Cynthia: Cost-Efficient Cloud Resource Provisioning for Predictable Distributed Deep Neural Network Training

Haoyue Zheng*, Fei Xu*, Li Chen†, Zhi Zhou‡, Fangming Liu§
*Shanghai Key Laboratory of Multidimensional Information Processing,
Department of Computer Science and Technology, East China Normal University.
†Department of Computer Science, University of Louisiana at Lafayette.
‡School of Data and Computer Science, Sun Yat-sen University.
§School of Computer Science and Technology, Huazhong University of Science and Technology.
*fxu@cs.ecnu.edu.cn, †li.chen@louisiana.edu, ‡zhouzhi9@mail.sysu.edu.cn, §fmliu@hust.edu.cn

ABSTRACT

It becomes an increasingly popular trend for deep neural networks with large-scale datasets to be trained in a distributed manner in the cloud. However, widely known as resource-intensive and time-consuming, distributed deep neural network (DDNN) training suffers from *unpredictable performance* in the cloud, due to the intricate factors of resource bottleneck, heterogeneity and the imbalance of computation and communication which eventually cause severe resource under-utilization. In this paper, we propose *Cynthia*, a cost-efficient cloud resource provisioning framework to provide predictable DDNN training performance and reduce the training budget. To explicitly explore the resource bottleneck and heterogeneity, *Cynthia* predicts the DDNN training time by leveraging a lightweight analytical performance model based on the resource consumption of workers and parameter servers. With an accurate performance prediction, *Cynthia* is able to optimally provision the cost-efficient cloud instances to jointly guarantee the training performance and minimize the training budget. We implement *Cynthia* on top of Kubernetes by launching a 56-docker cluster to train four representative DNN models. Extensive prototype experiments on Amazon EC2 demonstrate that *Cynthia* can provide predictable training performance while reducing the monetary cost for DDNN workloads by up to 50.6%, in comparison to state-of-the-art resource provisioning strategies, yet with acceptable runtime overhead.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; • **Computer systems organization → Cloud computing**.

KEYWORDS

cloud resource provisioning, deep neural network training, predictable performance, resource bottleneck, resource heterogeneity

ACM Reference Format:
Haoyue Zheng*, Fei Xu*, Li Chen†, Zhi Zhou‡, Fangming Liu§. 2019. *Cynthia:* Cost-Efficient Cloud Resource Provisioning for Predictable Distributed Deep Neural Network Training. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3337821.3337873

1 INTRODUCTION

Distributed Deep Learning (DDL) [16] has gained increasing popularity in both industry and academia over the past few years, as it has played a significant role in diverse areas ranging from image and speech recognition to autonomous driving [5]. Distributed Deep Neural Network (DDNN) [11], which is the core of DDL, is increasingly trained in the cloud in order to save training efforts and budget, as the datasets get larger in size and DNN models increase in complexity. The DDL frameworks such as Tensorflow [1] and MXNet [9] are designed to exploit parallel training (*e.g.,* data parallelism, model parallelism) using a cluster of workers. To facilitate DDNN training in clouds, large cloud providers like Amazon and Google have recently launched AWS Deep Learning [25] and Cloud Deep Learning [22], respectively, as reported by Nucleus Research that over 85% of Tensorflow projects running in the cloud are deployed on AWS [2].

Unfortunately, deploying DDNN training workloads in public clouds can suffer from *unpredictable performance* [12], which largely originates from three factors: (a) *Resource bottleneck.* The parameter server (PS) can easily become the CPU and network bandwidth hotspots [19] due to the heavy parameter synchronizations. (b) *Resource heterogeneity.* The hardware heterogeneity [30] among different instance types can slow down the DDNN training process during parameter synchronization [20]. (c) *Imbalance between computation and communication.* Provisioning more workers can adversely impact the DDNN training performance due to the prolonged communication overhead [18]. The underlying rationale is that the three factors above can under-utilize the CPU and network bandwidth resources of workers, leading to the *cost-inefficient* cloud resource provisioning. As evidenced by our motivation experiment in Sec. 2, the DDNN training performance in the cloud can be degraded by up to 137.6% because of *blindly* scaling out the provisioned DNN training cluster, which inevitably limits the processing ability of workers.

To solve the performance issue above, many research efforts have been devoted to optimizing the synchronization mechanism (*e.g.,* SpecSync [35]) and tuning the learning rate (*e.g.,* DYNSGD [15]) or the mini-batch size (*e.g.,* R^2SP [8]) for DDNN training workloads. However, little attention has been paid to guaranteeing the DDNN training performance in the cloud. There have also been investigations on optimizing cloud resource provisioning plans to minimize the DDNN training time. Most existing resource schedulers such as Optimus [21] and SLAQ [36] rely on the quality of profiling data samples and thus bring heavy workload profiling overhead. Moreover, the existing DDNN performance models (*e.g.,* Paleo [23]) imprecisely predict the training performance under the circumstance of resource bottleneck and heterogeneity. As a result, there has been scant research devoted to predicting the DDNN training performance in a lightweight manner to explicitly explore the resource bottleneck and heterogeneity, so as to deliver predictable performance [31] to DDNN training by an adequate resource provisioning in the public cloud.

To address the challenge above, in this paper, we design *Cynthia*, a cost-efficient resource scheduling framework to provide predictable performance [31] for DDNN training workloads in the cloud. As an example, given a DDNN training workload with an expected completion time and a target training loss value, *Cynthia* first predicts the DDNN training performance using a lightweight analytical performance model and then identifies an optimal resource provisioning plan in the cloud, so as to train the DNN model in a cost-efficient manner. Specifically, we make the following contributions in *Cynthia*.

▷ We build a lightweight analytical performance model for DDNN workloads. Specifically, our *Cynthia* model leverages the resource consumption of workers and PS nodes to explicitly capture the resource bottleneck (*i.e.*, CPU and network bandwidth) and heterogeneity in the cloud. To reduce the profiling overhead, our model requires profiling the DDNN workload only once on a baseline worker to obtain the essential model parameters.

▷ Based on the DDNN performance model, we design a cloud resource provisioning strategy to deliver predictable training performance to DDNN workloads. Given the training performance goal (*i.e.,* the objective training time and loss value), *Cynthia* first calculates the lower and upper bounds of provisioned workers, and then identifies a cost-efficient resource provisioning plan in the cloud to guarantee the training time and model accuracy.

▷ We implement a prototype of *Cynthia* by integrating our performance model (*i.e.,* performance predictor) and resource provisioning strategy (*i.e.,* resource provisioner) into the master node of a Kubernetes cluster. We conduct our prototype experiments in such a Kubernetes cluster consisting of 56 dockers with three types of cloud instances in Amazon EC2 [2]. Experimental results show that *Cynthia* is able to guarantee the DDNN training performance while saving the monetary cost by up to 50.6%, compared to the state-of-the-art resource provisioning strategies (*e.g.,* Optimus [21]).

The rest of the paper is organized as follows. Sec. 2 conducts an empirical study to identify the key factors that impact the training performance of DDNN workloads. Sec. 3 devises a DDNN training performance model that unifies the key factors above, which further guides the design of our *Cynthia* resource provisioning strategy in Sec. 4. Sec. 5 conducts extensive prototype experiments in the cloud to evaluate the effectiveness and runtime overhead of *Cynthia*. Sec. 6 discusses related work and Sec. 7 concludes our paper.

Table 1: Configurations of four representative DDNN training workloads in motivation experiments in Amazon EC2.

	ResNet-32	`mnist` DNN	VGG-19	`cifar10` DNN
#iterations	3,000	10,000	1,000	10,000
Batch size	128	512	128	512
Dataset	`cifar10`	`mnist`	`cifar10`	`cifar10`
Sync.	ASP	BSP	ASP	BSP
Cluster	Homogeneous: 18 m4.xlarge dockers Heterogeneous: 6 m4.xlarge and 4 m1.xlarge dockers			

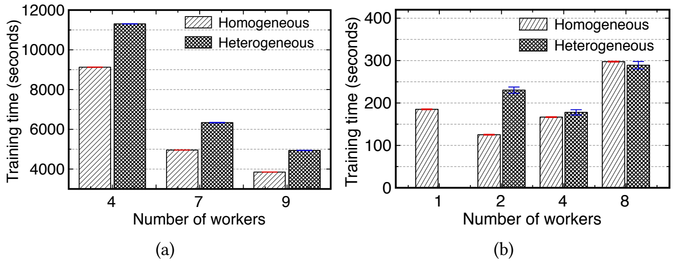


Figure 1: Training time of (a) ResNet-32 with ASP and (b) the `mnist` DNN with BSP in homogeneous and heterogeneous clusters. The heterogeneous cluster contains $\lfloor \frac{n}{2} \rfloor$ stragglers (*i.e.,* m1.xlarge dockers), where n is the number of workers.

