Machine Learning on Volatile Instances: Convergence, Runtime, and Cost Tradeoffs

Xiaoxi Zhang, *Member, IEEE*, Jianyu Wang, *Student Member, IEEE*, Li-Feng Lee, Tom Yang, Akansha Kalra, Gauri Joshi, *Member, IEEE*, and Carlee Joe-Wong, *Member, IEEE*

***Abstract*— Due to the massive size of the neural network models and training datasets used in machine learning today, it is imperative to distribute stochastic gradient descent (SGD) by splitting up tasks such as gradient evaluation across multiple worker nodes. However, running distributed SGD can be prohibitively expensive because it may require specialized computing resources such as GPUs for extended periods of time. We propose cost-effective strategies to exploit volatile cloud instances that are cheaper than standard instances, but may be interrupted by higher priority workloads. To the best of our knowledge, this work is the first to quantify how variations in the number of active worker nodes (as a result of preemption) affect SGD convergence and the time to train the model. By understanding these trade-offs between preemption probability of the instances, accuracy, and training time, we are able to derive practical strategies for configuring distributed SGD jobs on volatile instances such as Amazon EC2 spot instances and other preemptible cloud instances. Experimental results show that our strategies achieve good training performance at substantially lower cost.**

***Index Terms*— Machine learning, stochastic gradient descent, volatile cloud instances, bidding strategies.**

I. Introduction

STOCHASTIC gradient descent (SGD) is the core algorithm used by most state-of-the-art machine learning (ML) problems today [3]–[5]. Yet as ever more complex models are trained on ever larger amounts of data, most SGD implementations have been forced to distribute the task of computing gradients across multiple "worker" nodes, thus reducing the computational burden on any single node while speeding up the model training through parallelization. Currently, even distributed training jobs require high-performance computing infrastructure such as GPUs to finish in a reasonable amount of time. However, purchasing GPUs outright is expensive and requires intensive setup and maintenance. Renting such machines as on-demand instances from services like Amazon EC2 can reduce setup costs, but may still be prohibitively expensive since distributed training jobs can take hours or even days to complete.

A common way to save money on cloud instances is to utilize *volatile*, or *transient*, instances, which have lower prices but experience interruptions [6]–[8]. Examples of such instances include Google Cloud Platform's preemptible instances [7] and Azure's low-priority virtual machines [8]; both give users access to virtual machines that can be preempted at any time, but charge a significantly lower hourly price than on-demand instances with availability guarantees. Amazon EC2's spot instances offer a similar service, but provide users additional flexibility by dynamically changing the price charged for using spot instances. Users can then specify the maximum price they are willing to pay, and they do not receive access to the instance when the prevailing spot price exceeds their specified maximum price [9]. Volatile computing resources may also be used to train ML jobs outside of traditional cloud contexts, e.g., in datacenters that run on "stranded power." Such datacenters only activate instances when the energy network supplying power to the datacenter has excess energy that needs to be burned off [10], [11], leading to substantial temporal volatility in resource availability. SGD variants are also commonly used to train machine learning models in edge or fog computing contexts, where resource volatility is a significant practical challenge [12]–[14].

SGD algorithms can be run on volatile instances by deploying each worker on a single instance, and deploying a parameter server on an on-demand or reserved instance that is never interrupted [15]. This deployment strategy, however, has drawbacks: since the workers may be interrupted throughout the training process, they cannot update the model parameters as frequently, increasing the error of the trained model compared to deploying workers on on-demand instances. Compensating for this increased error would require either training the model for a larger number of iterations or increasing the number of provisioned workers, both of which will increase the training cost. In this paper, we *quantify the performance trade-offs between error, cost, and training time for volatile instances*. We then use our analysis to propose *practical strategies for optimizing these trade-offs* in realistic preemption environments. We first consider Amazon spot instances, for which users can indirectly control their preemptions by setting maximum bids, and derive the resulting optimal bidding strategies. We then derive the optimal number of iterations and workers when users cannot control their instances' preemptions, as in GCP's preemptible instances and

Manuscript received June 25, 2020; revised May 22, 2021; accepted August 11, 2021; approved by IEEE/ACM Transactions on Networking Editor J. Shin. Date of publication November 10, 2021; date of current version February 17, 2022. This work was supported in part by NSF under Grant CNS-1751075, Grant CNS-1909306, and Grant CCF-1850029; in part by the 2018 IBM Faculty Research Award; and in part by the Qualcomm Innovation Fellowship. A conference version of this paper appears at IEEE Conference on Computer Communications 2020 [DOI: 10.1109/INFOCOM41043.2020.9155448]. This is an extended version with new results especially on multiple variants of the SGD algorithm. Detailed proofs can be found in our technical report [2]. *(Corresponding author: Xiaoxi Zhang.)*

Xiaoxi Zhang is with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, Guangdong 510275, China (e-mail: zhangxx89@mail.sysu.edu.cn).

Jianyu Wang, Li-Feng Lee, Tom Yang, Akansha Kalra, Gauri Joshi, and Carlee Joe-Wong are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: jianyuw1@andrew.cmu.edu; gavin.lee@sv.cmu.edu; tom.yang@sv.cmu.edu; akanshak@andrew.cmu.edu; gaurij@andrew.cmu.edu; cjoewong@andrew.cmu.edu).

Digital Object Identifier 10.1109/TNET.2021.3112082

Azure's low-priority VMs. More specifically, this work makes the following contributions:

1) *Quantifying training error convergence with dynamic numbers of workers (Section III).* Using volatile instances that can be interrupted and may rejoin later presents a new research challenge: prior analyses of distributed SGD algorithms do not consider the possibility that the number of active workers will change over time. We derive new error bounds on the convergence of SGD methods when the number of workers varies over time and show that the bound is proportional to the expected reciprocal of the number of active workers.
2) *Deriving optimal spot bidding strategies (Section IV).* To the best of our knowledge, no works have yet explored bidding strategies for distributed machine learning jobs that consider the bidding's effect on error convergence and random iteration runtimes. We analyze a unique three-way trade-off between the cost, error, and training time, using which we can design optimal bidding strategies to control the preemptions of spot instances. For tractability, we focus on two scenarios where each worker submits the same bid, or one of two distinct bids, and then extend to more general bid types in special cases.
3) *Deriving the optimal number of workers (Section V).* For scenarios where users cannot control the preemption probability, we propose a general model to relate the number of provisioned workers to the expected reciprocal of the number of active workers, which can capture practical preemption distributions. Using this model, we then provide mathematical expressions to jointly optimize the number of provisioned workers and iterations. We also propose a strategy to dynamically adjust the number of provisioned workers, which can further improve the error convergence.
4) *Experimental validation on Amazon EC2 (Section VI).* We validate our results by running distributed SGD jobs analyzing the CIFAR-10 [16] dataset on Amazon EC2. We show that our derived optimal bid prices can reduce users' cost by 65% on real, and 62% on synthetic, spot price traces while meeting the same error and completion time requirements, compared with bidding a high price to minimize interruptions as suggested in [17]. Moreover, we implement our proposed dynamic strategy with an increasing number of workers over time and validate that it can reduce the cost and yield a better cost/completion time/error trade-off by: (i) adding workers later in the job and re-optimizing the bids according to the realized error and training time so far on Amazon spot instances [6], and (ii) exponentially increasing the number of provisioned workers and running for a logarithmic number of iterations on GCP preemptible instances [18].

II. Related Work

Our work is broadly related to prior works on convergence analysis for distributed machine learning, as well as exploiting spot instances to efficiently run computational jobs.

Distributed machine learning generally assumes that multiple workers send local computation results to be aggregated at a central server, which then sends them updated parameter values. The SGD algorithm [3], in which workers compute the gradients of a given objective function with respect to model parameters, is particularly popular. In SGD, workers individually compute the gradient over stochastic samples (usually a mini-batch [19]) chosen from data residing at each worker in each iteration. Recent work has attempted to limit device-server communication to reduce the training time of SGD and related models [12], [20]–[22], while others analyze the effect of the mini-batch size [19] or learning rate [23], [24] on SGD algorithms' training error. Bottou *et al.* [24] analyze the convergence of training error in SGD but do not consider the runtime per iteration. Dutta *et al.* [23] analyze the trade-off between the training error and the (wall-clock) training time of distributed SGD, accounting for stochastic runtimes for the gradient computations at different workers [25]. Our work is similar in spirit but focuses on volatile instances, introducing cost as another performance metric. We also go beyond [23], [24] to derive error bounds when the number of active workers changes in different iterations.

Utilizing spot and other transient cloud resources for computing jobs has been extensively studied. Zheng *et al.* [15] design optimal bids to minimize the cost of completing jobs with a pre-determined execution time and no deadline. Other works derive cost-aware bidding strategies that consider jobs' deadline constraints [26] or jointly optimize the use of spot and on-demand instances [27]. However, these frameworks cannot handle distributed SGD's dependencies between workers. Another line of work instead optimizes the markets in which users bid for spot instances. Sharma *et al.* [17] advocate bidding the price of an on-demand instance and migrating to VM instances in other spot markets upon interruptions. The resulting migration overhead, however, requires complex checkpointing and migration strategies due to SGD's substantial communication dependencies between workers, realizing limited savings [28]. Some software frameworks have been designed for running big data analytics on transient instances [29], but they do not include theoretical ML performance analyses.

III. Error and Runtime Analysis of Distributed SGD With Volatile Workers

The number of active computing nodes used for distributed SGD training affects the convergence of the training error versus the number of SGD iterations as well as the runtime spent per iteration. Unlike most previous works in the optimization theory literature, which focus only on error-versus-iterations convergence, we consider both these factors and analyze the true convergence of SGD with respect to the wall-clock time. Moreover, to the best of our knowledge this is the first work that presents an error and runtime analysis for volatile computing instances, which can result in a changing number of active workers during training.

We formally introduce distributed SGD in Section III-A. In Section III-B, we quantify how worker volatility adversely affects error convergence because having fewer active workers yields more noisy gradients. In Section III-C, we analyze the effect of worker volatility on the training runtime, which is affected in two opposing ways. A higher preemption probability results in longer "dead" time intervals with zero active workers. Although a lower preemption probability yields more active workers, it can increase synchronization delays in waiting for straggling nodes. This error and runtime analysis lays the foundation for subsequent results on bidding strategies that can dynamically control the availability of each individual worker node and the number of active worker nodes.

In Sections IV and V, we use our results on the error and runtime analysis from this section to minimize the cost of training a job, subject to constraints on the maximum allowable error and runtime. Our goal is to solve the optimization:

$$\begin{aligned} &\text{minimize : Expected total cost } \mathbb{E}[C] && (1)\\ &\text{st.: Expected training error } \mathbb{E}[\phi] \le \epsilon, && (2)\\ &\quad\text{Expected completion time } \mathbb{E}[\tau] \le \theta, && (3)\end{aligned}$$

where ϵ and θ denote the maximum allowed error and the (wall-clock) job completion time respectively.

