profiling with the parameter profiler, we first devise an analytical performance model of DDNN training in a heterogeneous cluster. The training performance predictor in spotDNN explicitly considers the performance degradation caused by bottleneck resources including the PS network bandwidth and limited PCIe bandwidth. It also leverages the weighted average batch size and the convergence coefficient to quantify 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 and minimize the training budget. It utilizes bounds calculation and sliding window to significantly reduce the algorithm computation overhead. 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 it provisions takeover spot instances to guarantee the workload performance SLOs.

Finally, we implement a prototype<sup>1</sup> of spotDNN on AWS EC2 [6] 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 while saving the user budget by up to 68.1% compared to the existing solutions, yet with a speedup of 10 in computation overhead compared to the cutting-edge solution (i.e., Siftly [14]).

The rest of the paper is organized as follows. Sec. II presents the background and motivation of this paper. Sec. III formulates our DDNN training performance model, which guides the design and implementation of our spotDNN instance provisioning strategy in Sec. IV. 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

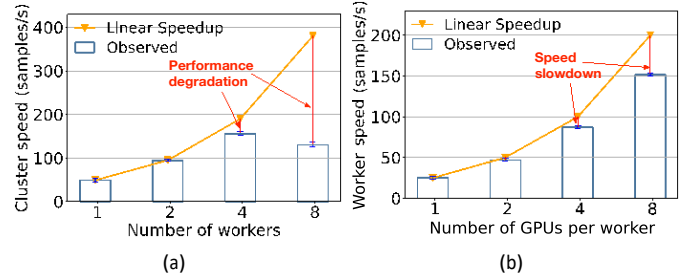


Fig. 2: Training speed of VGG-19 on ImageNet obtained by (a) an  $n$ -worker cluster consisting of  $\frac{n}{2}$  g3.4xlarge instances and  $\frac{n}{2}$  g4dn.4xlarge instances, and (b) a G4dn instance with various number of GPUs.

cluster. We then present a motivation example to show how to provision heterogeneous spot instances to save the 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., user quota) [6]. 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 tasks of revoked instances as different instance types are commonly not revoked at the same time [5]. To efficiently train DNN models on heterogeneous spot instances, we focus on the ASP [15] mechanism under the PS architecture [9]. We have the following three benefits. First, ASP allows each worker<sup>2</sup> to communicate with the PS individually, and thus it breaks the synchronization barrier in the heterogeneous environment. Second, the training process will not be interrupted in ASP even when most of the workers are revoked [8]. Third, the newly-launched takeover instances can pull the latest model parameters directly from the PS, so as to reduce 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 [6], by deploying P3, P2, G4dn, and G3 GPU instance types as workers and an m5.xlarge on-demand instance as the PS. We run three representative DDNN training workloads including ResNet-50 and ResNet-110 [16] on the CIFAR-100 [17] dataset, as well as VGG-19 [18] on a portion of the ImageNet [19] dataset. We illustrate the observed experiment results with error bars of standard deviation by repeating experiments 3 times.

**Training Speed.** To accelerate DDNN training, we simply set the batch size to fully utilize the GPU resource for each GPU type [11]. As shown in Fig. 2(a), the observed cluster

<sup>1</sup><https://github.com/icloud-ecnu/spotDNN>

<sup>2</sup>We use workers and instances interchangeably in this paper.

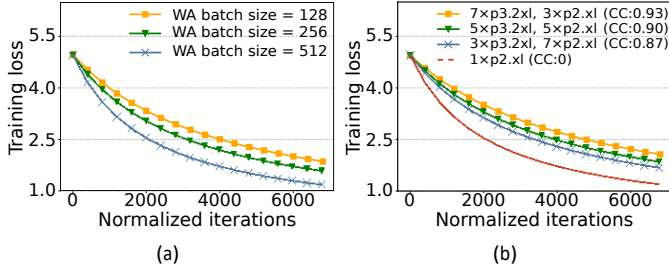


Fig. 3: Training loss of ResNet-50 on CIFAR-100 achieved by (a) various WA batch sizes on a heterogeneous cluster consisting of 6 workers (i.e., 3 g3.4xl, 3 g4dn.4xl), and (b) various CCs on a heterogeneous cluster consisting of 10 workers with the WA batch size set as 256.

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. To validate our analysis above, we record the PS network throughput over time, and the results show that the PS network bandwidth becomes saturated (i.e., reaching up to 1:2 GBps) as the number of workers grows to around 8. Such an observation confirms our analysis of PS network bandwidth as a bottleneck resource even when training with the ASP mechanism in a heterogeneous cluster.

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. 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, the PCIe bandwidth contention can cause a moderate speed slowdown, resulting in a non-linear acceleration of the training speed on a multiple-GPU instance.

**Training Loss.** To characterize DDNN training in heterogeneous clusters, we use a weighted average (WA) batch size to denote the average data size trained in an iteration. It can be defined as the amount of trained data samples per unit time divided by iterations trained per unit time. Deriving from Eq. (4) in Sec. III, the weight can be considered as the reciprocal of the worker iteration time. Accordingly, we define a normalized iteration as training a WA batch of data samples in a heterogeneous cluster. In particular, the WA batch size in a homogeneous cluster (i.e., a special case of a heterogeneous cluster) is actually reduced to the batch size of a worker. As shown in Fig. 3(a), the DDNN training loss converges faster as the WA batch size increases. This is because a larger WA batch size generates a more accurate training gradient, achieving the objective training loss value with fewer iterations [20]. Accordingly, we can leverage the WA batch size to quantify the DDNN training loss in heterogeneous clusters.

Interestingly, we find that the cluster heterogeneity can impact the training loss by varying the numbers of p3.2xlarge and p2.xlarge instances in a 10-worker heterogeneous cluster, as illustrated in Fig. 3(b). Since workers can miss more

TABLE I: Comparison of the training time, monetary cost, and computation overhead of ResNet-110 with different instance provisioning strategies.