2 UNDERSTANDING DDNN TRAINING PERFORMANCE IN THE CLOUD

In this section, we seek to understand the training performance (*i.e.,* training time and training loss) of DDNN workloads running in the public cloud. We then illustrate the key factors that cause the unpredictable performance of DDNN training.

Specifically, we conduct our motivation experiments on a cluster of 28 dockers deployed on Amazon EC2 instances [2] using the latest Kubernetes 1.14 [7]. Our 28-docker cluster is set up using two types of EC2 instances (*i.e.,* 24 m4.xlarge dockers each equipped with Intel Xeon CPU E5-2686 v4 plus 4 m1.xlarge dockers each equipped with Intel Xeon CPU E5-2651 v2). We train four representative types of DNN training workloads, including the `mnist` DNN and the `cifar10` DNN[1], as well as the ResNet-32 model and the VGG-19 model both trained on the `cifar10` dataset. We deploy the asynchronous parallel (ASP) mechanism and the bulk synchronous parallel (BSP) mechanism to update the model parameters. In particular, we focus on the data parallelism and the PS distribution strategy using the Stochastic Gradient Descent (*i.e.,* SGD) optimizer, as such a training configuration is commonly used in production DDNN training [17]. The detailed training parameters are summarized in Table 1.

Training time: As shown in Fig. 1, we observe that: (a) the DDNN training time with ASP is expectedly decreased and *interestingly*, (b) the training time for the `mnist` DNN with BSP is first decreased and then surprisingly increased, as more training workers are provisioned. The rationale is that, the CPU and network resources of the PS can easily become *bottlenecks* when training the DNN

[1]The default DNN structures defined in the path "/models/tutorials/images/" in Tensorflow are trained on the `mnist` and `cifar10` datasets, respectively.

Table 2: Average CPU utilization of the PS and the worker for training the mnist DNN in both homogeneous and heterogenous clusters.

	Homogeneous cluster		Heterogeneous cluster	
	PS	worker	PS	worker (m4)
1 worker	32.6%	100.0%	N/A	N/A
2 workers	80.5%	100.0%	40.8%	51.1%
4 workers	100.0%	54.0%	100.0%	50.5%
8 workers	100.0%	26.2%	100.0%	27.0%

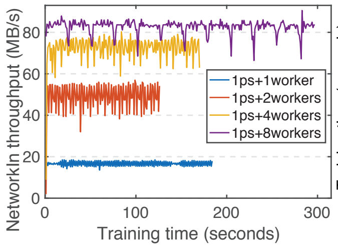


Figure 2: Network throughput of the PS node for training the mnist DNN with BSP.

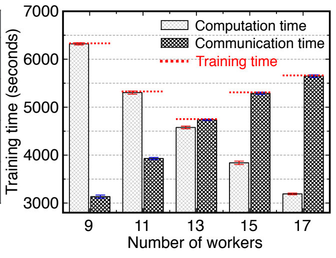
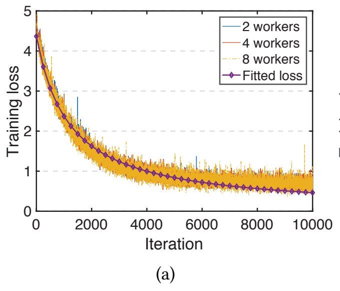


Figure 3: DDNN Training time breakdown for training the cifar DNN with BSP.



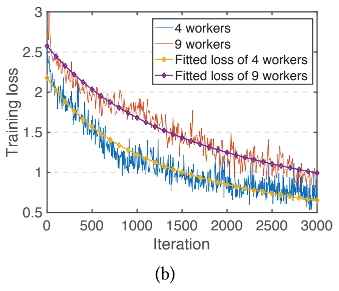


Figure 4: Training loss of (a) the cifar10 DNN with BSP and (b) ResNet-32 with ASP both trained on the cifar10 dataset in the homogeneous cluster.

model with the BSP synchronization mechanism. To verify that, we look into the CPU utilization of the PS and workers listed in Table 2. We find that the PS becomes *CPU bounded* when training the `mnist` DNN with more than 4 provisioned workers, which in turn restricts the CPU utilization of workers under 100%. In more detail, the CPU utilization of the worker is dramatically decreased from 100% to 26.2% as the number of workers increases from 2 to 8 in the homogeneous cluster.

We further examine the network throughput of the PS with different numbers of workers over time as shown in Fig. 2. As expected, we observe that the network bandwidth becomes almost *saturated* [8] (*i.e.,* around 70 MB/s-90 MB/s) for the `mnist` DNN, as the number of workers increases from 4 to 8. Similarly, we also observe the network bandwidth bottleneck (*i.e.,* around 60 MB/s-110 MB/s) on the PS when training VGG-19 with ASP. To exclude the impact of CPU resource, we also examine the network throughput of the PS configured with different amount of CPU resources (*i.e.,* 1, 2, and 4 CPU cores) when training the `mnist` DNN with 8 workers. We find that the network throughput of the PS remains saturated even when more CPU resources are configured to the PS. Such an observation confirms our analysis of CPU and network bandwidth resource bottlenecks on the PS.

In addition to the resource bottleneck on the PS discussed above, the *resource heterogeneity* can also impact the DDNN training performance. As expected, the training time is increased by up to 84% with both BSP and ASP in the heterogeneous cluster. This is because the processing ability of workers gets poor as the stragglers can block and prolong the training process. In particular, the training performance of `mnist` DNN with BSP in the heterogeneous cluster is comparable to that in the homogeneous cluster, as the number of provisioned workers exceeds 4 shown in Fig. 1(b). This is because the CPU and network resources on the PS become bottlenecks in both the heterogeneous and homogeneous clusters as discussed before, which in turn makes the worker (*i.e.,* m4.xlarge docker) underutilized, as evidenced by Table 2.

The increasing number of provisioned workers can cause the *imbalance* of computation and communication, which inevitably prolongs the DDNN training with BSP. To illustrate that, we break down the training time of `cifar10` DNN with BSP by examining the computation time and the communication time shown in Fig. 3. As the number of provisioned workers increases from 9 to 17, the computation time is decreasing sharply as the communication time is increasing, which implies that the CPU and network resources are not bottlenecked on the PS. As the DDNN training time is determined by the maximum of computation time and communication time for the BSP mechanism[2], blindly provisioning more workers for DDNN training can cause an inefficient resource usage (*e.g.,* spending most of the training time in communication makes the CPU computation resource idle). Accordingly, striking a balance between computation and communication (*e.g.,* 13 workers in Fig. 3) is mandated for the *cost-efficient* resource provisioning.

Summary 1: The DDNN training time is essentially determined by the CPU and network bandwidth resources of both PS nodes and workers. We identify three key factors that can cause the *cost-inefficient* resource provisioning of PS nodes and workers, including the resource bottleneck on the PS and the resource heterogeneity of workers, as well as the imbalance of computation and communication, during the process of DDNN training.

Training loss: As shown in Fig. 4, we observe that the training loss of DNN models is roughly in *inverse proportion* to the number of iterations, which is consistent with the latest measurement studies [21, 36] on the DNN training loss. This is because the commonly used SGD optimization converges at a rate of $O(\frac{1}{s})$ for both ASP and BSP [21], where s is the number of iterations. In more detail, the training loss stays unchanged for BSP as more workers are provisioned, as shown in Fig. 4(a). The rationale is that the mini-batch size of each iteration for BSP keeps constant, thereby resulting in a roughly fixed convergence rate for different numbers of workers when training `cifar10` DNN. In contrast, the training loss with ASP converges at a slower rate as the ResNet-32 model is trained on more workers illustrated in Fig. 4(b). This is because the ASP mechanism is likely to introduce the parameter staleness [15] on

[2]To optimize the DDNN training performance with the BSP mechanism, Tensorflow's API `SyncReplicasOptimizer` [1] overlaps the computation and communication.

Table 3: Key notations in our DDNN training performance model.

Notation	Definition
$\mathcal{I}$	Set of training iterations for a DNN model
$\mathcal{I}_{base}$	Set of training iterations on a baseline worker
t^i_{comp}	CPU computation time for an iteration i
t^i_{comm}	Data communication time for an iteration i
w_{iter}	Amount of FLOPs for each iteration
r^j_{wk}	CPU processing rate of a worker j
c^j_{wk}, c^k_{ps}	CPU capability of a worker j and a PS node k
n_{wk}, n_{ps}	Number of provisioned workers and PS nodes
g_{param}	Size of model parameter data on one worker
b^k_{ps}	Available network bandwidth of a PS node k
u^j_{wk}	CPU utilization of a worker j
r_{scale}	Scaling ratio of CPU and network resources

workers, as the worker will miss more fresh parameter updates when the scale of the training cluster increases, thereby slowing down the training convergence rate. In addition, we observe that the stragglers have a *negligible* impact on the training loss, which is different from our observation of the training time in Fig. 1. This is because the DNN model still converges regularly as long as the staleness of parameters is bounded [14].