A. Distributed SGD Primer

Most state-of-the-art machine learning systems employ Stochastic Gradient Descent (SGD) to train a neural network model so as to minimize the empirical risk function $G : \mathbb{R}^d \to \mathbb{R}$ over a training dataset $\mathcal{S}$, which is defined as

$$G(\mathbf{w}) \triangleq \frac{1}{|\mathcal{S}|}\sum_{s=1}^{|\mathcal{S}|} l(h(x_s,\mathbf{w}),y_s), \tag{4}$$

where the vector $\mathbf{w}$ denotes the model parameters (for example, the weights and biases of a neural network model), and the loss $l(h(x_s,\mathbf{w}),y_s)$ compares our model's prediction $h(x_s,\mathbf{w})$ to the true output y_s, for each sample (x_s,y_s).

The mini-batch SGD algorithm iteratively minimizes $G(\mathbf{w})$ by computing gradients of l over a small, randomly chosen subset of data samples $\mathcal{S}_j$ in each iteration j and updating $\mathbf{w}$ as per the update rule $\mathbf{w}_{j+1} = \mathbf{w}_j - \alpha_j g(\mathbf{w}_j)$. Here α_j is the (pre-specified) step size and $g(\mathbf{w}_j) = \sum_{s\in\mathcal{S}_j}\nabla l(h(x_s,\mathbf{w}_j),y_s)/|\mathcal{S}_j|$, the gradient computed using samples in the mini-batch $\mathcal{S}_j$.

Synchronous Distributed SGD. To further speed up the training, many practical implementations parallelize gradient computation by using the parameter server framework shown in Fig. 2 [23]. In this framework, there is a central parameter server and n worker nodes. Each worker has access to a subset of the data, and in each iteration each worker fetches the current parameters $\mathbf{w}_j$ from the parameter server, computes the gradients of $l(h(x_s,\mathbf{w}_j),y_s)$ over one mini-batch of its data, and pushes them to the parameter server. For fully-synchronous SGD as we elaborate below, the parameter server waits for gradients from all n workers before updating the parameters to $\mathbf{w}_{j+1}$ as per

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \frac{\alpha_j}{n}\sum_{i=1}^{n} g^{(i)}(\mathbf{w}_j), \tag{5}$$

where $g^{(i)}(\mathbf{w}_j)$ is the mini-batch gradient returned by the i^{th} worker. The updated $\mathbf{w}_{j+1}$ is then sent to all workers, and the process repeats. This gradient aggregation method is commonly referred to as synchronous SGD. Asynchronous gradient aggregation can reduce the delays in waiting for straggling workers, but causes staleness in the gradients returned by workers, which can give inferior SGD convergence [23].

Based on the gradient aggregation method used by the server, we consider three variants of synchronous SGD, as shown in Figure 1.

1. Fully-synchronous SGD: In each iteration j, the parameter server waits for all workers to finish processing their mini-batches and push the computed gradients, before updating the parameters $\mathbf{w}_{j+1}$ to all workers for the next iteration.

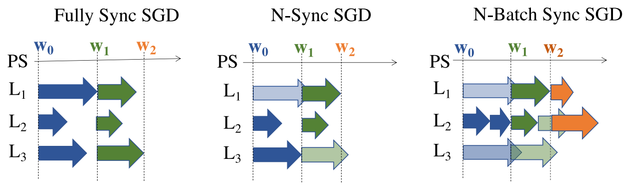


Fig. 1. Gradient computations on three workers for two iterations in fully, $N = 2$-, and $N = 2$-batch synchronous SGD. The x-axis indicates time, and lighter colored arrows indicate workers cancelled by the Parameter Server (PS).

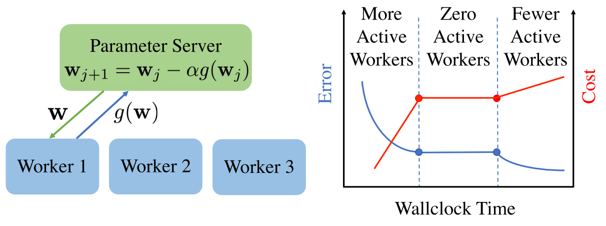


Fig. 2. Parameter Server Model and an illustration of how error and cost vary versus training time when the number of workers varies with time. Having more active workers results in a faster decrease in error, but a faster increase in cost.

2. N-synchronous SGD: In fully synchronous SGD, waiting for the slowest worker to finish its gradient computation can bottleneck an iteration's completion time. To overcome this bottleneck, in N-synchronous SGD the parameter server waits for the first N out of n workers to push their computed gradients. It updates the parameters to $\mathbf{w}_{j+1}$ according to these N gradients, and pushes $\mathbf{w}_{j+1}$ to all workers, canceling the outstanding gradient computations at slow workers.

3. N-batch-synchronous SGD: To further reduce synchronization delays, upon finishing a gradient computation, each worker continues to process another mini-batch using the same parameters $\mathbf{w}_j$ until N mini-batches are finished collectively by the workers. In contrast to N-synchronous SGD, the parameter server waits for the first N mini-batches to be finished rather than the first N workers. Once the parameter server receives N gradient updates, it cancels the remaining gradient computations, and updates $\mathbf{w}$ at all workers.

Distributed SGD on Volatile Workers. In this work we consider that the parameter server is run on an on-demand instance, while the n workers are run on volatile instances that can be interrupted or preempted during the training process, as illustrated in Fig. 2. Let y_j denote the number of active (i.e., not preempted) workers in iteration j, such that $0 < y_j \le n$ for all $j = 1,\ldots,J$, where J is the total number of iterations. The sequence $y_1, y_2, \ldots y_J$ can be considered as a random process. We do not count "iterations" where the number of active workers is 0, as there is then no gradient update. However, having zero workers will increase the total training completion time, which we will account for in Section III-C.

B. SGD Error Convergence With Variable Number of Workers

Since the number of active workers may vary over time, the number of gradient contributions used to update the parameters can be variable for the fully-synchronous SGD, while there are always a total of N gradient computes aggregated per iteration in a N-synchronous and N-batch-synchronous SGD. This fact leads to a more complex analysis of the error convergence for fully-synchronous SGD with dynamic workers. In the following, we first give an upper-bound on the expected training error in terms of $y_j,\ j=1,\dots J$ for fully-synchronous SGD (see Theorem 1); the error bounds for N-synchronous and N-batch-synchronous SGD directly follow. For error convergence analysis we make the following assumptions on the objective function G, which are common in most prior works on SGD convergence analysis [23], [24].

Assumption 1 (Lipschitz-Smoothness): The objective function $G(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}$ is L-Lipschitz smooth, i.e., it is continuously differentiable and there exists $L > 0$ such that

$$\| \nabla G(\mathbf{w}) - \nabla G(\mathbf{w}') \|_2 \leq L \| \mathbf{w} - \mathbf{w}' \|_2, \quad \forall \mathbf{w}, \mathbf{w}' \in \mathbb{R}^d \tag{6}$$

Assumption 2 (First and Second Moments): Let $\mathbb{E}_{\mathcal{S}_j}[\nabla G(\mathbf{w}_j, \mathcal{S}_j)]$ and $G(\mathbf{w})$ represent the expected gradient at iteration j for a mini-batch $\mathcal{S}_j$ of the training data and the gradient for the full data, respectively. Then there exist scalars $\mu_G \geq \mu > 0$ such that

$$\nabla G(\mathbf{w}_j)^T \mathbb{E}_{\mathcal{S}_j}[\nabla G(\mathbf{w}_j, \mathcal{S}_j)] \geq \mu \| \nabla G(\mathbf{w}_j) \|_2^2 \tag{7}$$

$$\text{and } \| \mathbb{E}_{\mathcal{S}_j}[\nabla G(\mathbf{w}_j, \mathcal{S}_j)] \|_2 \leq \mu_G \| \nabla G(\mathbf{w}_j) \|_2 \tag{8}$$

and scalars $M, M_V \geq 0$ with $M_G = M_V + \mu_G^2$ such that

$$\mathbb{E}_{\mathcal{S}_j}\left[\| \nabla G(\mathbf{w}_j, \mathcal{S}_j) \|_2^2\right] \leq M + M_G \| \nabla G(\mathbf{w}_j) \|_2^2 . \tag{9}$$

for any given size of mini-batch $\mathcal{S}_j$ on one worker.

Theorem 1 (Error Bound With Dynamic Active Workers): Suppose that the objective function $G(\cdot)$ satisfies Assumptions 1– 2 and is c-strongly convex [30] with parameter $c \leq L$. Assuming $G^* \geq 0$, for a fixed step size $0 < \alpha < \frac{\mu}{LM_G}$ and given $\mathbf{w}_1$, the expected training error, defined as the expected gap between the objective value of running a fully-synchronous SGD after J iterations and the optimum G^*, is:

$$\mathbb{E}[G(\mathbf{w}_{J+1}) - G^*] \leq (1-\alpha c\mu)^J \mathbb{E}[G(\mathbf{w}_1)] + \frac{\alpha^2 LM}{2} \sum_{j=1}^{J} (1-\alpha c\mu)^{J-j} \mathbb{E}\left[\frac{1}{y_j}\right]; \tag{10}$$

Taking $\mathbb{E}\left[\frac{1}{y_j}\right] = \frac{1}{N}$ in (10) gives the error bound for an N-synchronous SGD and N-batch-synchronous SGD.

The proof is given in the Appendix of our technical report [2], where we also extend Theorem 1 to handle non-convex objective functions and a diminishing step size, by analyzing the convergence speed to a stationary point instead of the error defined as $\mathbb{E}[G(\mathbf{w}_{J+1}) - G^*]$.

Discussion of the Assumptions. Assumption 1 and the c-strongly convexity assume a lower bound and an upper bound of $G(y)$ at any point y respectively, meaning $G(\cdot)$ is a function that does not change its value too fast or too slowly, as these two assumptions lead to: $G(x) + \nabla G(x)(y-x) + c/2||x-y||_2^2 <= G(y) <= G(x) + \nabla G(x)(y-x) + L/2||x-y||_2^2$, with $0 < c \leq L$ and any given two points x, y in the feasible region of $G(\cdot)$. Assumption 2 first states in (7) and (8) that, in expectation, the vector $-\nabla G(w_j, \mathcal{S}_j)$ is a direction of sufficient descent for $G(\cdot)$ from $\mathbf{w}_j$ with a norm comparable to the norm of the gradient. In particular, a special case $\mu_G = \mu = 1$ means tha $\nabla G(w_j, \mathcal{S}_j)$ is an unbiased estimate of the full gradient $\nabla G(w_j)$. It then assumes in (9) that the variance of $\nabla G(w_j, \mathcal{S}_j)$ is restricted, but is allowed to increase linearly with the squared norm of the full gradient.