Strategies	Provisioning plans	Time (secs)	Cost / Overhead	
			Money (\$)	Comp (secs)
On-demand	5g3.8xl, 3g3.16xl	2; 549:02	17:75	0:02
CM-DARE <sup>+</sup>	4g3.8xl, 3g3.16xl	2; 710:85	5:16	0:02
Srifty <sup>+</sup>	2g4dn.4xl, 5g3.8xl, 2g3.16xl	2; 597:34	4:97	0:21
spotDNN	5g3.8xl, 3g3.16xl	2; 549:02	5:33	0:02

fresh parameter updates in the ASP mechanism as the cluster scale increases [21], the local training with a single worker achieves the optimal loss convergence rate. To characterize the performance impact of cluster heterogeneity, we define a convergence coefficient (CC) as the Euclidean distance between the parameter update vector  $W = [w^1; w^2; \dots; w^{N_i}]$  in a heterogeneous cluster  $N$  and the local training  $W_{\text{local}} = [1; 0; \dots; 0]$ . In particular, we calculate the percentage of parameter updates  $w^i$  over the training process for each worker  $i \in [1; N]$ , which denotes the convergence contribution of the worker to the model training. Accordingly, a smaller CC indicates a more localized model training where the training process is mainly distributed among fewer workers, thereby speeding up the convergence of training loss.

#### C. A Motivation Example

Blindly configuring heterogeneous clusters can either under-provision or over-provision instance resources, which can adversely affect the training performance. In response, 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 ResNet-110 to illustrate its effectiveness. Specifically, we adopt g4dn.4xlarge, g3.8xlarge, g3.16xlarge, and p2.8xlarge instances each with a user quota of 5. The objective training loss is set as 0:8 and the objective training time is set as 2; 600 seconds.

As shown in Table I, the CM-DARE<sup>+</sup> strategy, which combines the performance model of CM-DARE [8] with the spotDNN instance provisioning plan, produces a provisioning plan that exceeds the training objective time by 110:85 seconds. This is because CM-DARE<sup>+</sup> neglects the bottlenecks on PS network and PCIe bandwidth resources during the training process. It then overestimates the training performance and under-provisions spot instances. Furthermore, the Srifty<sup>+</sup> strategy, which combines the performance model of spotDNN and instance provisioning strategy (i.e., exhaustive search with a high complexity) of Srifty [14], configures 9 workers with the lowest budget but 10 times more computation overhead (i.e., 0:21 seconds) than spotDNN. In contrast, spotDNN meets the training performance SLOs and saves the monetary cost by up to 69:9% 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 even in a heterogeneous cluster with the ASP mech-

TABLE II: Key notations in our analytical performance model of DDNN training in heterogeneous clusters.

Notation	Definition
$N$	Set of provisioned heterogeneous workers
$j$	Number of normalized iterations for a DNN model
$T^i$	Iteration time of a worker $i$
$T_{exp}$	Expected iteration time of a heterogeneous cluster
$b_w$	WA batch size of a heterogeneous cluster
$v$	Training speed of a heterogeneous cluster
$R$	CC of a heterogeneous cluster
$T_{comm}^i$	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 $i$
$B_{wk}^i$	Available network bandwidth between a worker $i$ and PS
$B_{pcie}$	Available PCIe bandwidth in a worker
$B_{ps}$	Available bandwidth of a PS node

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

### III. MODELING DDNN TRAINING PERFORMANCE IN HETEROGENEOUS CLUSTERS

In this section, we first model the training loss using the WA batch size and the configuration of heterogeneous clusters (i.e., the CC and the number of workers). We then predict the DDNN training time by explicitly considering the PS network and PCIe bandwidth resource bottlenecks. The notations in our performance model are summarized in Table II.

In general, the iteration time of DDNN training with the ASP mechanism 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 an expected iteration time  $T_{exp}$ . Therefore, we formulate the DDNN training time  $T$  as

$$T = j \cdot T_{exp}; \quad (1)$$

where  $T_{exp}$  is considered as the expectation of the iteration time of heterogeneous workers. Accordingly,  $T_{exp}$  can be formulated as the reciprocal of the number of iterations per unit time in a heterogeneous cluster, which is given by

$$T_{exp} = \frac{1}{\sum_{i \in N} \frac{1}{T^i}}; \quad (2)$$

where  $N$  is the set of provisioned workers, and  $\frac{1}{T^i}$  denotes the number of iterations per unit time on a worker  $i$ .

**Modeling DDNN Training Loss.** As discussed in Sec. II-B, the DDNN training loss converges faster as the WA batch size  $b_w$  gets larger and the CC  $R$  gets smaller. The convergence rate slows down as more workers are provisioned. Moreover, the DDNN training loss is inversely proportional to the normalized iterations  $j$  and converges at a rate of  $O(\frac{1}{j})$  [22]. According

to our motivation experiment in Fig. 3, we empirically fit the training loss in a heterogeneous cluster as

$$f_{loss}(b_w; R; N; j) = \frac{(2b_w + 3) \cdot (\frac{1}{R} + 4) \cdot j \cdot N \cdot j}{1 + \dots} + 5; \quad (3)$$

where  $1; \dots; 5$  are the model coefficients,  $1$  and  $2 < 0$ .

We continue to model the WA batch size  $b_w$ . As defined in Sec. II-B, the amount of data samples trained per unit time is considered as the cluster training speed (i.e.,  $v$ ). The number of iterations trained per unit time can be identified as the reciprocal of the expected iteration time  $T_{exp}$ , according to Eq. (2). Accordingly,  $b_w$  is formulated as

$$b_w = \frac{v}{T_{exp}} = v \cdot T_{exp}; \quad (4)$$

As workers communicate with the PS without a synchronization barrier under the ASP mechanism, we can calculate  $v = \frac{1}{\sum_{i \in N} T^i} \cdot v^i = \frac{1}{\sum_{i \in N} \frac{b^i}{T^i}}$ , where  $v^i$ ,  $b^i$ , and  $T^i$  denote the training speed, batch size, and iteration time of a worker  $i$ , respectively. According to the definition in Sec. II-B, the CC  $R$  can be formulated as

$$R = \frac{1}{\sum_{i \in N} (w^i - 1)^2 + (w^i - 0)^2 + \dots + (w^i - N)^2}; \quad (5)$$

where  $w^i = \frac{v^i}{v}$  denotes the percentage of parameter updates for a worker  $i$ .