Summary 2: Based on the characteristics of SGD above, we find that the DDNN training loss solely depends on *the number of iterations* for BSP, and it also correlated positively with *the number of provisioned workers* for ASP. Accordingly, we *empirically* fit the distributed training loss values of DNN models using the polynomial regression method [24]. The fitted training loss model is formulated as

$$f_{loss}(s,n) = \begin{cases} \frac{\beta_0}{s+\beta_1} & \text{BSP training,} \\ \frac{\beta_0\sqrt{n}}{s+\beta_1} & \text{ASP training,} \end{cases} \tag{1}$$

where s denotes the number of training iterations and n denotes the number of provisioned training workers. β_0 and β_1 are the model coefficients, and in general $\beta_0 > \beta_1$. Note that our model in Eq. (1) works for the DDNN training with the SGD optimizer, and we can use our method above to fit the training loss achieved by the other optimization methods (*e.g.,* Adam [10]).

3 PREDICTING DDNN TRAINING PERFORMANCE WITH RESOURCE CONSUMPTION

In this section, we proceed to devise a *simple yet effective* analytical model to capture the variance of distributed training performance for DNN models. Based on the observations in Sec. 2, our performance model leverages the resource consumption (*i.e.,* the *CPU processing rate* and the *available network bandwidth*) on both workers and PS nodes, to explicitly unify the three factors including the resource bottleneck and heterogeneity as well as the imbalance of computation and communication. The parameters of our training performance model are summarized in Table 3.

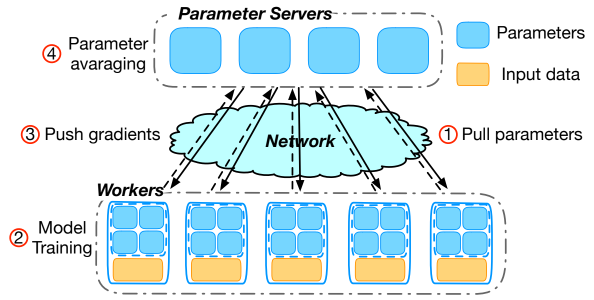


Figure 5: Distributed training process of a single iteration.

As evidenced by Sec. 2, the DNN model is commonly trained by a set of iterations $\mathcal{I}$, and the processing time for each iteration $i \in \mathcal{I}$ consists of the computation time t^i_{comp} for model training and the communication time t^i_{comm} for parameter updating. Accordingly, we formulate the DDNN training time T_{train} as the sum of the iteration processing time t^i_{iter}, which is given by

$$T_{train} = \sum_{i \in \mathcal{I}} t^i_{iter}. \tag{2}$$

In particular, the iteration processing time t^i_{iter} stays the same for each iteration i with BSP, while such a time can be different for the iterations trained on heterogeneous workers with ASP in parallel. We accordingly estimate the DDNN training time with ASP by leveraging the iteration set (*i.e.,* $\mathcal{I} = \mathcal{I}_{base}$) on a *baseline* worker.

As shown in Fig. 5, each iteration of the model training includes the computation using CPU resources of workers and PS nodes as well as the data communication over the network. In general, the gradient computation and the data communication are processed *in sequential* for ASP. To particularly optimize the training barrier with BSP, it is not uncommon for distributed machine learning frameworks to *overlap* the computation and communication as evidenced in Sec. 2, by pulling parameters and pushing gradients (*i.e.,* step 1 and step 3 in Fig. 5) during the process of model training on workers (*i.e.,* step 2). Accordingly, we formulate the iteration processing time t^i_{iter} as

$$t^i_{iter} = \begin{cases} \max(t^i_{comp}, t^i_{comm}) & \text{BSP training,} \\ t^i_{comp} + t^i_{comm} & \text{ASP training.} \end{cases} \tag{3}$$

(1) *Gradient computation time:* The computation time t^i_{comp} can be reduced to the model training time on the workers as the parameter averaging time on the PS (*i.e.,* step 4) is commonly negligible as compared to the model training process on workers (*i.e.,* step 2) [37]. Specifically, the BSP mechanism processes each training iteration i across all the provisioned workers, and thus the iteration processing time for BSP is determined by the slowest worker, while the ASP mechanism executes an iteration i on a specific worker j. Accordingly, we formulate the computation time t^i_{comp} as

$$t^i_{comp} = \begin{cases} \frac{w_{iter}}{n_{wk} \cdot \min_j r^j_{wk}} & \text{BSP training,} \\ \frac{w_{iter}}{r^j_{wk}} & \text{ASP training,} \end{cases} \tag{4}$$

where w_{iter} denotes the amount of floating point operations (*i.e.,* FLOPs) for each iteration, which consumes the same amount of CPU cycles [21] for a training iteration $i \in \mathcal{I}$. We use r^j_{wk} to denote the CPU processing rate (measured in FLOPS) of a worker $j \in [1, n_{wk}]$. We calculate $r^j_{wk} = c^j_{wk} \cdot u^j_{wk}$, where c^j_{wk} denotes the CPU processing capability of a worker j and $u^j_{wk} \in [0, 100\%]$.

(2) *Data communication time:* The data communication is comprised of pushing gradients and pulling model parameters. As the size of model parameters is equal to the amount of gradient update data [21], the communication time can be considered as the double transfer time of model parameter data g_{param} over the network. As discussed before, the PS node communicates with all provisioned workers (*i.e.,* n_{wk}) in one iteration for BSP, while the ASP mechanism only requires the data communication with a single worker in one iteration. Accordingly, the data communication time is formulated as

$$t^i_{comm} = \begin{cases} \frac{2 \cdot g_{param} \cdot n_{wk}}{\sum_k b^k_{ps}} & \text{BSP training,} \\ \frac{2 \cdot g_{param}}{\sum_k b^k_{ps}} & \text{ASP training,} \end{cases} \tag{5}$$

where b^k_{ps} denotes the amount of available network bandwidth of a PS node $k \in [1, n_{ps}]$, and thus the data communication bandwidth is the sum of available network bandwidth across the provisioned PS nodes (*i.e.,* n_{ps}).

Obtaining model parameters: We obtain the three key model parameters above (*i.e.,* $w_{iter}, g_{param}, r^j_{wk}$) through profiling the DDNN training workload with a fixed number of iterations (*e.g.,* 30) on one *baseline* worker with the CPU processing capability c_{base} (measured in FLOPS). Specifically, we calculate the amount of FLOPs for each iteration as $w_{iter} = t_{base} \cdot c_{base}$, where t_{base} is the computation time of a single iteration when the DNN model is trained on the baseline worker. The size of model parameters g_{param} is fixed and it can be measured as the amount of network communication data on the PS divided by the number of iterations (*e.g.,* 30) during the job profiling [33]. In addition, the amount of network bandwidth b^k_{ps} of a PS node k depends on the instance type of the provisioned PS, and can be measured only once using the `netperf` tool. The CPU processing capability c^j_{wk} of a worker j and c^k_{ps} of a PS node k can be *statically* obtained by looking up the CPU processing capability table [3].

Estimating resource utilization of workers: To calculate the CPU processing rate $r^j_{wk}, \forall j \in [1, n_{wk}]$, we need to obtain the resource utilization u^j_{wk} of a worker j, by taking into account the effect of *resource bottleneck* as discussed in Sec. 2. In particular, we use the *demand-supply ratio* [32] of CPU and network bandwidth resources to indicate the severity of resource bottleneck and estimate the resource utilization u^j_{wk} of a worker j. Specifically, we first measure the CPU consumption c_{prof} (calculated by multiplying the CPU utilization and CPU capability of the PS and measured in FLOPs) and the network throughput b_{prof} on the PS node[3], when profiling the DDNN workload on a *baseline* worker. We further obtain the CPU processing demand c_{demand} and the network bandwidth demand b_{demand} of PS nodes as

$$c_{demand} = c_{prof} \cdot r_{scale}, \quad b_{demand} = b_{prof} \cdot r_{scale}, \tag{6}$$

[3]In the environment of one PS node and one worker, the PS node commonly does not become the resource bottleneck and thus the worker fully utilizes the CPU resource.

where r_{scale} denotes the scaling ratio of the resource demand of the PS in a provisioned deep learning cluster, to that in the scenario of a single PS node and a baseline worker. As the workers are processing the DNN workload with the CPU processing capability $c^j_{wk}, j \in [1, n_{wk}]$ in the cluster, we simply estimate the r_{scale} as the ratio of the CPU processing capability of all provisioned workers to the baseline worker with the CPU processing capability c_{base} in the workload profiling above, which is given by

$$r_{scale} = \begin{cases} \frac{n_{wk} \cdot \min_j c^j_{wk}}{c_{base}} & \text{BSP training,} \\ \frac{\sum_j c^j_{wk}}{c_{base}} & \text{ASP training.} \end{cases} \tag{7}$$