Our Theorems 1 and 2 also assume i.i.d. workers' mini-batches. It generally holds when each worker's mini-batches are i.i.d. drawn from a portion of data partitioned from the original dataset uniformly at random, but might not hold if each portion is large and fixed during the training. One method to address this is to establish a shared "data lake" from which workers draw their mini-batches in an i.i.d. manner in each epoch, which would reduce the storage gains from parallelizing the data storage across multiple workers though. We leave the implementation of the data distribution to ensure this i.i.d. assumption into our future work.

Remark 1 (Penalty for Using Volatile Instances): The error bound in Theorem 1 given the expected number of active workers $\mathbb{E}[y_j]$ is minimized when y_j is not a random variable, i.e., SGD is run on on-demand instead of volatile instances. This result follows from the convexity of y_j^{-1}; using Jensen's inequality we can show that fixing the number of active workers to $y = \mathbb{E}[y_j]$ minimizes $\mathbb{E}\left[y_j^{-1}\right]$.

Remark 2 (Error and Preemption Probability): Suppose that a worker is preempted with probability q in each iteration. Then the bound in Theorem 1 increases with q because $\mathbb{E}[1/y_j]$ increases with q. Thus, more frequent preemption or interruption of workers yields worse error convergence.

Dynamic Numbers of Provisioned Workers for Fully-Synchronous SGD. While Theorem 1 gives us the error bound for a fixed number (n) of provisioned workers over iterations, ML practitioners often increase the number of workers over time [31]–[33]. Intuitively, in the later training stages, the parameter values are closer to convergence, and thus it is crucial that the gradient updates are accurate, i.e., averaged over a larger number of worker mini-batches. More formally, Theorem 1 shows that $\mathbb{E}\left[\frac{1}{y_j}\right]$'s contribution to the error bound increases exponentially with j by $\frac{1}{1-\alpha c\mu}$. This observation motives us to increase the number of provisioned workers over iterations so that $\mathbb{E}\left[\frac{1}{y_j}\right]$ will decrease and the error decay is further improved. Having smaller numbers of active workers in the early iterations (for which $j \ll J$) can further reduce the incurred cost, without affecting the error much due to the smaller contribution of $\mathbb{E}\left[\frac{1}{y_j}\right]$ to the error bound. In fact, guided by the above intuition, we have the following improved error bound when the number of provisioned workers exponentially increases with iterations.

Theorem 2 (Error With Increasing Provisioned Workers): Suppose n_j, the number of provisioned workers in iteration j, satisfies $n_j = \lceil n_1 \eta^{j-1} \rceil$ for some $n_1 > 0, \eta > 1$, and the number of active workers y_j satisfies $\mathbb{E}\left[\frac{1}{y_j}\right] \leq \frac{d}{n_j^\chi}$ for some $\chi \geq 0, d > 0$. Then for any $\alpha \leq \frac{\mu}{LM_G}$, the expected training error of running fully-synchronous SGD after J iterations is:

$$\mathbb{E}[G(\mathbf{w}_{J+1}) - G^*] \leq (1-\alpha c\mu)^J \mathbb{E}[G(\mathbf{w}_1)] + \frac{B}{n_1^\chi} \cdot (1-\alpha c\mu)^{J-1} \cdot \frac{1-x^J}{1-x}, \tag{11}$$

where $x = \frac{1}{\eta^\chi(1-\alpha c\mu)}$ and $B = \frac{\alpha^2\ LMd}{2}$.

Theorem 2 shows that the error bound can converge to 0 asymptotically with J if $\eta^\chi \geq \frac{1}{1-\alpha c\mu}$, in contrast to converging to a positive constant when using a static number of workers as shown in Theorem 1. It also implies that for any given increasing rate of the number of provisioned workers $\eta > 1$, there always exists a step size α such that the error bound decreases in the order of $\frac{1}{\eta^\chi}$.

One can similarly exponentially increase the batch size of each worker while using the same number of workers over iterations [34], but doing so will exponentially increase the runtime of each iteration. We prove in Corollary 1 that our dynamic strategy achieves the same error convergence rate and a better asymptotic error bound with a significantly smaller number of iterations than using a static number of workers.

Corollary 1 (Reduced Iterations With Extra Workers): Suppose the number of active workers y_j satisfies $\mathbb{E}\left[\frac{1}{y_j}\right] \leq O\left(\frac{1}{n_j^\chi}\right)$ for some $\chi \geq 0$. Then for any $\eta > 1$ and J sufficiently large, provisioning $\lceil n_1\eta^{j-1}\rceil$ workers in iteration j and running fully-synchronous SGD for $\lceil \log_{\eta^\chi}(1+(\eta-1)J)\rceil$ iterations can achieve an error bound no larger than provisioning n_1 workers for J iterations.

Remark 3 (Bounded Number of Provisioned Workers): If there is an upper limit of the number of provisioned workers, denoted by n_{max}, then the error bound (11) will be:

$$\mathbb{E}\left[G(\mathbf{w}_{J+1}) - G^*\right] \leq (1-\alpha c\mu)^J \mathbb{E}\left[G(\mathbf{w}_1)\right] + B\cdot\max\left\{\frac{(1-\alpha c\mu)^{J-1}1-x^J}{n_1^\chi(1-x)}, \frac{1-(1-\alpha c\mu)^J}{n_{max}^\chi \alpha c\mu}\right\} \quad (12)$$

where $x = \frac{1}{\eta^\chi(1-\alpha c\mu)}$ and $B = \frac{\alpha^2\ LMd}{2}$. It means that the error will converge to $\frac{\alpha LMd}{2c\mu n_{max}^\chi}$, instead of zero. One can verify that the asymptotic error in this case is equal to always using n_{max} provisioned workers each iteration (setting $\mathbb{E}\left[\frac{1}{y_j}\right] = \frac{d}{n_{max}^\chi}$ in Theorem 1), but clearly will consume much less cost as there are much fewer workers used in the early iterations.

Note that Theorem 2 and Corollary 1 hold only for fully-synchronous SGD, as a greater number of gradient updates are gained from having more active workers in each iteration, which can decrease the variance of the gradients computed from random mini-batches. The error convergence rates of N-synchronous and N-batch-synchronous SGDs, in contrast, are not improved by increasing the number of provisioned workers due to constantly processing N mini-batches per iteration, but the runtime can be reduced which we will discuss in Section III-C. To achieve a better convergence by increasing the number of workers, we can simply modify these two SGD variants to be N_j-(batch)-synchronous, with an increasing N_j.

Remark 4 (Theorem 2 Adaptation for SGD Variants): Theorem 2 can be applied to the N-synchronous and N-batch-synchronous SGD variants if we set the number of provisioned workers as required by Theorem 2 and modify N, the number of mini-batches to finish in iteration, so that it exponentially increases with the number of iterations, *e.g.*, $N_j = \bar{n}\bar{\eta}^{j-1}$ where $1 \leq \bar{n} \leq n_1, 1 < \bar{\eta} \leq \eta$. Similar to (11), the expected training error after J iterations is at most

$$(1-\alpha c\mu)^J\mathbb{E}\left[G(\mathbf{w}_1)\right] + \frac{B}{\bar{n}}\cdot(1-\alpha c\mu)^{J-1}\cdot\frac{1-x^J}{1-x}, \quad (13)$$

where $x = \frac{1}{\bar{\eta}(1-\alpha c\mu)}$ and $B = \frac{\alpha^2\ LMd}{2}$.

Remark 5 (Step Size Choice for Dynamic Workers): Theorem 2 can generalize to allowing a dynamic step size α_j that varies with the number of iterations $j = 1,\cdots,J$. We can show that a static step size can achieve the fastest error decay rate, due to the fact (shown formally in the Appendix of our technical report [2]) that the error bound adapted from (11) will still consist of a term that is proportional to $\mathbb{E}\left[G(\mathbf{w}_1)\right]$ and a term proportional to the variance bound (cf. the two terms in (11)). Moreover, the error bound is dominated by $\prod_{j=1}^J(1-\alpha_j c\mu)$ as in the first term since the maximum variance per iteration is diminishing with $\mathbb{E}\left[\frac{1}{y_j}\right]$ due to the exponentially increased number of provisioned workers. We can finally show that a static step size $0 < \alpha < \frac{1-\frac{1}{\eta^\chi}}{c\mu}$ can minimize the error bound.

C. SGD Runtime Analysis With Volatile Workers

Now let us analyze how using volatile workers affects the training runtime. The runtime has two components: 1) the time required to complete the J SGD iterations, and 2) the idle time when no workers are active and thus no iterations can be run.

Let $R(y_j)$ denote the runtime of the j^{th} iteration in which we have the set $\mathcal{Y}_j$ of y_j active workers. Suppose each worker takes time r_k to compute its gradient, where r_k is a random variable, for $k = 1,\cdots,y_j$. Fluctuations in computation time are common especially in cloud infrastructure due to background processes, node outages, network delays etc. [35]. Since the parameter server has to wait for all y_j workers to finish their gradient computations in a fully-synchronous SGD, the runtime per iteration is,

$$R(y_j) = \max_{k\in\mathcal{Y}_j} r_k + \Delta, \quad (14)$$

where Δ is the time taken by the parameter server to update $\mathbf{w}$ and push it to the y_j workers. The expected runtime $\mathbb{E}\left[R(y_j)\right]$ increases with the number of active workers. For example, if $r_k \sim \exp(\mu)$, an exponential random variable that is i.i.d. across workers and mini-batches, then $\mathbb{E}\left[R(y_j)\right] \approx (\log y_j)/\mu + \Delta$. For N-synchronous SGD, we have

$$R(y_j) = r_{(N)} + \Delta, \quad (15)$$

for any given $y_j \geq N$, where $r_{(N)}$ is the Nth order statistic of random variables $\{r_k\}_{k\in\mathcal{Y}_j}$, which is the largest runtime of N fastest workers among the active ones. In contrast to the fully-synchronous counterpart, $\mathbb{E}\left[R(y_j)\right]$ in (15) decreases with y_j and we have $\mathbb{E}\left[R(y_j)\right] \approx (\log\frac{y_j}{y_j-N})/\mu + \Delta$ if $r_k \sim \exp(\mu)$. For a N-batch-synchronous SGD, the general form of $R(y_j)$ is intractable, but for $r_k \sim \exp(\mu)$, it is equivalent to modeling the arrival of the gradient update of each mini-batch as an independent Poisson process with rate μ running in parallel, and thus the total arrival rate of y_j workers equals μy_j for any given y_j. Since we aggregate N gradient updates each iteration, we have $\mathbb{E}\left[R(y_j)\right] \approx \frac{N}{y_j\mu}$.