**Modeling Iteration Time of a Worker.** Each iteration of DDNN training is divided into two phases: gradient computation and parameter communication. They are processed sequentially under the ASP mechanism. Accordingly, we formulate the iteration time  $T^i$  of a worker  $i$  as

$$T^i = T_{comm}^i + T_{comp}^i; \quad (6)$$

where  $T_{comm}^i$  and  $T_{comp}^i$  denote the communication time and GPU computation time in an iteration of worker  $i$ , respectively. In particular, the GPU computation time  $T_{comp}^i$  is determined by the GPU type and batch size. Accordingly, we consider it as a constant value as long as the instance GPU types stay the same due to the same batch size for identical GPUs. We can calculate it as  $T^i - T_{comm}^i$  by the 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  $T_{comm}^i$  of a worker  $i$  can be calculated as

$$T_{comm}^i = \frac{2 \cdot S_{parm}}{B_{wk}^i} + \frac{2 \cdot g^i \cdot S_{parm}}{B_{pcie}}; \quad (7)$$

where  $B_{wk}^i$  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 analyzed in Sec. II-B,  $B_{wk}^i$  is restricted by the PS network bandwidth  $B_{ps}$ , which becomes a resource bottleneck as more workers are provisioned. As shown in Fig. 4, the



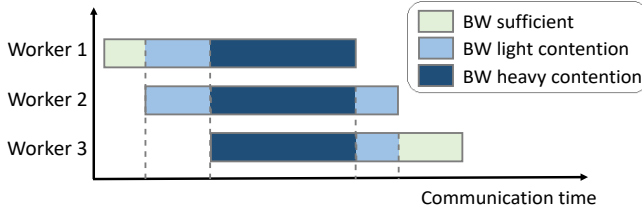


Fig. 4: Resource contention of PS network bandwidth (BW) over time as the communication between the worker and PS starts or ends.

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

$$B_{wk}^i = \begin{cases} P \frac{B_{ps}}{jNj} + (1-P) B_{req} & B_{req} > \frac{B_{ps}}{jNj}; \\ B_{req} & B_{req} < \frac{B_{ps}}{jNj}; \end{cases} \quad (8)$$

where  $B_{req}$  denotes the network bandwidth requirement between a worker and PS when there is only one worker in the cluster. Besides,  $P \in [0, 1]$  denotes the probability of PS network bandwidth bottleneck, which is positively correlated with the number of provisioned workers. We formulate it as

$$P = \min \left( \frac{1}{jNj}, 1 \right); \quad (9)$$

where  $\alpha$  and  $\beta$  are model coefficients. In particular,  $P = 1$  when the bandwidth competition reaches its peak.

**Identifying Batch Sizes for Heterogeneous GPUs.** To fully utilize the GPU resource, we assign a GPU of type  $k$  with its largest batch size  $b^k$ , which is usually a power of 2 and is limited by the GPU memory size. The relationship between  $b^k$  and the GPU memory usage  $mem^k$  can be formulated as

$$b^k = 2^{\lfloor \log_2(2 \cdot mem^k + \alpha) \rfloor}; \quad (10)$$

where  $\alpha$  and  $\beta$  are model coefficients that can be obtained by workload profiling. We acquire  $b^k$  by setting  $mem^k$  as the GPU memory size. Accordingly, the batch size of a worker  $i$  with  $g^i$  GPUs of type  $k$  can be calculated as  $b^i = g^i b^k$ .

**Obtaining Model Parameters.** Based on the model above, we have 5 workload-specific parameters (i.e.,  $S_{parm}$ ,  $B_{req}$ ,  $i$ ,  $i$ ,  $i$ ) and 4 instance-specific parameters (i.e.,  $T_{comp}$ ,  $g^i$ ,  $B_{pcie}$ ,  $B_{ps}$ ). Specifically, the number of GPUs  $g$  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 file of cloud providers. Then, we fit the model coefficients  $\alpha$  and  $\beta$  by running the DNN model on a single worker with 30 iterations and recording the GPU memory usage under different batch sizes. Meanwhile, the model parameter size  $S_{parm}$  can be measured as the PS network data traffic divided by the number of iterations. The required network bandwidth  $B_{req}$  of workers can be obtained using the nethogs tool. To fast acquire the iteration time of different GPUs and the model coefficients (i.e.,  $\alpha$ ,  $\beta$ , and  $i$ ), we perform 3 epochs on three heterogeneous clusters with various numbers of workers (i.e., 2, 4, 7) in parallel using the regression method [23]. With the obtained parameters above, the computation time  $T^i$  of different GPUs can be calculated by Eqs. (6)-(7) accordingly.

#### IV. GUARANTEEING DDNN TRAINING PERFORMANCE WITH CLOUD SPOT INSTANCES

In this section, we first formulate the provisioning optimization problem of heterogeneous spot instances. We then design and implement our spotDNN instance provisioning strategy to provide predictable DDNN training performance.

##### A. Optimizing Spot Instance Provisioning

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

$$\min_N \quad C = T \sum_{m \in M} n_m p_m \quad (11)$$

$$\text{s.t.} \quad f_{loss}(b_w; R; N; j) = L_{obj}; \quad (12)$$

$$T \leq T_{obj}; \quad (13)$$

$$n_m \leq Lim_m; \quad \forall m \in M; n_m \in \mathbb{Z} \quad (14)$$

where  $n_m$  and  $p_m$  denote the provisioned number and the unit price of an instance type  $m$ , respectively. The output  $N$  is a list of the provisioned instances with a length of  $\sum_{m \in M} n_m$ . The remaining objective training time  $T_{obj}$  requires re-calculation when instance revocations occur. Constraints (12) and (13) guarantee the training performance SLOs in terms of  $L_{obj}$  and  $T_{obj}$ . Constraint (14) denotes that the number of provisioned instances is below the user quota for each type  $m$ .