With the given CPU resource supply $c_{supply} = \sum_k c^k_{ps}, \forall k \in [1, n_{ps}]$ and the network bandwidth resource supply $b_{supply} = \sum_k b^k_{ps}, \forall k \in [1, n_{ps}]$, the resource bottleneck occurs on the PS node when the CPU resource demand exceeds the CPU resource supply (*i.e.,* $c_{demand} > c_{supply}$) or the network bandwidth demand exceeds the network bandwidth supply (*i.e.,* $b_{demand} > b_{supply}$) [32]. As evidenced by Sec. 2, the CPU and network bandwidth bottlenecks on the PS node can *adversely* limit the CPU utilization (and thus the CPU processing rate) of workers. As a result, when the resource bottleneck occurs, we estimate the CPU resource utilization of workers as the minimum of the demand-supply ratios on the CPU and network resources of the PS node, which is given by

$$u^j_{wk} = \begin{cases} \min\left\{\frac{b_{supply}}{b_{demand}}, \frac{c_{supply}}{c_{demand}}\right\} \cdot 100\% & \text{bottleneck occurs,} \\ 100\% & \text{otherwise.} \end{cases}$$

After obtaining the CPU utilization of workers above, we are able to calculate the CPU processing rate of workers r^j_{wk} by multiplying the CPU processing capability c^j_{wk} and the CPU utilization u^j_{wk} of a worker j. The accuracy of our DDNN training performance model will be validated in Sec. 5.1.

4 GUARANTEEING DDNN TRAINING PERFORMANCE WITH COST-EFFICIENT CLOUD RESOURCES

Based on the DDNN training performance model in Sec. 3, in this section, we proceed to formulate our resource provisioning problem in the form of a monetary cost minimization model (in Eq. (8)) with the constraints on the DDNN training time and the loss value. We then design *Cynthia*, a cost-efficient resource provisioning strategy (in Alg. 1) to guarantee the performance and reduce the monetary cost of DDNN training in the cloud.

Problem formulation: Given an objective DDNN training time T_g and loss value l_g, we aim to provision an appropriate amount of cloud resources to guarantee the training performance while minimizing the monetary cost. To avoid an inefficient resource usage of instances, it is beneficial to provision a *homogeneous* cluster of workers to train the DNN model with both BSP and ASP, as evidenced by summary 1 in Sec. 2. Furthermore, a cost-efficient cloud resource provisioning for the DDNN workload also avoids the resource bottleneck and balances the computation and communication of training with BSP. In particular, each training iteration

$i \in \mathcal{I}$ has the same processing time t_{iter} in the cluster of homogeneous workers. With the training time of a single iteration in Eq. (3), we accordingly formulate our resource provisioning cost model as below.

$$\min_{n_{wk}, n_{ps}} \quad C = (p^t_{wk} \cdot n_{wk} + p^t_{ps} \cdot n_{ps}) \cdot t_{iter} \cdot s \tag{8}$$

$$\text{s.t.} \quad t_{iter} \leq \frac{T_g}{s}, \tag{9}$$

$$f_{loss}(s, n_{wk}) = l_g, \quad n_{wk} \in \mathcal{Z}^+ \tag{10}$$

$$1 \leq \frac{n_{wk}}{n_{ps}} \leq r, \quad n_{wk}, n_{ps} \in \mathcal{Z}^+ \tag{11}$$

where p^t_{wk} and p^t_{ps} denote the hourly price for workers and PS nodes of the instance type t, respectively, and $s \in \mathcal{Z}^+$ denotes the number of training iterations which is required to achieve the objective training loss l_g. In more detail, Constraint (9) confines the iteration processing time below $\frac{T_g}{s}$, and Constraint (10) achieves the objective training loss value l_g. Constraint (11) represents the resource provisioning ratio of workers to PS nodes, so as to maximize the utilization of provisioned cloud resources. Typically, the number of provisioned PS nodes is less than that of workers to achieve an efficient instance resource utilization [19]. The coefficient r denotes the maximum resource provisioning ratio that prevents the bottlenecks on the CPU and network bandwidth resources.

In more detail, we calculate the coefficient r in the homogeneous cluster of workers and PS nodes. In order to make the workers fully utilized, the CPU and network bandwidth resource bottlenecks cannot occur on the PS (*i.e.*, $c_{demand} \leq c_{supply}$, $b_{demand} \leq b_{supply}$). By inputting Eq. (7) into Eq. (6), we are able to calculate the maximum resource provisioning ratio as

$$r = \min\left(\frac{c_{base} \cdot c_{ps}}{c_{prof} \cdot c_{wk}}, \frac{b_{ps} \cdot c_{base}}{b_{prof} \cdot c_{wk}}\right). \tag{12}$$

Model analysis: By substituting Eq. (3)–Eq. (5) into Constraint (9), we find that the constraint on the iteration processing time is non-linear. The resource provisioning cost minimization problem defined in Eq. (8) above is in the form of *non-linear integer programming*, which is *non-convex* and is NP-hard to solve [6]. Accordingly, we turn to devising a heuristic algorithm to solve such a problem. To reduce the algorithm complexity, we calculate the lower and upper bounds of provisioned workers according to the constraints defined in Constraint (9)–(11).

Theorem 4.1. *Given a DDNN training workload with the performance goal, the cost-efficient resource provisioning plan is to launch the minimum number of PS nodes n_{ps}. The upper bound and lower bound of n_{wk} can be formulated as below.*

$$n^{lower}_{wk} = \begin{cases} \left\lceil \frac{w_{iter} \cdot s}{T_g \cdot c_{wk}} \right\rceil & BSP, \\ \left\lceil \left(\frac{w_{iter} \cdot (\beta_0 - \beta_1)}{c_{wk} \cdot T_g \cdot l_g} \right)^2 \right\rceil & ASP. \end{cases} \tag{13}$$

$$n^{upper}_{wk} = \begin{cases} \left\lceil \min\left(u \cdot n_{ps}, \sqrt{\frac{w_{iter} \cdot n_{ps} \cdot b_{ps}}{2 \cdot g_{param} \cdot c_{wk}}}\right) \right\rceil & BSP, \\ \lceil r \cdot n_{ps} \rceil & ASP, \end{cases} \tag{14}$$

where $u = \min\left(r, \frac{T_g \cdot b_{ps}}{2 \cdot s \cdot g_{param}}\right)$ *is the updated resource provisioning ratio for BSP.*

The proof can be found in Appendix A.

Cloud resource provisioning strategy: Given a target loss value l_g and an expected training time T_g for a DDNN workload with a training loss function $f_{loss}(s, n_{wk})$, we first initialize the monetary cost and the amount of provisioned PS nodes and workers. Through profiling the DDNN training workload on a PS node and a baseline worker c_{base}, we then obtain the DNN training specific parameters including w_{iter}, g_{param}, c_{prof}, and b_{prof} (lines 1-2). By inputting the target loss value l_g into the DNN loss function, we are able to obtain the number of iterations s to achieve the expected training loss (line 3). For each instance type t, we further obtain the instance specific parameters c_{wk}, c_{ps}, and b_{ps} in the cluster of homogeneous workers and PS nodes as well as the coefficient r in Constraint (11) (line 5). To narrow down the searching space of the number of provisioned workers, we calculate the upper and lower bounds of n_{wk} as well as the minimum number of PS nodes for both BSP and ASP training (line 6). Through iterating all the possible number of provisioned workers, we calculate the iteration processing time by Eq. (3) and the monetary cost using Eq. (8). Among all the possible number of provisioned workers, we find the candidate provisioned cloud resources to achieve the training iteration time within $\frac{T_g}{s}$, and then identify the cost-efficient cloud resource provisioning plan including the instance type t_{min} and the number of PS nodes n_{ps} and workers n_{wk} with the minimum monetary cost C_{min} (lines 7-13).