Adding this per-iteration runtime to the idle time when no workers are active, we can show that the expected wall-clock time required to complete the J SGD iterations is

$$\mathbb{E}\left[\tau\right] = \sum_{j=1}^J \mathbb{E}\left[R(y_j)\right] + \mathbb{E}\left[\text{idle time with no active workers}\right].$$

We assume that if workers are interrupted during any iteration i, we can store the intermediate results of the gradient computations when the workers receive the interruption notifications. If iteration $i+1$ starts before the workers resume from the interruptions, the stored intermediate gradients will be discarded, which can happen in the following cases: (1) for fully-sync SGD, at least one worker was active and has finished the mini-batch for iteration i; (2) for N-sync and N-batch-sync SGD, N mini-batches (from N active workers or any $y_j \leq N$ workers) have been finished for iteration i. Otherwise, the resumed workers continue the gradient computation based on the stored results as they are still in iteration i. For example, when each worker is preempted uniformly at random with probability q in each iteration (as described in Remark 2), then the expected completion time becomes $\mathbb{E}[\tau] = \sum_{j=1}^{J} \mathbb{E}[R(y_j)]/(1-q^n)$. In fact, the ratio of the total time that the job is running to the total completion time equals the probability that at least one worker is active, if this probability is static over time. This observation leads to Lemma 1 in Section IV.

IV. Optimizing Spot Instance Bids

In this section, we use the results of Section III to derive the bid prices and number of iterations that minimize the cost of running distributed SGD with workers placed on spot instances. We first consider the simple case in which we submit the same bid for each worker in Section IV-A and then consider the heterogeneous bid case in Section IV-B.

Spot Price and Bidding Model. Let p_t denote the spot price of each instance at time t. We assume p_t is i.i.d. and is bounded between a lower-bound $\underline{p}$ and an upper-bound $\bar{p}$, similar to prior works on optimal bidding in spot markets [15]. Let $f(\cdot)$ and $F(\cdot)$ denote the probability density function (PDF) [36] and the cumulative density function (CDF) [37] of p_t. When a bid b is placed for an instance, we consider that the provider assigns available spot capacity to users in descending order of their bids, stopping at users with bids below the prevailing spot price. Thus, a worker is active only if its bid price exceeds the current spot price. Hence, without loss of generality the range of the bid price can also be assumed to be $\underline{p} \leq b \leq \bar{p}$. Whenever a worker is active ($b \geq p_t$), the per-time cost incurred for running it is equal to the prevailing spot price p_t (not the bid price). According to AWS's policy for spot instances [6], the bids are placed before instances are launched and cannot be changed after that; therefore, we assume that each worker's bid is fixed throughout the model training.

A. Identical Worker Bids

Suppose we choose bid price b for each of the n provisioned workers. We first simplify the error and runtime in Section III for this case, and then solve the cost minimization problem (1)-(3). Our results hold for all possible distributions of worker running times (Theorem 3) and all our considered SGD variants by using their respective error bounds (Theorem 1) and expected runtime per iteration (Section III-C).

Observe that the n provisioned workers are either all available or all interrupted depending on the bid price b. This insight implies that $\mathbb{E}\left[y_j^{-1}\right] = 1/n$, and thus that the error bound in Theorem 1 is *independent of the bid* b: this bid affects only the frequency with which iterations are executed, not the number of active workers in an iteration. We can thus rewrite the error bound (10) as a function of J, the number of iterations required to reach error ϵ. Formally, we set $\hat{\phi}$ to be the right-hand side of (10) and $J \geq \hat{\phi}^{-1}(\epsilon)$, where $\hat{\phi}^{-1}(\epsilon)$ is the number of iterations required to ensure that (our upper bound on) the expected error is no larger than ϵ.

We further observe that, the number of active workers y_j always equals n when the job is running. Thus, the expected runtime per iteration can be rewritten as $\mathbb{E}[R(y_j)] = \mathbb{E}[R(n)]$. Accounting for the idle time we can show that the expected completion time is monotonic with b:

Lemma 1 (Completion Time in Terms of Bid Price): Using the same bid price b for all workers, the expected completion time to complete J iterations of synchronous SGD is

$$\mathbb{E}[\tau] = J\mathbb{E}[R(n)]/F(b), \tag{16}$$

which increases with J and is non-increasing in the bid price b. The function $F(\cdot)$ is the CDF of the spot price.

We can further show the expected cost (defined in (1)) is monotonically non-decreasing with b and J.

Lemma 2 (Cost in Terms of Bid Price): Using one bid price for all workers, the expected cost of finishing a synchronous SGD job is given by

$$\mathbb{E}[C] = Jn\mathbb{E}[R(n)]\left(\underline{p} + \int_{\underline{p}}^{b}\left(1 - \frac{F(p)}{F(b)}\right)dp\right), \tag{17}$$

which is non-decreasing in the bid price b and J. The function $F(\cdot)$ is the CDF of the spot price.

Since both $\mathbb{E}[\tau]$ and $\mathbb{E}[C]$ increase with J, we should set J to be equal to $\hat{\phi}^{-1}(\epsilon)$ in order to reach the target error in the minimum time and at the minimum cost.

Optimizing the Bid Price. Having shown that $J = \hat{\phi}^{-1}(\epsilon)$, we now find the optimal bid b that minimizes the expected cost (17) to solve the optimization problem (1)–(3).

According to Amazon's policy [6], b is determined upon the job submission without knowing the future spot prices and will be fixed for the job's lifetime. Although the user can effectively change the bid price by terminating the original request and re-bidding for a new VM, doing so induces significant migration overhead. Thus, we assume that users employ persistent spot requests: a worker with a persistent request will be resumed once the spot price falls below its bid price, exiting the system once its job completes. Using Lemma 1 and Lemma 2, we can show the following theorem for the optimal bid price b.

Theorem 3 (Optimal Uniform Bid): When we make an identical bid b for n workers and use them to perform distributed synchronous SGD to reach error ϵ within time θ, the optimal bid price that minimizes the cost is $b^* = F^{-1}\left(\frac{\hat{\phi}^{-1}(\epsilon)\mathbb{E}[R(n)]}{\theta}\right)$.

Theorem 3 provides a general form of the optimal bid price, given the number of workers per iteration, n, the deadline θ, and the target error bound ϵ, for any distributions of the spot price and training runtime per iteration. In particular, it holds for all our considered SGD variants by using their respective $\mathbb{E}[R(n)]$ as discussed in Section III-C.

B. Optimal Heterogeneous Bids

We next extend our results from Section IV-A to find the optimal bidding strategy allowing different bids for different workers. This strategy is motivated by the observation that bidding lower prices for some workers yields a larger number of active workers when the spot price is relatively low, which possibly improves the training error but may not cost much.

To solve for the closed-form expression of the optimal bids from this optimization problem, we first examine two distinct bid prices b_h and b_l for two groups of workers for a fully-synchronous SGD with arbitrary runtime distribution in Section IV-B1, and then derive the optimal bids when allowing a more flexible choice of bids for N-synchronous and N-batch-synchronous SGD with exponential runtime distribution in Sections IV-B2 and IV-B3.

1) Fully-Synchronous SGD: Formally, we place bids of b_h for workers $1,\cdots,n_h$ and b_l ($< b_h$) for workers $n_h+1,\cdots,n$. We define the random variable $y(\vec{b}) \in \{n_h, n\}$ as the number of active workers when the bid prices are $\vec{b} = (b_h, b_l)$; y_j is then a realization of $y(\vec{b})$. Recall that the times when 0 workers are active are not considered as an SGD 'iteration'. Thus, $y(\vec{b})$ can only be either n_h (with probability $\frac{F(b_h)-F(b_l)}{F(b_h)}$) or n (with probability $F(b_l)/F(b_h)$) in each iteration.

Optimized Bids. We initially assume that n_h, the number of workers in the first group, and J, the required number of iterations, are fixed; thus, we optimize the trade-off between the expected cost, expected completion time, and the expected training error using only the bid prices $\vec{b}$. After deriving the closed-form optimal solutions of b_h and b_l in Theorem 4, we discuss co-optimizing n_h and J with the bids $\vec{b}$. The expected cost minimization problem (1)–(3) then becomes:

$$\min_{\vec{b}} J\int_{\underline{p}}^{b_h} \mathbb{E}\left[R(\vec{b},p)\right] y(\vec{b}) p \frac{f(p)}{F(b_h)} \mathrm{d}p \tag{18}$$

$$\text{subject to: } \mathbb{E}\left[\hat{\phi}(\vec{b})\right] \le \epsilon \quad \text{(Error constraint)} \tag{19}$$

$$\frac{J}{F(b_h)}\int_{\underline{p}}^{b_h} \mathbb{E}\left[R(\vec{b},p)\right] \frac{f(p)}{F(b_h)} \mathrm{d}p \le \theta \tag{20}$$

$$\bar{p} \ge b_h \ge b_l \ge \underline{p}, \quad \forall i \le j \tag{21}$$

To derive the cost and completion time expressions in (18) and (20) respectively, we express the expected runtime of iteration j as $\mathbb{E}\left[R(\vec{b},p)\right]$, a function of the bids and price. For simplicity, we assume that the spot prices do not change within each iteration. In practice, the spot price changes at most once per hour [38], compared to a runtime of several minutes per iteration, and thus this assumption usually holds. Note that we did not need this assumption for the identical bid case in Section IV-A since all workers become active/inactive at the same time with a uniform bid.

To derive the optimal bid prices, we first relate the distribution of the spot price and our bid prices to the training error through the number of active workers, *i.e.*, $y(\vec{b})$. From Theorem 1, the expected error is at most ϵ if $y(\vec{b})$ satisfies:

$$\mathbb{E}\left[\frac{1}{y(\vec{b})}\right] \le \frac{2c\mu\left(\epsilon - (1-\alpha c\mu)^J \mathbb{E}\left[G(\mathbf{w}_1)\right]\right)}{\alpha LM\left(1-(1-\alpha c\mu)^J\right)} \triangleq Q(\epsilon) \tag{22}$$

Further, we simplify $\mathbb{E}\left[R(\vec{b},p)\right]$ to be function $\mathbb{E}\left[R(X)\right]$, the expected runtime per iteration given X workers are active. We then provide closed-form expressions for the optimal bid prices through Theorem 4.