**Problem Analysis.** According to Eq. (11), the monetary cost  $C$  is actually impacted by the number of provisioned workers  $jNj$  and the training time  $T$ . By substituting Eqs. (2)-(10) into Constraint (13), we conclude that the constraint on the training time  $T$  is non-linear. As a result, our optimization problem in Eq. (11) turns out to be a non-linear integer programming, which is an NP-hard problem [24]. Accordingly, we turn to designing a heuristic algorithm to solve our instance provisioning problem.

To narrow down the solution search space, we design two optimization techniques. First is to analyze the lower bound  $n_{lower}$  and upper bound  $n_{upper}$  of the number of provisioned workers  $jNj$ . As evidenced in Sec. II-B, the PS network bandwidth contention negatively impacts the training speed as more workers are provisioned. To avoid severe performance degradation, we let  $P < 1$ . According to Eq. (9), we calculate the upper bound  $n_{upper}$  as

$$n_{upper} = \frac{1}{1 - \frac{1}{2^{\beta}}}; \quad (15)$$

To meet the objective loss (i.e., Constraint (12)), we can infer from Eq. (3) that the minimum number of normalized iterations is  $\frac{(2b_{min} + 3) \cdot 4^{\beta}}{L_{obj} \cdot 5}$ , where  $b_{min} = \min_{i \in N} b^i$  is the minimum batch size of all available instances. To meet the

---

**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  $L_{obj}$ ) of a DNN model, a set of available instance types  $M$ , user quotas  $Lim$ , and a list of existing instances  $E$ .  
Output: Instance provisioning plan  $N$  and the monetary cost  $C$ .

- 1: Acquire instance-specific parameters (i.e.,  $m_p^k, g^i, B_{ps}^i, B_{ps}^i$ ) and workload-specific parameters (i.e.,  $S_{parm}, B_{req}, r_i$ ) by lightweight workload profiling;
- 2: Initialize:  $C \leftarrow 1; N \leftarrow \emptyset$ ; Put all the available instances into the list  $I$ ;
- 3: Calculate the batch size  $b^k$  Eq: (10) and the computation time  $T_{comp}^k$  Eq: (6) of GPU type  $k$  on a single-GPU instance;
- 4: Calculate the upper bound  $n_{upper}$  Eq: (15), the lower bound  $n_{lower}$  Eq: (16), and the iteration time  $T^i$  Eq: (6) of a worker  $i$  with the required network bandwidth  $B_{req}$ ;
- 5: Sort the available instances list  $I$  in descending order by  $T$ ;
- 6: for all  $n \in [n_{lower}, n_{upper}]$  do
- 7:   for all  $idx \in [0; j|j|n]$  do
- 8:     Acquire  $N = E + I[idx; idx + n - 1]$  by sequentially selecting  $n$  instances from the  $idx$ -th instance in the available instances list  $I$ ; // sliding window
- 9:     Calculate the WA batch size  $b_w$  Eq: (4), the CC  $R$  Eq: (5), and the number of normalized iterations  $j$  to meet the objective loss value  $f_{loss}(b_w; R; N; j) = L_{obj}$ ;
- 10:     Calculate  $P$  Eq: (1),  $C$  Eq: (11);
- 11:     if a revocation occurs then
- 12:        $P \leftarrow P + T_{ohd}$ ; // add revocation overhead
- 13:     end if
- 14:     if  $P \leq T_{obj}$  &  $C < C$  then
- 15:       Record the monetary cost  $C$  and the instance provisioning plan  $N$ ;
- 16:     end if
- 17:   end for
- 18: end for

---

objective training time (i.e., Constraint (13)), we can calculate the lower bound  $n_{lower}$  as

$$n_{lower} = \frac{6}{6} \frac{b_{min}^{\frac{(2b_{min}+3)}{5}} \frac{4}{5} \frac{3}{7}}{v_{max} T_{obj}} \frac{1}{7}; \quad (16)$$

where  $v_{max} = \max_{i \in N} \frac{b^i}{T^i}$  is the maximum training speed of the available instances. Second is to provision a set of spot instances with small differences in iteration time. We make the decision because such workers introduce a lower degree of gradient staleness [21], which makes the training loss converge more steadily. Accordingly, we store all the available instances in a list by sorting their iteration time in descending order. We then adopt a sliding window to select the adjacent instances with small differences in iteration time in the sorted list.

### B. Design of spotDNN Instance Provisioning Strategy

Given a DNN model with the performance SLOs, a set of available instance types  $M$ , the user quotas  $Lim$ , and a list of existing instances  $E$ , spotDNN first obtain the model parameters using the profiling method elaborated in Sec. III (line 1).

After initializing several algorithm variables, spotDNN then calculates the batch size  $b^k$  and the computation time  $T_{comp}^k$  for a GPU type  $k$ , as well as the upper and lower bounds of the number of provisioned workers (lines 2-4). By iterating each possible number  $n$  of provisioned workers, spotDNN leverages a sliding window to select the possible provisioning plans  $N$  (lines 5-8). It further calculates the number of normalized iterations  $j$  to meet the objective training loss value  $L_{obj}$  and then obtains the estimated training time and monetary cost (lines 9-10). In particular, spotDNN adds an instance revocation overhead  $T_{ohd}$  (e.g., less than 30 seconds) to the estimated training time  $P$  when a revocation occurs (lines 11-13). Finally, spotDNN identifies a cost-efficient instance provisioning plan  $N$  that guarantees the objective time  $T_{obj}$  and minimizes the monetary cost  $C$  (lines 14-18).

**Remark.** The complexity of Alg. 1 is in the order of  $O(xy)$ , where  $x = n_{upper} - n_{lower} + 1$  denotes the possible search space of the number of provisioned workers  $j|N|$ , and  $y = |I| - n_i + 1$  denotes the candidate provisioning

under each possible number  $n_i$ . As a result, the computation overhead of spotDNN is well controlled and will be validated in Sec. V-D. In addition, the user quotas  $Lim$  can be adjusted online according to the availability of spot instances over time.