Algorithm 1: *Cynthia:* Cloud resource provisioning strategy to guarantee DDNN training performance and minimize the monetary cost.

```
Input: Objective training time $T_g$ and training loss value $l_g$ for a DDNN
    workload, and training loss function $f_{loss}(s, n_{wk})$.
Output: Cost-efficient cloud resource provisioning plan including the
    number and the type $t_{min}$ of PS nodes $n_{ps}$ and workers $n_{wk}$.
1:  Initialize: the monetary cost of cloud resource provisioning
    $C_{min} \leftarrow \infty$, $n_{ps} \leftarrow 0$, and $n_{wk} \leftarrow 0$;
2:  Obtain the model training parameters $w_{iter}$, $g_{param}$, $c_{prof}$, and
    $b_{prof}$ through profiling the DNN workload;
3:  Calculate the number of iterations $s$ that achieves the target loss
    $f_{loss}(s, n_{wk}) \leftarrow l_g$;
4:  for each instance type $t$ do
5:      Obtain the instance parameters $c_{wk}$, $c_{ps}$, and $b_{ps}$, and calculate
        the maximum resource provisioning ratio $r \leftarrow$ Eq. (12);
6:      Calculate the lower bound $n^{lower}_{wk} \leftarrow$ Eq. (13), $n^{upper}_{wk} \leftarrow$ Eq. (14)
        and the minimum number of provisioned PS nodes $n_{ps}$.
7:      for all $\hat{n_{wk}} \in [n^{lower}_{wk}, n^{upper}_{wk}]$ do
8:          Calculate $t_{iter} \leftarrow$ Eq. (3), and $C \leftarrow$ Eq. (8);
9:          if $t_{iter} \cdot s < T_g$ && $C_{min} > C$ then
10:             Record the monetary cost $C_{min} \leftarrow C$ and the instance type
                $t_{min} \leftarrow t$, as well as the number of workers $n_{wk} \leftarrow \hat{n_{wk}}$;
11:             break out of the loop;
12:         end if
13:     end for
14: end for
```

Remark: The complexity of our algorithm is in the order of $O(m \cdot p)$, where $m = n^{upper}_{wk} - n^{lower}_{wk}$ denotes the searching space of n_{wk}, and p is the *limited* number of provisioned instance types in a public cloud. The instance specific parameters above can be offline

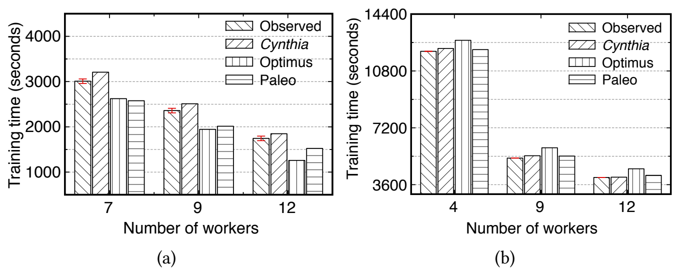


Figure 6: Comparison of the observed and predicted training time for (a) VGG-19 with ASP and (b) the `cifar10` DNN with BSP, under the *Cynthia*, Optimus, and Paleo models.

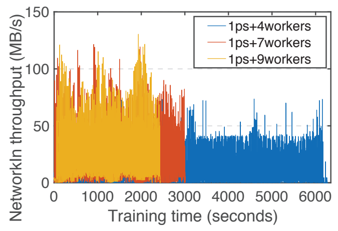


Figure 7: Network throughput of the PS node for training VGG-19 with ASP in a homogeneous cluster.

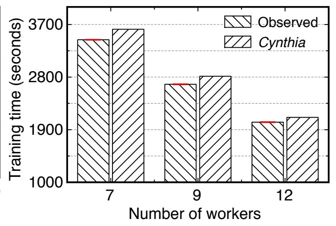


Figure 8: Comparison of the observed and *Cynthia* predicted time for VGG-19 with ASP in the r3.xlarge cluster.

measured only once. Also, the loss function $f_{loss}(s, n_{wk})$ can be obtained by executing the DDNN training job once, as the DDNN workloads are repeatedly executed in production clusters [12]. In particular, the loss function for the ASP mechanism is related to both the number of iterations s and the number of provisioned workers n_{wk} as evidenced by Eq. (1). We accordingly update the number of iterations s that is required to achieve the expected loss value l_g using Eq. (20) in Appendix A for each number of provisioned workers n_{wk}. The runtime overhead of our *Cynthia* strategy is well contained and will be validated in Sec. 5.3.

5 PERFORMANCE EVALUATION

In this section, we carry out a set of prototype experiments in Amazon EC2 to evaluate the effectiveness and runtime overhead of *Cynthia*, in comparison to the state-of-the-art resource provisioning strategies (*e.g.,* Optimus [21]). Specifically, we examine the prediction accuracy of our DDNN training performance model, the training performance and monetary cost saving achieved by our resource provisioning strategy, as well as the computation time and profiling overhead of *Cynthia*.

***Cynthia* prototype:** We implement a prototype of *Cynthia* with two pieces of modules including a *performance predictor* and a *resource provisioner* based on Kubernetes v1.14 [7] in Amazon EC2 [2]. Specifically, we initialize a master node of the Kubernetes cluster using a general-purpose cloud instance (*e.g.,* m4.xlarge). The two modules of *Cynthia* above are integrated into the master node of the training cluster. Once a DDNN training job is submitted, the expected number of iterations to achieve the objective training loss can be calculated by Eq. (1). The *performance predictor* predicts the DDNN training time using the performance model elaborated in Sec. 3, which is built according to the workload profiling on a baseline worker. The *resource provisioner* further identifies the cost-efficient cloud instance provisioning plan. Once the plan is determined, the cloud instances are provisioned using `AWS CLI`. After the instances automatically install the `docker`, `kubelet`, and `kubeadm` components, the provisioned cloud instances can join the training cluster with the `token` and `discovery-token-ca-cert-hash` generated by the master node using the command "`kubeadm join`".

Testbed: We conduct our *Cynthia* prototype experiment on a cluster of 56 dockers using 28 EC2 instances. In addition to the 28 dockers with two instance types (*i.e.,* m4.xlarge instances and m1.xlarge instances) in Sec. 2, we further add 28 dockers on 14 r3.xlarge instances which is equipped with Intel Xeon CPU E5-2670 v2 to our docker cluster. To avoid potential CPU contention caused by Hyper-threading, we host one docker on each instance physical CPU core. We focus on two key metrics including the DDNN training time and the monetary cost for each resource provisioning plan. We illustrate the DDNN workload performance with error bars of standard deviations by repeating the DDNN training workload for three times.

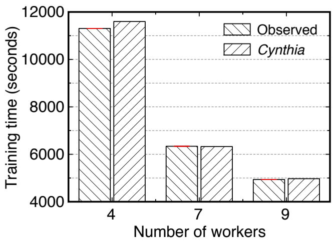
Table 4: DNN training specific parameters obtained from the 30-iteration profiling for four DDNN workloads on an m4.xlarge worker.

	ResNet-32	VGG-19	`cifar10`	`mnist`
w_{iter} (GFLOPS)	39.87	58.81	26.86	0.04
g_{param} (MB)	2.22	135.84	4.94	0.33
c_{prof} (GFLOPS)	0.12	0.33	0.06	1.13
b_{prof} (MB/s)	0.19	13.49	1.56	16.69

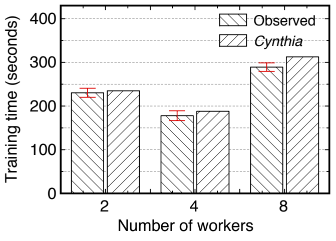
5.1 Validating DDNN Training Performance Model in *Cynthia*

We evaluate the training time of representative DNN benchmark workloads including ResNet-32, VGG-19, the `mnist` DNN, and the `cifar10` DNN, as listed in Table 1, running on four different instance types including m4.xlarge, m1.xlarge, c3.xlarge, and r3.xlarge. We compare our *Cynthia* performance model with the traditional Paleo model [23] and the state-of-the-art Optimus model [21]. By profiling the DDNN workloads with 30 iterations, we are able to obtain the training specific parameters including w_{iter}, g_{param}, c_{prof}, and b_{prof} as listed in Table 4.

Can *Cynthia* well predict the DDNN training time? We examine the DDNN training time by varying the number of provisioned workers with one fixed PS node. As shown in Fig. 6, we observe that our *Cynthia* model can well predict the training time of VGG-19 with ASP and `cifar10` DNN with BSP in the homogeneous cluster with an error of 1.6%-6.3% on average, in comparison to 2.2%-19.4% achieved by the Optimus and Paleo performance models. The DDNN training time predicted by Optimus and Paleo is comparatively accurate with a prediction error of 2.2%-10.6%, as long as the provisioned PS resources are sufficient and thus the resource contention does not occur on the PS node (*i.e.,* `cifar10` DNN with BSP shown in Fig. 6(b)). The training time of `cifar10` DNN predicted by Optimus and Paleo is slightly beyond the observed time, because both of them are oblivious to the training

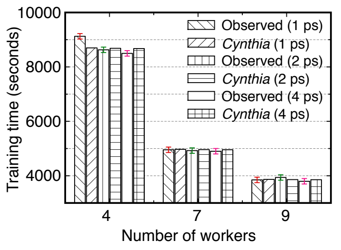


(a)

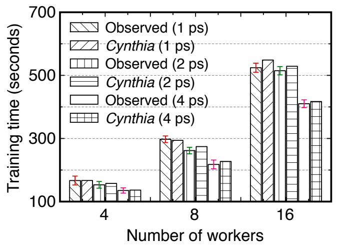


(b)

Figure 9: Observed and *Cynthia* predicted training time for (a) ResNet-32 with ASP and (b) mnist DNN with BSP in the heterogeneous cluster, which contains $\lceil \frac{n}{2} \rceil$ m4.xlarge and $\lfloor \frac{n}{2} \rfloor$ m1.xlarge workers, where n is the number of workers.