Theorem 4 (Optimal-Two Bids With a Fixed J): Suppose the objective function $G(\cdot)$ satisfies Assumptions 1–2. Given a number of iterations (J) and maximum allowed error (ϵ) that can guarantee $1/n < Q(\epsilon) \le 1/n_h$ ($Q(\epsilon)$ is defined as the right-hand side of (22)),[1] *a fixed step size α, and a feasible deadline ($\theta \ge J\mathbb{E}\left[R(n)\right]$), we have the optimal bid prices b_h^* and b_l^*:*

$$F(b_h^*) = \frac{J}{\theta}\left(\left(\mathbb{E}\left[R(n)\right] - \mathbb{E}\left[R(n_h)\right]\right)\frac{\frac{1}{n_h} - Q(\epsilon)}{\frac{1}{n_h}-\frac{1}{n}} + \mathbb{E}\left[R(n_h)\right]\right)$$

$$F(b_l^*) = \frac{\frac{1}{n_h} - Q(\epsilon)}{\frac{1}{n_h}-\frac{1}{n}} \times F(b_h^*), \tag{23}$$

for any i.i.d. spot prices and any i.i.d. runtime per mini-batch, i.e., *$F(\cdot)$ and $\mathbb{E}\left[R(n)\right]$ (or $\mathbb{E}\left[R(n_h)\right]$) do not change during the training process.*

For brevity, we use Figure 3 to illustrate our proof of Theorem 4. The key steps are: (i) change the variables of the optimization problem (18) to be $F(b_h)$ and $\gamma = \frac{F(b_l)}{F(b_h)}$; (ii) show that the expected cost, completion time, and error are monotonic w.r.t. to $F(b_h)$ and γ. Intuitively, the expected error should depend only on the number of active workers *given that some workers are active*, which is controlled by the relative difference between $F(b_h)$ and $F(b_l)$: γ. Formally, the error bound decreases with $\mathbb{E}\left[y(\vec{b})^{-1}\right]$. Applying $\mathbb{E}\left[y(\vec{b})^{-1}\right] = \frac{1}{F(b_h)}\left(\frac{F(b_h)-F(b_l)}{n_h} + \frac{F(b_l)}{n}\right) = \frac{1}{n_h} - \frac{1}{\gamma}\left(\frac{1}{n_h}-\frac{1}{n}\right)$ to (22) gives us the optimal γ, since the expected cost increases with both $F(b_h)$ and γ. We then choose $F(b_h^*)$ to the one that yields $\mathbb{E}\left[\tau\right] = \theta$ (making constraint (20) tight). Intuitively, $F(b_h^*)$ should be high enough to guarantee that some workers are active often enough that the job completes before the deadline.

Co-optimizing n_h and $\vec{b}$. If n_h is not a known input but a variable to be co-optimized with $\vec{b}$, we can write n_h and b_l^* in terms of $F(b_h^*)$ according to (23) and plug them into (18)-(21) to solve for b_h^* first, and then derive b_l^* and the optimal n_h.

Co-optimizing J and $\vec{b}$. Taking J as an optimization variable may allow us to further reduce the job's cost. For instance, allowing the job to run for more iterations, *i.e.*, increasing J, increases $Q(\epsilon)$ (the right-hand side of (22)). We can then increase $\mathbb{E}\left[\frac{1}{y(\vec{b})}\right]$ by submitting lower bids b_l, making it less likely that workers $n_h+1,\ldots,n$ will be active, while still satisfying (22). A lower b_l may decrease the expected cost by making workers less expensive, though this may be offset by the increased number of iterations. To co-optimize J, we show it is a function of $\vec{b}$ and ϵ:

Corollary 2 (Relationship of J and $\vec{b}$): To guarantee a training error $\le \epsilon$, the number of iterations J should be at least

$$J = \log_{(1-\alpha c\mu)} \frac{\epsilon - \frac{\alpha LM}{2c\mu}\mathbb{E}\left[\frac{1}{y(\vec{b})}\right]}{\mathbb{E}\left[G(\mathbf{w}_1)\right] - \frac{\alpha LM}{2c\mu}\mathbb{E}\left[\frac{1}{y(\vec{b})}\right]}. \tag{24}$$

For brevity, we show the idea of co-optimizing J and $\vec{b}$: We first replace J in (18) and (20) by (24). Constraint (19) is already guaranteed by (24) and can be removed. We then solve for the remaining optimization variables, the bids $\vec{b}$.

[1]If ϵ and J are so large such that $Q(\epsilon) > 1/n_h$, it means that using n_h workers when the job is running is sufficient to achieve the training error (trivial setting); else if $Q(\epsilon) < 1/n$, it means that even having n workers all active whenever the job is running cannot satisfy the error bound requirement (infeasible setting). We omit these two problem settings.

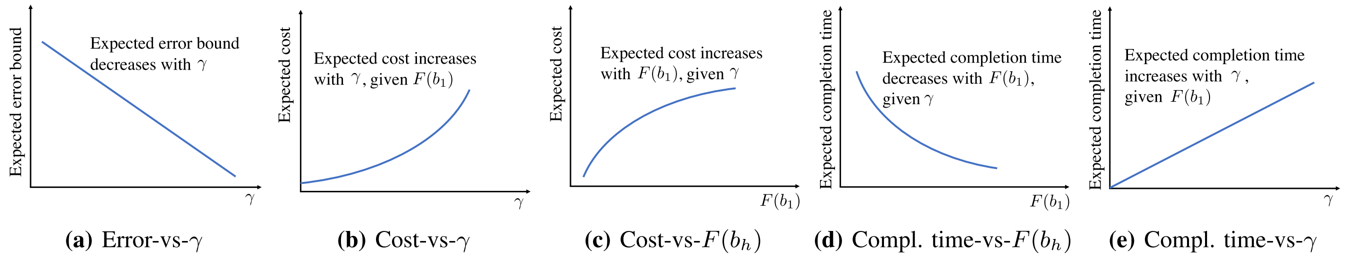


Fig. 3. Illustration of how the expected cost, completion time and error vary w.r.t. $F(b_h)$ and $\gamma = \frac{F(b_l)}{F(b_h)}$. As a larger γ leads to a smaller expected error (Fig. 3a) but a larger expected cost (Fig. 3b) and completion time (Fig. 3e), and the expected error is only controlled by γ, the optimal γ should be the smallest possible γ, *i.e.*, the one that yields error $= \epsilon$. The optimal $F(b_h)$ should be the one that yields the completion time equal to the deadline under the optimal γ (Fig. 3d).

2) N-Synchronous SGD: We now consider N-synchronous SGD, where in each iteration, the parameter server waits for N $(< n)$ workers. For this SGD variant, we guarantee that at least N workers are active when the job is running ($p_t \le b_1$), avoiding cases where fewer than N workers are active at the start of the iteration and the algorithm must wait to start the next iteration until the spot price falls sufficiently far that more workers become active. Hence, we suppose that $N+1$ workers have the same bid price b_1, and that the bid prices for workers $N+2, \cdots, n$ are $b_2, \cdots, b_{n-N}$, respectively with $b_1 \ge b_2 \ge \ldots \ge b_{n-N}$. Thus, $N \le y(p_t, \vec{b}) \le n$ when $\underline{p} \le p_t \le b_1$.

Since we allow a distinct bid price for each remaining worker, the above model is harder to analyze than the two bid prices for two groups of workers adopted in Theorem 4. To derive the closed-form of the optimal bid prices, we simplify the problem by assuming that the running time per mini-batch is randomly drawn from an exponential distribution, as assumed in [23], [39]. Surprisingly, we can prove that the optimal bid prices should be the same for all workers:

Theorem 5 (N-Sync SGD With Exponential Runtimes): Suppose the running time per iteration is i.i.d. drawn from an exponential distribution $\exp(\lambda)$ *over all time and all workers and the number of iterations* $J = \hat{\phi}^{-1}(N, \epsilon)$. *Then the optimal bid prices are*:

$$b_i = b_j = F^{-1}\left(\frac{\hat{\phi}^{-1}(\epsilon, N)\log\frac{n}{n-N}}{\lambda\theta}\right).$$

Intuitively, since N-synchronous SGD always processes N mini-batches per iteration, having more workers active when the spot price is low does not change the error. It does, however, increase the cost due to paying for all workers' computing usage. If the mini-batch runtimes are exponentially distributed, this is not offset by the lower iteration runtime due to canceling stragglers. It is then optimal to have the same number of active workers in each iteration, *i.e.*, uniform bids.

Proof Sketch: While the formal proof can be found in the Appendix of our technical report [2], we provide the outline of the proof here. The basic idea to prove Theorem 5 consists of three parts: 1) the expected runtime per iteration and the expected completion time will be non-increasing with $\{b_i\}_{i=1}^n$; 2) the expected cost is non-decreasing with the highest bid price b_1 while 3) non-increasing with all the other (lower) bid prices. The third part can be intuitively explained: if any $b_i (2 \le i \le n-N)$ increases, the probability of having only workers $1, \cdots, N+i-1$ active is non-increasing while the probability of having workers $1, \cdots, N+i$ active is non-decreasing; and the potential decrease in cost due to the former probability change is at least the increase in cost due to the latter given the particular form of the expected runtime per iteration and workers in this scenario. ■

3) N-Batch-Synchronous SGD: We finally consider N-batch-synchronous SGD, where the parameters are updated and a new iteration begins once N mini-batches are processed. We consider a more general bidding model where each worker is allowed to have a distinct bid price, denoting the bid prices for workers $1, \cdots, n$ as $b_1 \ge b_2 \ge \cdots \ge b_n$.

As in Section IV-B2, we assume that the wall-clock time of finishing each mini-batch for all workers is an i.i.d. random variable drawn from an exponential distribution with parameter λ. Unlike N-synchronous SGD, for N-batch-synchronous SGD, workers can continuously process mini-batches in each iteration, until a total of N mini-batches are finished. Therefore, the arrival rate of mini-batch completion at each time equals $y(p_t, \vec{b})\lambda$. We then observe that *the product of the number of active workers* $y(p_t, \vec{b})$ *and the running time* $\frac{1}{y(p_t, \vec{b})\lambda}$ *will be* $\frac{1}{\lambda}$, *a constant*. This product is the *Resource-time*, which is the total amount of resource units in each iteration we need to pay for. Since it is a constant, the expected cost of each iteration will be determined only by the spot price distribution, the largest bid price b_1, and λ. This result implies that the optimal bid prices are the same for all workers:

Theorem 6 (N-Batch-Sync SGD With Exponential Runtime): Given a deadline θ and maximum error threshold ϵ, suppose the runtime per iteration follows an exponential distribution $\exp(\lambda)$ *for all workers. Then the optimal cost-minimizing bid prices equal* $F^{-1}\left(\frac{N\hat{\phi}^{-1}(\epsilon, N)}{n\lambda\theta}\right)$ *for each worker.*

The proof of Theorem 6 is given in the Appendix of [2].