### C. Implementation of our spotDNN Prototype

We implement a prototype of spotDNN on Amazon EC2 based on TensorFlow [25] v1.15.0 with over 1,000 lines of Python and Linux Shell codes, which are publicly available on GitHub. To avoid network traffic across Availability Zones, we place all PS and worker instances within one Amazon Virtual Private Cloud (VPC). We use the AWS CLI command to provision spot instances, particularly setting the subnet ID according to the VPC information. To timely capture revocation events, our revocation detector periodically checks the instance status (e.g., every 15 seconds) using the Amazon instance metadata service<sup>3</sup>. Once an instance revocation is detected, it first sends the current model training status (i.e., the remaining instances and unfinished performance SLOs) to the training performance predictor. Then, spotDNN re-predicts the performance and launches takeover spot instances to guarantee the training performance. To achieve dynamic worker addition

without interrupting the training process, spotDNN leverages sparse mapping [7] to store all the available IPv4 addresses in the VPC. It configures the newly added workers within the same VPC to enable efficient network communication. In particular, spotDNN can support other cloud platforms (e.g., Google GCP, Microsoft Azure) by simply substituting AWS-related commands and metadata service APIs.

**Discussion.** First, how does spotDNN deal with the prediction error of our performance model? To mitigate the impact of SLO violations, spotDNN overpredicts the network contention in Eq. (9) by an empirical value (e.g., 1-5%), which is the fluctuating range of network bandwidth  $B_{wk}^i$  based on

<sup>3</sup><https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2-instance-metadata.html>

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

Instance type	GPU devices	Max quota	Unit price (\$)	
			spot <sup>4</sup>	on-demand
p3.2xlarge	1 V100	10	1:25	3:06
p2.8xlarge	8 K80	10	2:16	7:20
g4dn.12xlarge	4 T4	10	1:22	3:91
g4dn.4xlarge	1 T4	10	0:36	1:20
g3.16xlarge	4 M60	10	1:37	4:56
g3.8xlarge	2 M60	10	0:69	2:28
g3.4xlarge	1 M60	10	0:34	1:14

our experiments in Sec. II-B. It can cause an underestimation of the cluster training speed, making the training process faster than expected. As evidenced in Fig. 5, our bandwidth overprediction can basically offset the prediction error of the performance model. Second, how to use spotDNN in multi-PS scenarios? spotDNN is applicable in multi-PS scenarios. Though adding more PS can alleviate the PS network bottleneck, the bottleneck still remains as it is intrinsic to the PS architecture. By simply replacing  $B_{ps}$  with the total network bandwidth that multi-PS provides, our performance model can readily be applied to multi-PS scenarios.

## V. PERFORMANCE EVALUATION

In this section, we evaluate spotDNN by carrying out a set of prototype experiments on Amazon EC2 [6]. We seek to answer the following questions:

**Accuracy:** Can our performance model in spotDNN predict the DDNN training performance (i.e., time and loss) in heterogeneous clusters? (Sec. V-B)

**Effectiveness:** Can our instance provisioning strategy in spotDNN provide predictable training performance while saving the 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 deploy an m5.xlarge on-demand instance to serve as the PS. We provision heterogeneous workers with 7 representative instance types listed in Table III. To reduce the network traffic cost, we configure the provisioned instances within a VPC in the AWS us-east-1b region. In addition, the available bandwidth for the PS and the available PCIe bandwidth inside a worker is set as 1:2 GBps 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 only adopt CIFAR-100 [17] as the training dataset for VGG-19 [18], Inception-v3 [26], and ResNet-110 [16] models for image classification. We also choose an NLP DNN model (i.e., EsperBERTo [27]) trained on the Esperanto portion of the OSCAR dataset<sup>5</sup>.

<sup>4</sup>We list the spot price during the period of our experiments (Jan. 2023).

<sup>5</sup><https://oscar-corpus.com/>

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

Workload	ResNet-110	EsperBERTo	VGG-19	Inception-v3
Dataset	CIFAR-100	OSCAR	CIFAR-100	CIFAR-100
$S_{par}$ (MB)	11:54	336	574	98
$B_{req}$ (MBps)	142	543	620	562

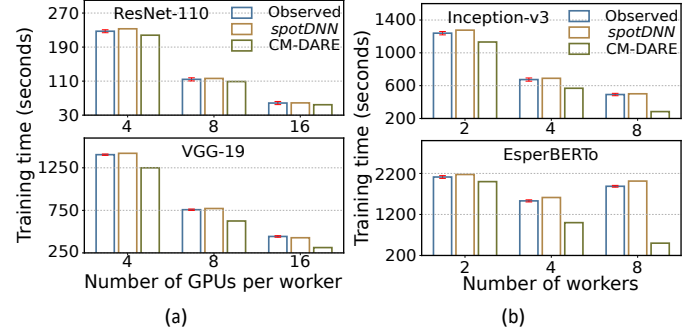


Fig. 5: Comparison of the observed and predicted training time with spotDNN and CM-DARE by using (a) a P2 instance by varying the number of GPUs per instance from 4 to 16, and (b) n-worker heterogeneous clusters consisting of  $\frac{n}{2}$  g4dn.4xlarge and  $\frac{n}{2}$  g3.16xlarge workers.

**Baselines and Metrics.** We compare spotDNN with the three strategies as discussed in Sec. II-C. In brief, CM-DARE<sup>+</sup> considers the cluster speed as the sum of ideal GPU training speeds [8]. Sfrity<sup>+</sup> can be considered as the optimal solution based on the exhaustive search [14]. We focus on three key metrics including the DDNN training time, the monetary cost, and the algorithm computation overhead.

### B. Validating Training Performance Model in spotDNN

Can spotDNN well predict the DDNN training time? We train the ResNet-110, VGG-19, Inception-v3, and EsperBERTo models for 10, 5, 6, and 1:5 epochs, respectively. As shown in Fig. 5(a), we observe that spotDNN can well predict the DDNN training time on a GPU instance, with a prediction error from 0:2% to 2:5%. In contrast, CM-DARE poorly predicts the DDNN training time as it overlooks the performance impact of PCIe bandwidth bottleneck during the gradient aggregation. The prediction error of the VGG-19 achieved by CM-DARE gets larger (i.e., from 13:7% to 28:5%) as the number of GPUs on a worker increases from 4 to 16. This is because the PCIe bandwidth contention within a worker increases the gradient aggregation time, thereby causing a non-linear speedup of the training speed.