(a)



(b)

Figure 10: Comparison of the observed and *Cynthia* predicted training time for (a) ResNet-32 with ASP and (b) the mnist DNN with BSP, by varying the number of PS nodes from 1 to 4 and the number of workers from 4 to 16.

optimization for the BSP training. In more detail, rather than overlapping the computation and communication as mentioned in Sec. 2, Optimus and Paleo simplistically consider the training time as the combination of computation and communication.

However, Optimus and Paleo poorly predict the DDNN training time when the provisioned PS node becomes a resource bottleneck. As shown in Fig. 6(a), we observe that the prediction error of VGG-19 achieved by Optimus and Paleo gets larger as the number of provisioned workers exceeds 7. In more detail, the prediction error of VGG-19 achieved by Optimus (Paleo) with 9 workers is 17.6% (12.7%) and that with 12 workers is 27.9% (14.7%). This is because the network bandwidth becomes the bottleneck on the PS node when the number of workers reaches 9. As evidenced by Fig. 7, the network throughput of the PS node increases to around 110 MB/s and thus becomes saturated with 9 workers, thereby limiting the CPU utilization of a worker to 85.4%. When the number of workers exceeds 11, both CPU and network bandwidth bottlenecks occur on the PS node, which further degrades the processing ability of workers as the CPU utilization of a worker is reduced to 64.0%. Moreover, the prediction accuracy of Optimus relies on the quality and the number of data profiling samples, which also results in the inaccurate prediction of the DDNN training performance with resource bottleneck. In contrast, our *Cynthia* model explicitly captures the resource bottleneck using the demand-supply ratio [32] of the resource consumption of workers and PS nodes. Specifically, the prediction error of VGG-19 achieved by our prediction model with 9 workers is 6.4% and that with 12 workers is 5.9% which are significantly better than that achieved by Optimus and Paleo.

In addition, we evaluate the predicted DDNN training time of VGG-19 on the r3.xlarge instance using the workload profiling on the m4.xlarge instance. As shown in Fig. 8, we observe that *Cynthia* predicts the training time of VGG-19 with ASP with an error of 4.0%-5.2%, as the number of provisioned r3.xlarge instances increases from 7 to 12. Such an observation confirms that *Cynthia* does not need to profile the DDNN workload on each instance type, as the DDNN workload requires profiling *only once* on a baseline worker (*e.g.,* m4.xlarge) as discussed in Sec. 3. Accordingly, *Cynthia* is able to predict the training time of the DDNN workload running on different types of cloud instances with the well-contained profiling overhead, which will be validated in Sec. 5.3.

Can *Cynthia* adapt to the resource heterogeneity of workers? As shown in Fig. 9, we observe that our *Cynthia* model can accurately predict the training time of ResNet-32 and mnist DNN with an average error of 1.0%-5.3% in a heterogeneous cluster. Specifically, increasing the number of provisioned workers from 2 to 4 slightly speeds up the training process by 22.7% for the mnist DNN with BSP. This is because the performance of the BSP training is determined by the CPU processing rate of stragglers (*i.e.,* m1.xlarge instance) due to the parameter synchronization (discussed in Sec. 2). Moreover, the training time increases as more workers are provisioned (*i.e.,* n_{wk} grows from 4 to 8), due to the CPU and network resource bottlenecks on the PS node (which further limits CPU utilization of a worker to 27.0%). In contrast, the training performance of ResNet-32 with ASP is still improved as the number of provisioned workers increases as shown in Fig. 9(a).

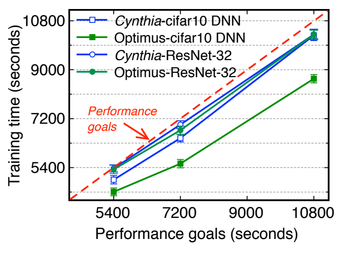
Can *Cynthia* work with multiple PS nodes? As shown in Fig. 10, we observe that *Cynthia* predicts the training time of ResNet-32 and mnist DNN with an error of 1.1%-3.5% as multiple PS nodes are provisioned. Specifically, provisioning more PS nodes can hardly increase the training performance of ResNet-32 with ASP. This is because the ResNet-32 workload cannot saturate the CPU and network resources of the PS node as more workers are provisioned. In contrast, the mnist DNN with BSP has high demand for both CPU and network bandwidth resources on the PS node. Increasing the number of PS can accordingly alleviate the resource bottleneck, so as to expedite the training process of the mnist DNN. As a result, we empirically show that blindly adding more PS nodes to DDNN training can undoubtedly reduce the cost efficiency of resource provisioning, which further validates the selection choice of the minimum number of provisioned PS nodes in Sec. 4.

5.2 Effectiveness of *Cynthia* Resource Provisioning Strategy

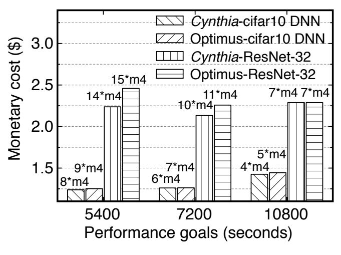
We further examine whether *Cynthia* can guarantee the DDNN training performance by provisioning an appropriate number of workers and PS nodes. Specifically, we set different training performance goals (*i.e.,* training time and loss values) for ResNet-32 and cifar10 DNN models with BSP and VGG-19 with ASP. We compare the DDNN training time and monetary cost achieved by *Cynthia* and the modified Optimus[4] [21].

Can *Cynthia* guarantee the DDNN training time with BSP? As shown in Fig. 11(a), we observe that *Cynthia* basically meets the

[4]We substitute the Optimus model for our performance model in *Cynthia* to calculate the resource provisioning plan, because Optimus is designed to minimize the DDNN training time rather than guaranteeing the training performance.

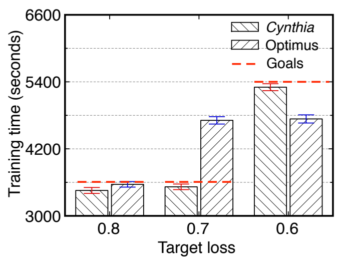


(a)

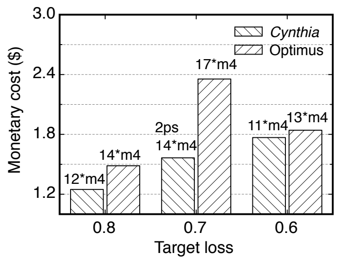


(b)

Figure 11: DDNN training performance and the monetary cost of `cifar10` DNN and ResNet-32 both with BSP achieved by *Cynthia* and Optimus under different performance goals.



(a)



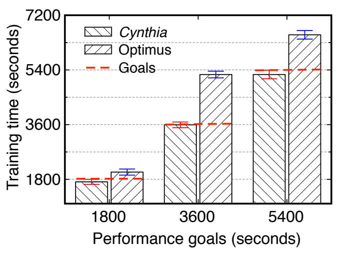
(b)

Figure 12: Comparison of DDNN training performance and the monetary cost achieved by *Cynthia* and Optimus, under various target loss values of `cifar10` DNN with BSP.

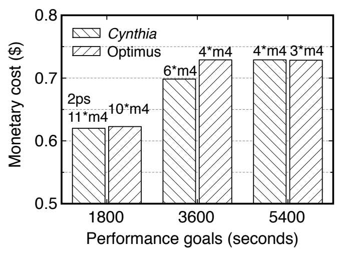
performance goals (*i.e.,* 90, 120, and 180 minutes), with the target loss value of 0.8 for `cifar10` DNN and 0.6 for ResNet-32, respectively. Meanwhile, the monetary cost of DDNN training achieved by *Cynthia* is reduced by 0.9%-9.9% as compared to Optimus. This is because the Optimus prediction model does not overlap the computation time and communication time for BSP, thereby underestimating the training performance. Such performance underestimation further results in the over-provisioning of cloud instances, leading to higher monetary cost and resource under-utilization. In addition, *Cynthia* can reduce more monetary cost of ResNet-32 with ASP in Fig. 11(b), in comparison to the `cifar10` DNN with BSP which has a low demand of cloud instances, as ResNet-32 provisions more workers than `cifar10` DNN.