Theorem 6 indicates that a smaller bid price comes with a larger number of workers (n), a larger deadline (θ), and a smaller number of required mini-batches ($N\hat{\phi}^{-1}(\epsilon, N)$) to achieve a smaller-than-ϵ error. Further, it also implies that as we increase the number of workers, n, the expected cost keeps decreasing until it becomes $\frac{N\hat{\phi}^{-1}(\epsilon, N)}{\lambda} \times \underline{p}$. This non-intuitive conclusion follows from the exponentially distributed running times per mini-batch, under which the expected running time per iteration converges to zero as $n \to \infty$.

V. Optimal Number of Preemptible Instances

In this section, we consider preemptible instances offered by other cloud platforms, *e.g.*, low priority VMs from Microsoft Azure [8] and preemptible instances from Google Cloud Platform [7]. Unlike spot instances where users can specify the maximum prices they are willing to pay, on these platforms users can only decide the number of provisioned instances to request in each iteration, as well as the number of iterations.

Therefore, in this section, we choose to optimize the number of instances (workers) and assume the instance price is stable during the entire training time [7]. To better quantify the relationship between the number of active workers y_j and provisioned workers n, we consider the two preemption distributions in Lemma 3: a uniform distribution and a binomial distribution where each worker has an equal and independent probability of being preempted. We will make use of the fact that for both distributions, there exists a parameter $\chi > 0$ such that $\mathbb{E}\left[\frac{1}{y_j}\right] \le O\left(\frac{1}{n^\chi}\right)$ (same as the upper-bound assumption on $\mathbb{E}\left[\frac{1}{y_j}\right]$ in our error bounds from Theorem 2 and Corollary 1). The problem of minimizing the job cost is then equivalent to minimizing $\mathbb{E}\left[\sum_{j=1}^{J} y_j R(y_j)\right]$, subject to the completion time and error constraints.

Lemma 3 (Example Distributions of y_j): If the number of active workers y_j follows a uniform distribution $\mathbb{P}[y_j = k] = \frac{1}{n_j}, \forall k = 1, \cdots, n_j$, we have $\mathbb{E}\left[\frac{1}{y_j}\right] \le O\left(n_j^{-\frac{1}{2}}\right)$; if each worker is preempted with probability q each iteration, we have $\mathbb{E}\left[\frac{1}{y_j}\right] \le O\left(\frac{1}{n_j^\chi}\right)$ for some $\chi \in (0,1)$.

We use our error bound derived in Theorem 1 and the above relationship $\mathbb{E}\left[\frac{1}{y_j}\right] \le O\left(\frac{1}{n^\chi}\right)$ to find closed-form solutions for the optimal number of workers n and iterations J when $\chi \ge 1$ in Theorem 7, which holds for the fully-synchronous, N-synchronous, and N-batch-synchronous SGD algorithms. We then optimize our strategy of exponentially increasing the number of provisioned workers over iterations by using our improved error bound with a dynamic number of workers from Theorem 2.

Theorem 7 (Co-Optimizing n and J): Suppose $\mathbb{E}[y_j] \propto n$ and $\mathbb{E}\left[\frac{1}{y_j}\right] \le \frac{d}{n}$ $(d > 0)$, the probability of no active workers does not depend on n, and the runtime per iteration is deterministic. Then the completion time constraint (3) *is simply $J \le \theta\delta$ where δ is a constant, and the optimal J and n (denoted by J^* and n^*) satisfy*:

$$J^* = \min\left\{\underset{J\in\{J_1,J_2\}}{\arg\min}\ \frac{BJ(1-\beta^J)}{(1-\beta)(\epsilon - A\beta^J)}, \lfloor\theta\delta\rfloor\right\},$$

$$J_1 = \left\lfloor \tilde{J} \right\rfloor,\ J_2 = \left\lceil \tilde{J} \right\rceil, \frac{A\beta^{\tilde{J}}\left(\tilde{J}\ln\frac{1}{\beta} + 1 - \beta^{\tilde{J}}\right)}{1 + \beta^{\tilde{J}}(\tilde{J}\ln\frac{1}{\beta} - 1)} = \epsilon,$$

$$n^* = \left\lceil \frac{B(1-\beta^{\tilde{J}})}{(1-\beta)(\epsilon - A\beta^{\tilde{J}})} \right\rceil,$$

where $\beta = 1 - \alpha c\mu$, $A = \mathbb{E}[G(\mathbf{w}_1)]$, and $B = \frac{\alpha^2\ LMd}{2}$.

Cost Minimization with Negligible Stragglers. We also consider the strategy of increasing the provisioned workers per iteration as in Theorem 2 and optimize η, the rate of increasing the number of provisioned workers, to minimize the expected cost, subject to the error and completion time constraints. If we ignore straggler effects, we can define $\mathbb{E}[R(y_j)] = R,\ \forall j$. Suppose z_j denotes the number of active workers including the case $z_j = 0$, and z_j follows a binomial distribution with parameter n_j and probability q (the probability that each instance is inactive), namely, the probability that $z_j = 0$ equals $q^{n_1\eta^{j-1}}$. Given the error bound in Theorem 2, $\mathbb{E}[y_j] = n_j(1-q) = n_1(1-q)\eta^{j-1}$, and $\mathbb{E}\left[\frac{1}{y_j}\right] \le \frac{d}{n_j^\chi}$ for some $0 < \chi < 1$ according to Theorem 2, our cost minimization problem can be modified as follows.

$$\mathrm{minimize}_\eta \ \sum_{\mathrm{j=1}}^{\mathrm{J}} \mathrm{R(1-q)n_1}\eta^{\mathrm{j-1}} \tag{25}$$

$$\text{subject to}: \ \sum_{\mathrm{j=1}}^{\mathrm{J}} \mathrm{R/(1 - q^{n_1\eta^{j-1}})} \le \theta \tag{26}$$

$$A\beta^J + \frac{B\beta^{J-1}\left(1 - (\frac{1}{\beta\eta^\chi})^J\right)}{n_1^\chi\left(1 - \frac{1}{\beta\eta^\chi}\right)} \le \epsilon \tag{27}$$

$$\eta^\chi > 1/\beta, \tag{28}$$

where $\beta = 1 - \alpha c\mu$, $A = \mathbb{E}[G(\mathbf{w}_1)]$, and $B = \frac{\alpha^2\ LMd}{2}$. For any given J, both the objective function and constraints are convex functions of η (refer to the operations that preserve convexity in [30]). Therefore, we can use standard algorithms for convex optimization to solve for the optimal η.

Modified (26) **to Capture Straggling:** We can also solve for the optimal η when workers's per-iteration runtimes follow an exponential distribution ($\exp(\lambda)$), capturing some straggler effects, and each worker is preempted with probability q. We first consider the fully-synchronous SGD. We replace the constant per-iteration runtime R in (25) and (26) with $\mathbb{E}[R(y_j)] \approx \frac{1}{\lambda}\log\left(n_1(1-q)\eta^{j-1}\right)$ in the completion time constraint (3), assuming $n_1(1-q) \ge 1$ to ensure that the approximated $\mathbb{E}[R(y_j)]$ is valid. This constraint accounts for the fact that as we have more active workers in each iteration, the per-iteration runtime will likely increase because we need to wait for the slowest worker to finish. Similar to the case without stragglers, for each fixed J, our optimization problem for this case has a convex objective function and monotonic error constraint in η. For the completion time constraint with any fixed J, the left-hand side of (26) is monotonically increasing if $\sum_{j=1}^{J}\frac{1}{x}(1-q^x) + \log((1-q)x)q^x\log(q) \ge 0$, where $x := n_1\eta^{j-1}$, which essentially means that if J and (or) n_1 are (is) sufficiently large and (or) q is relatively low, the expected completion time will monotonically increase with η. For example, one can verify that if $q \le 0.8$, the above monotonicity holds for any J and x. For large q that approaches 1, the expected completion time could first decrease with η when η is small and then increase with η when η is large. Therefore, we can solve for the optimal η with any fixed J by first converting (26) and (27) to be feasible ranges of η and then solve for the η by minimizing (25) within the feasible range. Moreover, there exists a finite maximum number of iterations J for which the modified (26) is feasible. Thus, we can jointly optimize the optimal rate of increase in the number of workers, η, and J by iterating over all possible values of J.

Other SGD variants. We next consider N_j-synchronous and N_j-batch-synchronous SGD, where we wait for $N_j = \bar{n}\bar{\eta}^{j-1}$ mini-batches to be processed in each iteration j. Based on our Remark 4, the appropriate error constraint can be adapted from (27) by replacing n_1^χ and η^χ by $\bar{n}$ and $\bar{\eta}$, respectively, where $1 < \bar{\eta} \le \eta, 1 \le \bar{n} \le qn_1$, ensuring that in each iteration the number of mini-batches to wait for is no larger than the expected number of active workers. We can approximate the expected runtime per iteration by $\mathbb{E}[R(y_j)] \approx \frac{1}{\lambda}\log(\frac{(1-q)n_1\eta^{j-1}}{(1-q)n_1\eta^{j-1} - \bar{n}\bar{\eta}^{j-1}})$ for N_j-synchronous

SGD and $\mathbb{E}\left[R(y_j)\right] \approx \frac{\bar{n}\bar{\eta}^{j-1}}{\lambda(1-q)n_1\eta^{j-1}}$ for N_j-batch-synchronous SGD. One can verify that the expected cost is convex in η for N_j-synchronous and is a constant for N_j-batch-synchronous SGD. The error constraint is then no longer dependent with η, and the expected completion time is monotonically decreasing with η, meaning constraint (26) is equivalent to a lower-bound on η. Combined with the assumption that $\eta \geq \bar{\eta} > 1$, the optimal η can be found by minimizing the convex objective function subject to a lower-bound of η, for each J. Similar to the fully-synchronous case, for both N_j-synchronous and N_j-batch-synchronous SGD, we can jointly optimize the optimal rate of increase in the number of workers, η, and J by iterating over all possible values of J.