Furthermore, Fig. 5(b) shows that spotDNN can accurately predict the DDNN training time in a heterogeneous cluster with a prediction error of 1:8% to 6:9% as the cluster scale varies from 2 to 8. However, CM-DARE fails to predict the training time with a prediction error reaching up to 73:5%. The rationale is that the contention of PS network bandwidth among workers gets severe as the number of provisioned workers and the parameter size increase, which is ignored by CM-DARE and thus leads to inaccurate training time prediction. As the number of workers increases from 4 to 8, the PS network bandwidth contention among workers dominates

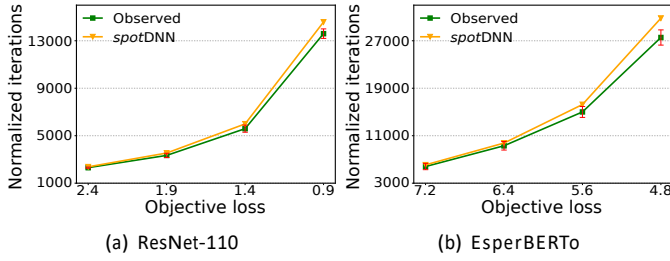


Fig. 6: Comparison of the observed and predicted number of normalized iterations under different objective loss values with a heterogeneous cluster consisting of 4 g4dn.4xlarge and 4 g3.16xlarge workers.

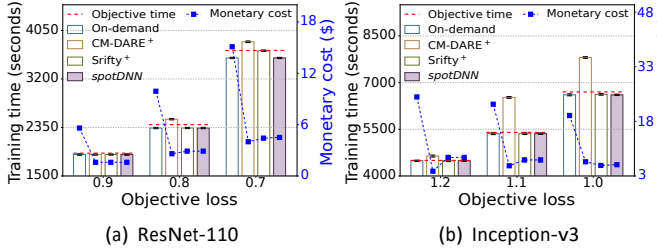


Fig. 7: Comparison of DDNN training time and monetary cost of various instance provisioning strategies under different performance SLOs.

the training process of EsperBERTo, thereby increasing the DDNN training time.

Can spotDNN well predict the DDNN training loss? We further evaluate the accuracy of our DDNN training loss model in an 8-worker heterogeneous cluster by setting the objective loss value as 2:4, 1:9, 1:4, 0:9 for ResNet-110 and 7:2, 6:4, 5:6, 4:8 for EsperBERTo, respectively. As shown in Fig. 6, we observe that spotDNN can basically predict the number of normalized iterations to converge to different objective loss values. Specifically, spotDNN achieves a prediction error of 4:8% – 11:7% for EsperBERTo, which is higher than that of ResNet-110 (i.e., 3:1% – 7:1%). This is because large models (i.e., EsperBERTo) consume more bandwidth resources to transfer gradients in the communication phase. It leads to severe contention of the PS network bandwidth and PCIe bandwidth, making the training performance fluctuate moderately (i.e., up to 1;261 normalized iterations).

### C. Effectiveness of spotDNN Instance Provisioning Strategy

Can spotDNN guarantee the DDNN training performance while minimizing the monetary cost? To examine the efficacy of our instance provision strategy, we set three different training performance SLOs listed in Table V for ResNet-110 and Inception-v3. As shown in Fig. 7, we observe that spotDNN can well meet the objective training time under different objective loss values, while saving the monetary cost by up to 68:1% compared with the three strategies. Specifically, CM-DARE+ can hardly provide predictable training performance for DNN models, because it neglects the bottlenecks on PS network and PCIe bandwidth resources. In contrast, both spotDNN and Srifty+ can guarantee training performance while reducing the budget. In most cases, spotDNN can obtain the optimal resource provisioning plans with Srifty+, as listed

TABLE V: Comparison of instance provisioning plans achieved by on-demand, CM-DARE+, Srifty+, and spotDNN for ResNet-110 and Inception-v3. The provisioning plan in the array is defined as [#p3.2xl, #g4dn.4xl, #g3.8xl, #g3.16xl, #p2.8xl].

Workloads/ Strategies	Performance SLOs (loss value, training time)		
	(0:9, 1900)	(0:8, 2400)	(0:7, 3700)
ResNet-110	On-demand	[2, 0, 2, 0, 0]	[2, 0, 4, 0, 0]
	CM-DARE+	[2, 0, 2, 0, 0]	[2, 0, 3, 0, 0]
	Srifty+	[2, 0, 2, 0, 0]	[2, 1, 3, 0, 0]
	spotDNN	[2, 0, 2, 0, 0]	[2, 1, 4, 0, 0]
		(1:2, 4500)	(1:1, 5400)
Inception-v3	On-demand	[2, 0, 0, 3, 0]	[2, 0, 0, 2, 0]
	CM-DARE+	[2, 0, 0, 1, 0]	[0, 5, 2, 0, 0]
	Srifty+	[2, 0, 0, 3, 0]	[2, 0, 2, 0, 0]
	spotDNN	[2, 0, 0, 3, 0]	[2, 0, 1, 0, 0]
		(1:0, 6700)	

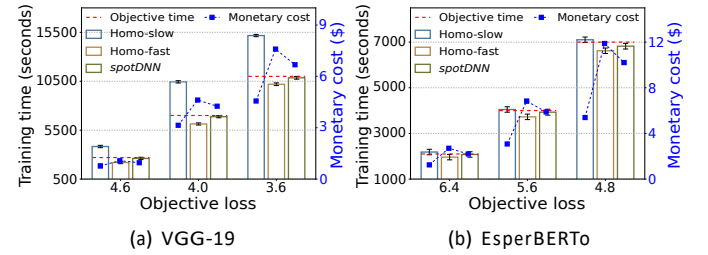


Fig. 8: Comparison of DDNN training time and monetary cost achieved by spotDNN and two homogeneous instance provisioning strategies.

in Table V. The rationale is that spotDNN simply uses a sliding window to capture the majority of near-optimal solutions, as discussed in Sec. IV-A. Furthermore, spotDNN can generate the instance provisioning plans much faster than Srifty+ by 84:6% (i.e., 0:2 seconds vs. 1:3 seconds) when the user quota is set as 25. We will elaborate the computation overhead of spotDNN in a large cluster in Sec. V-D.