We further validate whether *Cynthia* can satisfy different objective training loss values (*i.e.,* 0.6, 0.7, and 0.8). As shown in Fig. 12(a), Optimus fails to provide predictable training time (*i.e.,* 60 minutes) for the `cifar10` DNN with the target loss value of 0.7. The rationale is that Optimus cannot accurately capture the imbalance between computation and communication when the number of workers exceeds a threshold (*i.e.,* 13 shown in Fig. 3). In contrast, *Cynthia* provides two PS nodes in a cost-efficient manner to reduce the communication time, thereby balancing the computation and communication during the training process. With the provisioned cloud instances, *Cynthia* brings 4.2%-50.6% monetary cost saving as compared to Optimus under different objective loss values of `cifar10` DNN with BSP as shown in Fig. 12(b).

Can *Cynthia* guarantee the DDNN training time with ASP? We examine whether our *Cynthia* resource provisioning strategy can guarantee the DDNN training with ASP using the VGG-19



(a)



(b)

Figure 13: DDNN training performance and the monetary cost of VGG-19 with ASP achieved by *Cynthia* and Optimus under different performance goals.

workload. Specifically, as shown in Fig. 13(a), we observe that *Cynthia* is basically able to deliver predictable performance to VGG-19 with the target loss of 0.8, under the different performance goals (*i.e.,* 30, 60, and 90 minutes) with ASP. However, Optimus fails to meet the performance goals due to its performance overestimation as illustrated in Fig. 6(b). In particular, *Cynthia* provisions two PS nodes to guarantee the 30-minute performance goal. This is because the CPU and network resources become bottlenecks on the PS node when the number of workers reaches 11 (as illustrated in Sec. 5.1), which cannot be predicted by Optimus. As shown in Fig. 13(b), *Cynthia* marginally reduces the training budget by 0.5%-4.4% with the guaranteed training performance in comparison to Optimus.

5.3 Runtime Overhead of *Cynthia*

We evaluate the runtime overhead of *Cynthia* in terms of the DDNN workload profiling overhead and the computation time of our resource provisioning strategy (*i.e.,* Alg. 1). First, by training the DDNN workloads listed in Table 1 with 30 iterations on a single baseline worker (*e.g.,* m4.xlarge), the profiling time of `mnist` DNN, `cifar10` DNN, ResNet-32, and VGG-19 is only 0.9 seconds, 4.0, 6.0, and 10.4 minutes, respectively. In particular, each DDNN workload requires profiling *only once* to obtain the essential training specific parameters, as the DDNN workloads are repeatedly executed in production clusters [12]. Second, by deploying *Cynthia* on an m4.xlarge instance equipped with 4 Intel Xeon E5-2686 v4 vCPUs and 16 GB memory, *Cynthia* is able to calculate the optimal resource provisioning plan in Sec. 5.2 for `cifar10` DNN, ResNet-32 with BSP, and VGG-19 with ASP using 19, 39, and 13 milliseconds, respectively. Such computation overhead is largely in proportion to the space between the lower and upper bounds of provisioned workers, which further validates the complexity analysis of in Sec. 4. As a result, the runtime overhead of *Cynthia* is practically acceptable.

6 RELATED WORK

DDNN training performance modeling: There have been efforts devoted to predicting the training performance of DDNN workloads. For example, Yan *et al.* [34] devise a fine-grained performance model using the prior knowledge of DNN models such as the network structure and the parallelization strategy. Similarly, Paleo [23] predicts the DDNN training time on a *single* node, by considering various factors including network structures, computation speed, and communication strategies. Yet, these two models above are oblivious to the resource bottleneck and heterogeneity which are not uncommon in DDL clusters. To model the DDNN training time in the GPU cluster, Shi *et al.* [26] profiles the iteration time at a fine granularity, such as the time of forward phase and backward propagation for each DNN workload trained in the cluster, which inevitably brings heavy profiling overhead. A more recent work [21] builds a high-level performance model by considering the pattern of computation and communication of DDNN training and fitting the training speed online. Nevertheless, it highly relies on the quality of data samples and cannot be applied to the heterogeneous cluster as evidenced in Sec. 5. Different from the prior works above, our *Cynthia* model is able to capture the resource bottleneck and heterogeneity in the cloud, by leveraging the CPU and network resource consumption of workers and PS nodes, which brings practically acceptable runtime overhead.

Resource provisioning for DDNN training: There have been recent works on resource provisioning for DDNN training workloads. To minimize the DDNN training time, Dike [29] leverages the characteristics of DNN models (*i.e.,* the number of parameters and layers) to decide the amount of resource allocated to workers and PS nodes. To maximize the overall utility and reduce the average completion time of DDNN training workloads, Optimus [21] and OASiS [4] are designed to provision an appropriate number of workers and PS nodes for each workload in a DDL cluster. To reduce the monetary cost of DDNN workloads, Proteus [13] and FC^2 [27] exploit the EC2 spot instances and different instance types for cloud resource provisioning, respectively. While these prior works above focus on finding an optimal resource provisioning plan to either minimize the training time or save the monetary cost, *Cynthia* aims to *jointly* deliver the predictable performance to DDNN training workloads and save the resource provisioning budget in the cloud.

DDNN training performance optimization: To mitigate the network bandwidth bottleneck on PS nodes, a straightforward approach is to compress the gradients and model parameters without impacting the training performance [18]. A more recent work [8] coordinates the parameter synchronization between workers and PS nodes in a round-robin manner. To alleviate the performance impact caused by stragglers in a heterogeneous cluster, traditional techniques are designed to tune the training hyper-parameters such as the learning rate [15] and the batch size [28]. Recent works (*e.g.,* SpecSync [35], Hop [20]) focus on optimizing the synchronization mechanism by selectively aborting and skipping the iterations on the stragglers without impacting the training accuracy. Orthogonal to these works above, *Cynthia* incorporates the resource bottleneck and heterogeneity into our DDNN training performance model, and it can work with the performance optimization methods so as to improve the cost efficiency of resource provisioning.

7 CONCLUSION AND FUTURE WORK

To provide predictable performance and minimize the monetary cost for DDNN training workloads, this paper presents *Cynthia*, a cost-efficient resource provisioning framework in the cloud, by explicitly exploiting the characteristics of resource bottleneck and heterogeneity as well as the imbalance between computation and communication. Leveraging the resource consumption of workers and PS nodes and the repetitiveness of training iterations, we build an analytical performance model at the system level to accurately predict the DDNN training time through a lightweight workload profiling. Based on our performance model, *Cynthia* further provisions an appropriate type and number of cloud instances to achieve the target training loss within an expected training time while minimizing the monetary cost. Extensive prototype experiments in Amazon EC2 demonstrate that *Cynthia* can guarantee the DDNN training performance and save the monetary cost by up to 50.6%, compared with the state-of-the-art resource provisioning strategies.

As our future work, we plan to examine the effectiveness of *Cynthia* with other DNN models and training datasets (*e.g.,* ResNet-50 on the ImageNet dataset). We also plan to deploy *Cynthia* in the GPU cluster and evaluate its effectiveness and runtime overhead.

ACKNOWLEDGMENTS

The corresponding author is Fei Xu (fxu@cs.ecnu.edu.cn). This work was supported in part by the Science and Technology Commission of Shanghai Municipality under grant No.17511102602 and No.14DZ2260800. Li Chen's work was supported by a grant from BoRSF-RCS. Zhi Zhou's work was supported by the NSFC under Grant 61802449 and the Guangdong Natural Science Funds under Grant 2018A030313032. Fangming Liu's work was supported in part by the NSFC under Grant 61761136014 and 61722206, and in part by National Key Research & Development (R&D) Plan under grant 2017YFB1001703.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: A System for Large-Scale Machine Learning. In *Proc. of OSDI.* 265–283.

[2] Amazon. 2019. *Amazon Elastic Compute Cloud (Amazon EC2).* http://aws.amazon.com/ec2/

[3] Asteroids. 2018. *CPU performance.* https://asteroidsathome.net/boinc/cpu_list.php

[4] Yixin Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. 2018. Online Job Scheduling in Distributed Machine Learning Clusters. In *Proc. of Infocom.* 495–503.

[5] Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *arXiv preprint arXiv:1802.09941* (2018).

[6] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press.

[7] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, omega, and kubernetes. *ACM Queue* (2016), 10–10.

[8] Chen Chen, Wei Wang, and Bo Li. 2019. Round-Robin Synchronization: Mitigating Communication Bottlenecks in Parameter Servers. In *Proc. of Infocom.*

[9] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, et al. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[10] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project adam: Building an efficient and scalable deep learning training system. In *Proc. of OSDI.* 571–582.

[11] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, et al. 2012. Large Scale Distributed Deep Networks. In *Proc. of NIPS.* 1223–1231.

[12] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *Proc. of NSDI.* 485–500.

[13] Aaron Harlap, Alexey Tumanov, Andrew Chung, Gregory R. Ganger, and Phillip B. Gibbons. 2017. Proteus: agile ml elasticity through tiered reliability in dynamic resource markets. In *Proc. of Eurosys.* 589–604.