VI. Experimental Validation

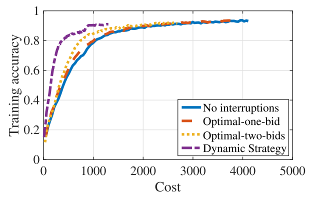
We evaluate our bidding strategies from Section IV and our strategy to optimize the number of provisioned workers from Section V on the CIFAR-10 image classification benchmark dataset in three sets of experiments. We test the ResNet-50 [40] model on our local cluster with multiple NVIDIA TitanX GPUs, a small Convolutional Neural Network (CNN) [41] with two convolutional layers and three fully connected layers on Amazon EC2's c5.xlarge spot instances, and a larger CNN featuring two convolutional layers with more channels and two fully connected layers on Google Cloud Platform's Preemptible n1-standard-2 VM Instances [18]. We use synthetic pricing data and historical price traces of AWS Spot instances to test our bidding strategies in the first and second sets of experiments, while experiments on the GCP Preemptible instances are subject to the prevailing prices and preemptions imposed by the GCP platform while the experiments ran. The distributed SGD algorithms for all experiments are implemented based on Ray [42] and Tensorflow [43].

Choosing the Experiment Parameters. By default, we run $J = 5000$ iterations for our ResNet-50 experiments and $J = 10000$ iterations for the experiments on both the Spot instances and GCP Preemptible instances. We set the deadline (θ) to be twice the estimated runtime of using 8 workers to process J iterations without interruptions and the step size α to be 0.1 by default. We estimate that $Q(\epsilon) \in [\frac{1}{n}, \frac{1}{n_1}]$ for our choices of ϵ and J ($\epsilon = 0.98$ for ResNet-50 and $\epsilon = 0.65$ for the small CNN), demonstrating the robustness of our optimized strategies to mis-estimations. To estimate the probability distribution of the spot prices, we first consider two synthetic spot price distributions for the ResNet-50 experiments: a uniform distribution in the range $[0.2, 1]$ and a Gaussian distribution with mean and variance equal to 0.6 and 0.175; we draw the spot price when each iteration starts and re-draw it every 4 seconds after the job is interrupted. We download the historical price traces of c5.xlarge spot instances using Amazon EC2's DescribeSpotPriceHistory API for the small CNN experiments, demonstrating that our bidding strategy is robust to non-i.i.d spot prices.

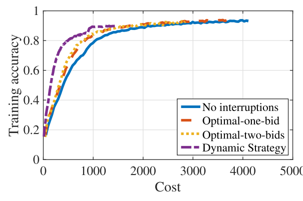
A. Advantage of Our Bidding Strategies

We evaluate the bidding strategies with both the optimal single bid price for all workers (**Optimal-one-bid**) and the optimal bid prices for two groups of workers derived in Theorem 4 (**Optimal-two-bids**) against an aggressive **No-interruptions** strategy that chooses a bid price larger than the maximum spot price. To further minimize the expected total cost while guaranteeing a low training/test error, we propose a **Dynamic strategy**, which updates the optimal two bid prices when increasing the total number of workers. More specifically, we initially launch four workers ($n_1 = 2$, $n = 4$) and apply our optimal two bid prices. After completing 4000 iterations, we add four more workers ($n_1 = 4, n = 8$) and re-compute the optimal bids by subtracting the consumed time from the original deadline θ and taking J to be the number of remaining iterations. This dynamic strategy can be regarded as a coarse grained version of our strategy of exponentially increasing the number of provisioned workers each iteration (cf. Theorem 2 and Section V) – it adds extra workers once instead of persistently adding workers in each iteration.

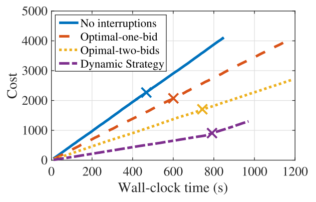
Figures 4 and 5 compare the performance of our strategies on synthetic and real spot prices, respectively. Figures 4a and 4b show that *our dynamic strategy leads to a lower cost and the no interruptions benchmark to a higher cost for any given accuracy*, compared to the optimal-one-bid and optimal-two-bids strategies. In Figures 4c and 4d, we indicate the cumulative cost as we run the jobs. The markers indicate the costs where we achieve 90% training accuracy; while the no interruptions benchmark achieves this accuracy



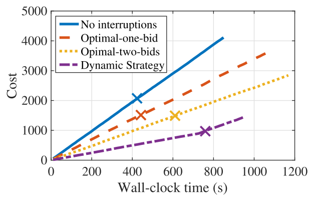
(a) Accuracy-vs-cost, uniform spot price distribution



(b) Accuracy-vs-cost, Gaussian spot price distribution

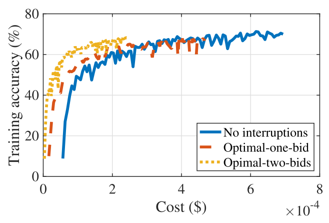


(c) Cost-vs-time, uniform spot price distribution

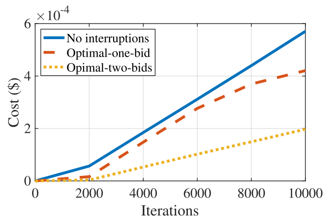


(d) Cost-vs-time, Gaussian spot price distribution

Fig. 4. The dynamic strategy (a,b) achieves the highest training accuracy under any given cost using the ResNet-50 model for CIFAR-10 classification, under synthetic spot prices. The markers on the curves in (a,b) show the cost when achieving a 90% training accuracy; at which point No-interruptions, Optimal-one-bid, and Optimal-two-bids respectively increase the cost by 134%, 82%, 46% under the uniform distribution, and 103%, 101%, 43% under the Gaussian distribution relative to the dynamic strategy.

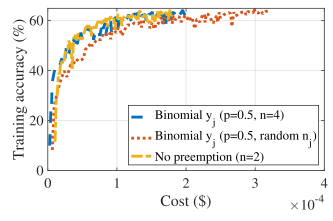


(a) Accuracy-vs-cost

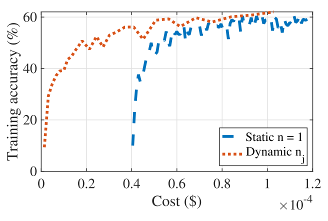


(b) Cost-vs-iterations

Fig. 5. Under historical price traces of the c5x.large spot instances in the region of us-west-2a (Oregon) and using a small CNN for CIFAR-10 classification, Optimal-one-bid and Optimal-two-bids can reduce the cost by 26.27% and 65.46% respectively compared with No-interruptions (Figure 5a) while achieving 96.78% and 96.46% of the training accuracy that No-interruptions achieves (Figure 5b).

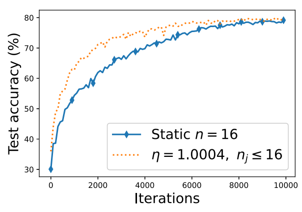


(a) Accuracy-vs-cost varying preemption probability and n

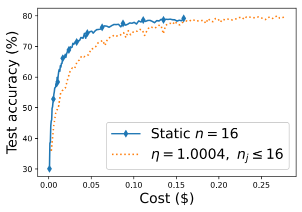


(b) Accuracy-vs-cost: static-vs-dynamic strategies

Fig. 6. Using n estimated based on Theorem 7 achieves higher accuracy per dollar than randomly setting n (Figure 6a). Compared with using 1 worker for $J = 10000$ iterations, dynamically setting $n_j = 1.0004^{j-1}$ and the number of iterations according to Theorem 2 with $\chi = 1$ achieves higher accuracy per dollar on EC2 spot instances.



(a) Accuracy-vs-iterations



(b) Accuracy-vs-cost

Fig. 7. Compared with using 16 workers, dynamically setting $n_j = \min\{1.0004^{j-1}, 16\}$ achieves the same accuracy after 10000 iterations (Figure 7a), but can reduce the cost by at least 46% (Figure 7b).

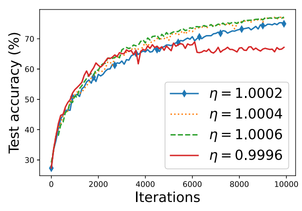
much faster, it *costs nearly three times as much as our dynamic strategy and twice as much as our optimal-two-bids strategy*. Figures 5a and 5b show that our optimal-one-bid and optimal-two-bids strategies can significantly reduce the cost under the real spot prices while achieving almost the same training accuracy as the no interruptions benchmark.

B. Advantage of Our Choices of the Number of Workers

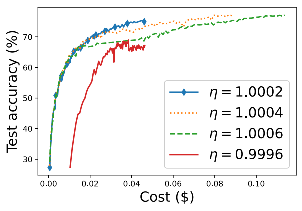
To verify our results in Section V, we conduct experiments on both AWS Spot instances and GCP Preemptible instances.

1) AWS experiments: We first simulate **No preemption** by running 2 workers for 10000 iterations without preemption and observe that the final accuracy can approach 63%. We then suppose instances are preempted with probability $p = 0.5$ and provision $n = 4$ workers, using the fact that the optimal n for each fixed J is proportional to $1/(1-p)$ and aiming to achieve the same accuracy 65%. Co-optimizing n and J (Theorem 7) may yield further cost improvements. Figure 6a shows that *using our estimated n achieves a better accuracy per dollar than randomly choosing n.* Note that to simulate various preemption probability distributions used in Figure 6a, we manually set the random preemption events in our code and use on-demand prices as our bids to simulate a lack of control over bidding as considered in Section V. We further show in Figure 6b that *our strategy* **Dynamic $\mathbf{n_j}$**, *which exponentially increases n_j by a fixed rate* 1.0004 *and runs for a much smaller number of iterations set according to Theorem 2, achieves a better accuracy per dollar, compared with using 1 worker for $J = 10000$ iterations (***Static n=1***).*

2) GCP experiments: Following Figure 6b, we compare our dynamic strategy with the rate $\eta = 1.0004$ and a maximum number of workers $n_j \leq 16$ with the strategy of always using 16 workers (**Static n=16**). Using 16 workers for the entire training leads to a faster accuracy convergence than gradually increasing the workers to 16 as shown in Figure 7a. However, *gradually increasing the number of workers leads to a higher accuracy per dollar*, shown in Figure 7a, demonstrating the advantage of our dynamic strategy in cost reduction subject to the accuracy constraint. To further evaluate our strategy of using dynamic workers with a fixed rate η, we compare different choices of η in Figure 8 where the choices for $\eta > 1$ and $\eta < 1$ represent increasing and decreasing the number of the workers over iteration, respectively. Figures 8a and 8b show that *increasing the number of workers can achieve both a steady accuracy increase and a smaller accuracy per dollar, compared to decreasing the number of workers over iterations*, verifying our motivation observed from Theorem 1 that increasing the number of workers over iterations achieves a better error bound as proven in Theorem 2.