Can spotDNN outperform homogeneous instance provisioning strategies? To illustrate the training efficiency of heterogeneous clusters, we evaluate spotDNN against two homogeneous instance provisioning strategies (i.e., Homo-slow, Homo-fast). Homo-slow provisions a cluster of instances with the slowest training speed, while Homo-fast provisions instances with the fastest training speed in the spotDNN provisioning plan. Both strategies provision the same number of workers as spotDNN. By setting three training performance SLOs listed in Table VI, we observe from Fig. 8 that spotDNN outperforms both Homo-slow and Homo-fast strategies. Specifically, Homo-slow violates the time constraint due to the inefficiency of slower instances and the longer training time required by the clusters with larger CCs. Though Homo-fast can guarantee the performance SLOs as spotDNN, its monetary cost increases by up to 18:5% due to the high unit price of instances. In contrast, spotDNN configures heterogeneous clusters with smaller CCs than homogeneous strategies. It ensures fast convergence even with several slower instances while saving the training budget.



TABLE VI: Comparison of instance provisioning plans and the corresponding CCs achieved by spotDNN, Homo-slow, and Homo-fast strategies for VGG-19 and EsperBERTo. The provisioning plan in the array is defined the same as Table V.

Workloads/ Strategies	Performance SLOs (loss value, training time)		
	(4:6, 2700)	(4:0, 7000)	(3:6, 11000)
VGG-19	Homo-slow	[0, 2, 0, 0, 0] / 0:71	[0, 3, 0, 0, 0] / 0:82
	Homo-fast	[2, 0, 0, 0, 0] / 0:71	[3, 0, 0, 0, 0] / 0:82
	spotDNN	[1, 1, 0, 0, 0] / 0:51	[2, 1, 0, 0, 0] / 0:75
	(6:4, 2100)	(5:6, 4000)	(4:8, 7000)
EsperBERTo	Homo-slow	[0, 0, 3, 0, 0] / 0:82	[0, 0, 4, 0, 0] / 0:87
	Homo-fast	[0, 0, 0, 3, 0] / 0:82	[0, 0, 0, 4, 0] / 0:87
	spotDNN	[0, 0, 1, 2, 0] / 0:59	[0, 0, 1, 3, 0] / 0:83

Can spotDNN guarantee DDNN training performance even when revocations occur? As shown in Fig. 9, spotDNN starts the training of ResNet-110 by provisioning 3 g3.16xlarge and 5 g3.8xlarge instances. At the 29:8-th minute, the revocation detector in spotDNN captures the revocation signal, and it identifies 5 g3.8xlarge instances will be revoked. Leveraging the current training status (i.e., remaining 3 g3.16xlarge instances, remaining training time as 810 seconds, and loss value as 0:9), the performance predictor further works with the instance provisioner in spotDNN to generate a new provisioning plan. Specifically, spotDNN launches 3 p3.2xlarge instances to takeover the unfinished model training work. In particular, the 3 newly added workers only spend 2:1 minutes joining the training process, which is slightly longer than the 2-minute notification [6] (i.e., the 5 g3.8xlarge instances are revoked at the 31:4-th minute). Due to the fast failover mechanism of spotDNN, the performance degradation only lasts for 29:6 seconds (i.e., from the 31:4-th to 31:9-th minute), as depicted in Fig. 9. spotDNN successfully achieves the training performance SLOs of ResNet-110 by completing the training process at the 41:9-th minute.

#### D. Runtime Overhead of spotDNN

We evaluate the runtime overhead of spotDNN in terms of the workload profiling overhead and the computation overhead of Alg. 1. Specifically, we launch a p2.xlarge EC2 instance to profile the workload-specific parameters (i.e.,  $S_{param}$ ,  $B_{req}$ ,  $z$ ,  $2$ ). By training the four DDNN models for 30 iterations, the profiling time of ResNet-110 [16], Inception-v3 [26], VGG-19 [18], and EsperBERTo [27] models are merely 114, 75, 141, and 21:9 seconds, respectively. The remaining parameters (i.e.,  $1$ ,  $1$ ,  $i$ ) are profiled in parallel on 3 heterogeneous clusters with 2, 4, 7 workers. The profiling time of the four workloads is 0:6, 5:8, 12:8, and 11:7 minutes, respectively. Such job profiling overhead is negligible compared to the several hours required by a typical DDNN training workload.

To illustrate the computation overhead of spotDNN, we conduct another experiment by provisioning a ResNet-110 model using 70 available instances (i.e., the maximum user quota) in Table III. spotDNN can achieve the optimal provisioning plan as Sfrity<sup>+</sup> with a negligible computation overhead (i.e., 0:3 seconds). In contrast, the computation overhead of Sfrity<sup>+</sup>

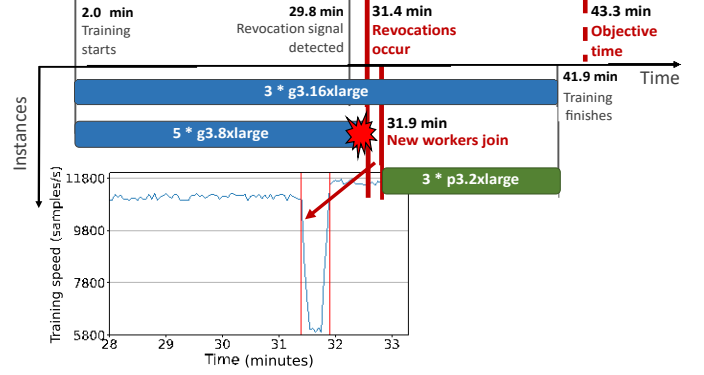


Fig. 9: Timeline of spotDNN dealing with multiple instance revocations, during the training process of ResNet-110 with an objective loss of 0:8 and an objective time of 2; 600 seconds.

increases to 1; 503:7 seconds, due to the exhaustive search with the complexity in the order of  $O(\sum_{i=1}^M \text{Lim}_i)$ . Accordingly, spotDNN can be applicable to large-scale clusters and its runtime overhead is practically acceptable.