[14] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A Gibson, et al. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *Proc. of NIPS.* 1223–1231.

[15] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware distributed parameter servers. In *Proc. of SIGMOD.* 463–478.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521, 7553 (2015), 436–444.

[17] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, et al. 2014. Scaling distributed machine learning with the

parameter server. In *Proc. of OSDI*. 583–598.
[18] Hyeontaek Lim, David G. Andersen, and Michael Kaminsky. 2019. 3LC: Lightweight and Effective Traffic Compression for Distributed Machine Learning. In *Proc. of SysML*. 32–43.
[19] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. 2018. Parameter Hub: a Rack-Scale Parameter Server for Distributed Deep Neural Network Training. In *Proc. of SOCC*. 41–54.
[20] Qinyi Luo, Jinkun Lin, Youwei Zhuo, and Xuehai Qian. 2019. Hop: Heterogeneity-Aware Decentralized Training. In *Proc. of ASPLOS*.
[21] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proc. of Eurosys*. 3–3.
[22] Google Cloud Platform. 2018. *Cloud Deep Learning VM Image*. https://cloud.google.com/deep-learning-vm/
[23] Hang Qi, Evan R Sparks, and Ameet Talwalkar. 2017. Paleo: A performance model for deep neural networks. In *Proc. of ICLR*. 1–10.
[24] George A. F. Seber and Alan J. Lee. 2012. *Linear regression analysis*. Vol. 329. John Wiley & Sons.
[25] Amazon Web Service. 2018. *AWS Deep Learning AMIs*. https://aws.amazon.com/machine-learning/amis/
[26] Shaohuai Shi, Qiang Wang, and Xiaowen Chu. 2018. Performance modeling and evaluation of distributed deep learning frameworks on gpus. In *Proc. of DASC/PiCom/DataCom/CyberSciTech*. 949–957.
[27] Nguyen Binh Duong Ta. 2019. FC2: cloud-based cluster provisioning for distributed machine learning. *Cluster Computing* (2019), 1–17.
[28] Shaoqi Wang, Wei Chen, Aidi Pi, and Xiaobo Zhou. 2018. Aggressive Synchronization with Partial Processing for Iterative ML Jobs on Clusters. In *Proc. of Middleware*. 253–265.
[29] Erci Xu and Shanshan Li. 2019. Revisiting Resource Management for Deep Learning Framework. *Electronics* (2019), 327–327.
[30] Fei Xu, Fangming Liu, and Hai Jin. 2016. Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud. *IEEE Trans. Comput.* 65, 8 (2016), 2470–2483.
[31] Fei Xu, Fangming Liu, Hai Jin, and Athanasios V. Vasilakos. 2014. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. *Proc. IEEE* 102, 1 (2014), 11–31.
[32] Fei Xu, Fangming Liu, Linghui Liu, Hai Jin, Bo Li, and Baochun Li. 2014. iAware: Making live migration of virtual machines interference-aware in the cloud. *IEEE Trans. Comput.* 63, 12 (2014), 3012–3025.
[33] Fei Xu, Haoyue Zheng, Huan Jiang, Wujie Shao, Haikun Liu, and Zhi Zhou. 2019. Cost-Effective Cloud Server Provisioning for Predictable Performance of Big Data Analytics. *IEEE Transactions on Parallel and Distributed Systems* 30, 5 (2019), 1036–1051.
[34] Feng Yan, Olatunji Ruwase, Yuxiong He, and Trishul Chilimbi. 2015. Performance Modeling and Scalability Optimization of Distributed Deep Learning Systems. In *Proc. of KDD*. 1355–1364.
[35] Chengliang Zhang, Huangshi Tian, Wei Wang, and Feng Yan. 2018. Stay Fresh: Speculative Synchronization for Fast Distributed Machine Learning. In *Proc. of ICDCS*. 99–109.
[36] Haoyu Zhang, Logan Stafman, Andrew Or, and Michael J. Freedman. 2017. SLAQ: quality-driven scheduling for distributed machine learning. In *Proc. of SOCC*. 390–404.
[37] Wei Zhang, Minwei Feng, Yunhui Zheng, Yufei Ren, Yandong Wang, Ji Liu, Peng Liu, et al. 2017. Gadei: On scale-up training as a service for deep learning. In *Proc. of ICDM*. 1195–1200.

A PROOF OF THEOREM 4.1

PROOF. As the training performance model (in Eq. (3)) and the loss model (in Eq. (1)) are both different for BSP and ASP, we obtain the lower and upper bounds of n_{wk} separately for the two training synchronization mechanisms.

(1) BSP training: By Eq. (1) and Constraint (10), we first obtain the number of training iterations s as

$$s = \left\lceil \frac{\beta_0}{l_g} - \beta_1 \right\rceil, \tag{15}$$

which can be considered as a constant given the objective training loss value l_g. By Eq. (3) and Constraint (9), we then obtain that the computation time and the communication time for each iteration cannot exceed the expected iteration time (*i.e.,* $\frac{w_{iter}}{n_{wk} \cdot c_{wk}} \leq \frac{T_g}{s}$, $\frac{2 \cdot n_{wk} \cdot g_{param}}{n_{ps} \cdot b_{ps}} \leq \frac{T_g}{s}$). Accordingly, we calculate the lower bound of n_{wk} and the upper limit of the resource provisioning ratio $\frac{n_{wk}}{n_{ps}}$ as

$$n_{wk}^{lower} = \left\lceil \frac{w_{iter} \cdot s}{T_g \cdot c_{wk}} \right\rceil, \tag{16}$$

$$\frac{n_{wk}}{n_{ps}} \leq u, \quad \text{where } u = \min\left(r, \frac{T_g \cdot b_{ps}}{2 \cdot s \cdot g_{param}}\right). \tag{17}$$

By substituting Eq. (16) into Eq. (17), we are able to calculate the number of PS nodes n_{ps} as

$$n_{ps} = \left\lceil \frac{n_{wk}^{lower}}{u} \right\rceil. \tag{18}$$

The rationale is that blindly increasing the number of provisioned PS nodes can reduce the cost efficiency of cloud resources, though the DDNN training performance can be slightly improved, which will be validated in Sec. 5.1. We accordingly select the minimum number of provisioned PS nodes in practice.

According to Eq. (17), we have $n_{wk} \leq u \cdot n_{ps}$. Moreover, as evidenced by Sec. 2, the cost-efficient provisioning for BSP balances the computation time and the communication time (*i.e.,* $t_{comm} \leq t_{comp}$). We accordingly have $n_{wk} \leq \sqrt{\frac{w_{iter} \cdot n_{ps} \cdot b_{ps}}{2 \cdot g_{param} \cdot c_{wk}}}$. Based on the above, we finally obtain the upper bound of n_{wk} as

$$n_{wk}^{upper} = \left\lceil \min\left(u \cdot n_{ps}, \sqrt{\frac{w_{iter} \cdot n_{ps} \cdot b_{ps}}{2 \cdot g_{param} \cdot c_{wk}}}\right) \right\rceil. \tag{19}$$

(2) ASP training: As the training iterations are evenly distributed among the homogeneous workers, we first calculate the number of training iterations on each worker as

$$s = \left\lceil \frac{\beta_0}{l_g \cdot \sqrt{n_{wk}}} - \frac{\beta_1}{n_{wk}} \right\rceil. \tag{20}$$

As the target loss value l_g is commonly less than 1 while $n_{wk} \in \mathcal{Z}^+$ is larger than 1, we have $l_g \leq \sqrt{n_{wk}}$. Thus, Eq. (20) can be further reduced to $s \geq \frac{\beta_0 - \beta_1}{l_g \cdot \sqrt{n_{wk}}}$, where $\beta_0 > \beta_1$ as evidenced in Sec. 2. Meanwhile, as the iteration processing time is larger than the gradient computation time, we have $\frac{w_{iter}}{c_{wk}} \leq \frac{T_g \cdot l_g \cdot \sqrt{n_{wk}}}{\beta_0 - \beta_1}$ according to Constraint (9). We further calculate the lower bound of n_{wk} as

$$n_{wk}^{lower} = \left\lceil \left(\frac{w_{iter} \cdot (\beta_0 - \beta_1)}{c_{wk} \cdot T_g \cdot l_g} \right)^2 \right\rceil. \tag{21}$$

Similarly to BSP training, we use the lower bound of n_{wk} to obtain the minimum number of provisioned PS nodes as

$$n_{ps} = \left\lceil \frac{n_{wk}^{lower}}{r} \right\rceil. \tag{22}$$

By the minimum number of provisioned PS nodes in Eq. (22) and Constraint (11), the upper bound of n_{wk} is calculated as

$$n_{wk}^{upper} = \lceil r \cdot n_{ps} \rceil. \tag{23}$$

□