## VI. RELATED WORK

**Resource Provisioning of DDNN Training.** To improve the DDNN training performance, Proteus [28] scales the cluster resources with on-demand and spot instances. FC<sup>2</sup> [29] aims to save the budget by considering the capacity and utilization of network bandwidth during resource provisioning. DNN [30] and Cynthia [1] provision homogeneous workers with serverless functions and EC2 instances, respectively. However, we focus on the heterogeneous clusters while handling spot instance revocations without interruption. A more recent work Sfrity [14] leverages the exhaustive search method to provision heterogeneous spot instances for DDNN training with the AllReduce architecture. In contrast, spotDNN designs a heuristic algorithm to dramatically reduce the computation overhead as evidenced in Sec. V-D. Also, spotDNN constructs an analytical performance model to predict training speed and loss, while Sfrity relies on regression models which highly depend on the quality of training data and model features.

**Performance Modeling of DDNN Training.** Several works focus on modeling the training performance of homogeneous clusters, such as predicting the training time of DDNN operations [31], simulating the interaction between the PS and workers [32], and predicting the training loss through online fitting [22]. CM-DARE [8] performs regression fitting on the training speed of heterogeneous clusters without considering the performance degradation caused by bottleneck resources. Jiang et al. [33] qualitatively analyze the impact of cluster heterogeneity on the DDNN training loss. Unlike prior works, spotDNN quantitatively models the training loss by introducing the WA batch size and CC. Moreover, it models the DDNN training time of heterogeneous clusters by explicitly considering the contention of PS network bandwidth and PCIe bandwidth with the ASP mechanism.

**Fault Tolerance of Cloud Spot Instances.** To enhance the spot instance availability, Spotlake [34] builds a random forest model based on historical spot price datasets. To handle spot

instance revocations, checkpointing is a straightforward and intuitive solution [3]. The checkpointing frequency can be optimized through online profiling of checkpointing overhead [4] or building a three-stage revocation probability model [35]. Bamboo [36] adds redundant computations to avoid data loss. Spotnik [37] designs an adaptive collective communication and synchronization mechanism for Ring-AllReduce. Orthogonal to these works above, we focus on the ASP mechanism under the PS architecture, which can intrinsically maintain the DDNN training process even when instance revocations occur. Moreover, we implement a revocation detector to provision takeover instances without interrupting the training process.

## VII. CONCLUSION

This paper presents spotDNN, a heterogeneity-aware spot instance provisioning framework for achieving predictable DDNN training performance in the cloud. By explicitly considering the severe contention for the bottleneck bandwidth resources, spotDNN devises a lightweight analytical performance model for DDNN training in heterogeneous clusters. It leverages the WA batch size and CC to characterize the DDNN training loss in heterogeneous clusters. Such a performance model further guides the design of our cost-efficient spot instance provisioning strategy in spotDNN, which utilizes bounds calculation and sliding window to effectively identify the appropriate provisioning plan. Extensive prototype experiments on AWS EC2 demonstrate that spotDNN can deliver predictable DDNN training performance, while saving the monetary cost by up to 68:1% compared to existing solutions.

## 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] S. M. Iqbal, H. Li, S. Bergsma, I. Beschastnikh, and A. J. Hu, "Cospot: a cooperative vm allocation framework for increased revenue from spot instances," in Proc. of SOCC, Nov. 2022, pp. 540–556.
- [3] 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.
- [4] J. Mohan, A. Phanishayee, and V. Chidambaram, "CheckFreq: Frequent, Fine-Grained DNN Checkpointing," in Proc. of USENIX FAST, Feb. 2021, pp. 203–216.
- [5] 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.
- [6] Amazon. (2023, Feb.) Amazon EC2 Spot Instances. [Online]. Available: <https://aws.amazon.com/cn/ec2/spot/>
- [7] 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.
- [8] 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.
- [9] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in Proc. of NIPS, Dec. 2014, pp. 1–9.
- [10] 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.
- [11] 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.
- [12] A. C. Zhou, J. Lao, Z. Ke, Y. Wang, and R. Mao, "Farspot: Optimizing monetary cost for hpc applications in the cloud spot market," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, pp. 2955–2967, 2021.
- [13] H. Jiang, X. Zhang, and C. Joe-Wong, "Doll: Distributed online learning using preemptible cloud instances," *ACM SIGMETRICS Performance Evaluation Review*, vol. 50, no. 2, pp. 21–23, 2022.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Technical Report, 2009.
- [18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in Proc. of ICLR, May 2015, pp. 1–14.
- [19] 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.
- [20] 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.
- [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. Peng, 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 and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
- [25] 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.
- [26] 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.
- [27] J. Chaumond. (2020, Feb.) How to train a new language model from scratch using Transformers and Tokenizers. [Online]. Available: <https://huggingface.co/blog/how-to-train>
- [28] 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.
- [29] N. B. D. Ta, "FC<sup>2</sup>: cloud-based cluster provisioning for distributed machine learning," *Cluster Computing*, vol. 22, no. 4, pp. 1299–1315, 2019.
- [30] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "DNN: Achieving Predictable Distributed DNN Training With Serverless Architectures," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 450–463, 2022.
- [31] 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.
- [32] 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.
- [33] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in Proc. of ACM SIGMOD, May 2017, pp. 463–478.
- [34] S. Lee, J. Hwang, and K. Lee, "Spotlake: Diverse spot instance dataset archive service," in Proc. of IISWC, Nov. 2022, pp. 242–255.
- [35] 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.
- [36] J. Thorpe, P. Zhao, J. Elyofson, Y. Qiao, Z. Jia, M. Zhang, R. Netravali, and G. H. Xu, "Bamboo: Making preemptible instances resilient for affordable training of large dnn," arXiv preprint arXiv:2204.12013, 2022.
- [37] 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.