(a) Accuracy-vs-iterations



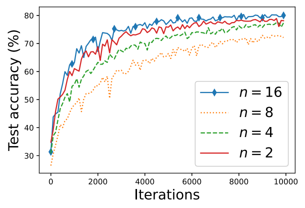
(b) Accuracy-vs-cost

Fig. 8. Using a ratio $\eta > 1$ to set the number of workers according to $n_j = \min\{\eta^{j-1}, 16\}$ for our dynamic strategy achieves a higher accuracy (Figure 8a) and higher accuracy-per-dollar (Figure 8b) than using $\eta = 0.0096$, which decreases the number of workers.

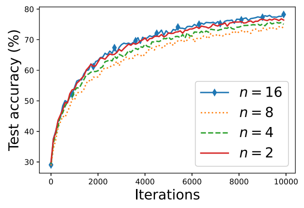
C. Improving the Step Size Based on Our Error Bounds

While picking the right step size α is crucial to the training success of ML jobs [23], [24], in practice the step size is estimated based on experimental experience with few studies showing how the number of workers affects the best choice of step size. From our Theorem 1, we observe that when the number of iterations is sufficiently large, the error bound is dominated by the term $\frac{1}{2}\alpha^2 LM \sum_{j=1}^{J}(1-\alpha c\mu)^{J-j}\mathbb{E}\left[\frac{1}{y_j}\right]$ and converges to $\frac{\alpha LM}{2}\mathbb{E}\left[\frac{1}{y_j}\right]$ when $J \to +\infty$, i.e., approximately $\frac{\alpha LM}{2n}$ when preemption rarely happens. This observation motivates us to use a larger α when n is larger, which decreases the error bound faster for early iterations (where $(1-\alpha c\mu)^J$ dominates (10)) while a larger n can offset the increased asymptotic error due to using a larger α. In the following experiments, we evaluate this insight by choosing $\alpha = 0.03\, n$ in Figure 9a. The results verify our theoretical insight that *proportionally increasing the step size with the number of workers n accelerates the accuracy convergence compared to using a step size independent of n, while achieving a slightly better asymptotic error.* In contrast, in Figure 9b, all experiments use the same step size 0.1 and show a similar convergence rate throughout the training. Figure 9a indicates that in practice we could use increase the learning rate on the same order of the increase of the number of workers to achieve our target accuracy in a shorter training time, which is consistent with a recent strategy of linearly scaling the step size when the mini-batch size is increased [44].

We further verify our insights gained from Theorem 2 for our strategy of increasing the number of workers with a fixed rate η over iterations. Theoretically, for static provisioned workers, an $O(\frac{1}{j})$ diminishing step size can improve the asymptotic error to be zero, in contrast to the error converging to a positive constant as achieved with a static step size (see our Theorem 1). However, no existing study shows the

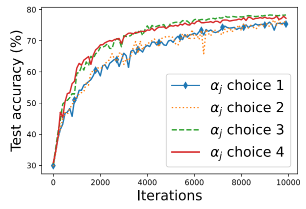


(a) Linearly scaled step sizes

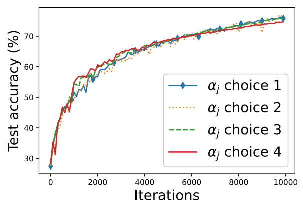


(b) Identical step size

Fig. 9. Linearly scaling the step size with the number of provisioned workers ($\alpha = 0.03 \times n$ in Figure 9a) improves the accuracy convergence rate compared to using the same step size when varying the number of provisioned workers ($\alpha = 0.1$ in Figure 9b).



(a) Static number of provisioned workers



(b) Dynamic number of provisioned workers

Fig. 10. Using a diminishing step size (α_j choice 3 or 4) improves the accuracy when using a static $n = 4$ provisioned workers (Figure 10a), but does not significantly affect the dynamic strategy which increases the number of workers with a constant rate ($\eta = 1.0002$ in Figure 10b).

best choice of step size for dynamic provisioned workers. Our Theorem 2 indicates that the standard $O(\frac{1}{j})$ diminishing step size does not further improve the exponential error decay rate achieved by using a static step size when the number of workers geometrically increases over the iterations. To verify our hypothesis, we first construct four choices of step size: 1) $\alpha = 0.1$; 2) $\alpha_j = \min\left(0.03 \times 1.2^{\lfloor \frac{j}{1000} \rfloor}, 0.18\right)$; 3) $\alpha_j = 0.03 + \frac{0.15}{\lfloor \frac{j}{1000} \rfloor + 1}$; and 4) $\alpha_j = \frac{0.18}{\lfloor \frac{j}{1000} \rfloor + 1}$. Figure 10a shows that *when the number of workers is static, the diminishing step sizes (Choices 3 and 4) achieve a faster convergence than both a static step size (Choice 1) and an increasing step size (Choice 2)*. In contrast, Figure 10b shows that *the accuracy convergence is nearly independent of our step size choice when the number of workers geometrically increases*. This result verifies our Theorem 2 and Remark 5, which indicate that any sequence of step sizes that is upper-bounded by $\frac{1 - \frac{1}{\eta \chi}}{c\mu}$ can achieve exponential decay convergence when using geometrically increased numbers of workers.

VII. Discussion and Conclusion

In this work, we consider the use of volatile workers that run distributed SGD algorithms to train machine learning models. We first focus on Amazon EC2 spot instances, which allow users to reduce job cost at the expense of a longer training time to achieve the same model accuracy. Spot instances allow users to choose how much they are willing to pay for computing resources, thus allowing them to control the trade off between a higher cost and a longer completion time or higher training error. We quantify these trade-offs and derive new bounds on the training error when using time-varying numbers of workers. We finally use these results to derive optimized bidding strategies for users on spot instances and propose practical strategies for scenarios when users cannot control preemption of their instances by submitting bids. We validate these strategies by comparing them to heuristics when training neural network models on the CIFAR-10 image dataset.

Our proposed strategies are an initial step towards a more comprehensive set of methods that allow distributed ML algorithms to exploit the benefits of volatile instances. As a simple extension, one might adapt the bids over time as we obtain better estimates of the iteration running time. Our bidding strategies might also be generalized to allow different bids for each worker. Even more generally, one can envision dividing a resource budget across workers, with the budget controlling each worker's availability. This budget might be a monetary budget when workers are run on cloud instances, but if the workers are instead run on mobile devices, it might instead represent a power budget that controls how often these devices can afford to process data. We will also investigate adjusting the step size (learning rate) based on the worker volatility, as our theoretical results and experiments for our dynamic strategy imply that the optimal way to set the step size in improving the accuracy can depend on our strategy to adjust the number of workers.

Acknowledgment

The authors would like to thank Fangjing Wu and Wanquan Wu for their assistance with the experimental setup.

References

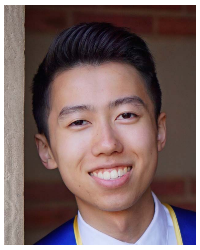
[1] X. Zhang, J. Wang, G. Joshi, and C. Joe-Wong, "Machine learning on volatile instances," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020.
[2] *Machine Learning on Volatile Instances: Convergence, Runtime, and Cost Tradeoffs*. [Online]. Available: http://andrew.cmu.edu/user/cjoewong/Spot_ML.pdf
[3] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, Sep. 1951.
[4] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*, 2010.
[5] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Int. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 1, Dec. 2012, pp. 1223–1231.
[6] Amazon EC2. (2019). *Amazon EC2 Spot Instances*. [Online]. Available: https://aws.amazon.com/ec2/spot/
[7] Google Cloud Platform. (2019). *Preemptible Virtual Machines*. [Online]. Available: https://cloud.google.com/preemptible-vms/
[8] Microsoft Azure. (2018). *Announcing Low-Priority VMS on Scale Sets Now in Public Preview*. [Online]. Available: https://azure.microsoft.com/en-us/blog/low-priority-scale-sets/
[9] Amazon EC2. (2019). *Spot Price Overrides*. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html#spot-price-overrides
[10] F. Yang and A. A. Chien, "ZCCloud: Exploring wasted green power for high-performance computing," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2016, pp. 1051–1060.
[11] A. A. Chien, F. Yang, and C. Zhang, "Characterizing curtailed and uneconomic renewable power in the mid-continent independent system operator," 2016, *arXiv:1702.05403*. [Online]. Available: http://arxiv.org/abs/1702.05403
[12] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*. [Online]. Available: http://arxiv.org/abs/1610.05492
[13] Z. Tao and Q. Li, "eSGD: Communication efficient distributed deep learning on the edge," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2018.
[14] Y. Tu, Y. Ruan, S. Wagle, C. G. Brinton, and C. Joe-Wong, "Network-aware optimization of distributed learning for fog computing," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 71–84.
[15] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015.
[16] A. Krizhevsky, V. Nair, and G. Hinton. *The CIFAR-10 Dataset*. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[17] P. Sharma, D. Irwin, and P. Shenoy, "How not to bid the cloud," in *Proc. USENIX Conf. Hot Topics Cloud Comput. (HotCloud)*, 2016.
[18] Google Cloud Platform. (2019). *Preemptible VM Instances*. [Online]. Available: https://cloud.google.com/compute/docs/instances/preemptible
[19] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *J. Mach. Learn. Res.*, vol. 13, pp. 165–202, Jan. 2012.
[20] A. Ghosh, R. K. Maity, A. Mazumdar, and K. Ramchandran, "Communication efficient distributed approximate Newton method," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2020, pp. 1000–1008.
[21] M. Kamp *et al.*, "Efficient decentralized deep learning by dynamic model averaging," 2018, *arXiv:1807.03210*. [Online]. Available: http://arxiv.org/abs/1807.03210
[22] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016, *arXiv:1602.05629*. [Online]. Available: http://arxiv.org/abs/1602.05629
[23] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," in *Proc. AISTATS*, 2018, pp. 75–82.
[24] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, 2018.
[25] D. Wang, G. Joshi, and G. W. Wornell, "Efficient straggler replication in large-scale parallel computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 2, pp. 1–23, Jun. 2019, doi: 10.1145/3310336.
[26] M. Zafer, Y. Song, and K.-W. Lee, "Optimal bids for spot VMs in a cloud for deadline constrained jobs," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012.
[27] 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. 12th Eur. Conf. Comput. Syst.*, Apr. 2017.
[28] K. Lee and M. Son, "DeepSpotCloud: Leveraging cross-region GPU spot instances for deep learning," in *Proc. IEEE 10th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2017, pp. 98–105.
[29] Y. Yan, Y. Gao, Y. Chen, Z. Guo, B. Chen, and T. Moscibroda, "TR-Spark: Transient computing for big data analytics," in *Proc. 7th ACM Symp. Cloud Comput.*, Oct. 2016, pp. 484–496.
[30] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2014.
[31] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD," in *Proc. SysML Conf.*, 2018. [Online]. Available: http://arxiv.org/abs/1810.08313
[32] H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon, "Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion," in *Proc. VLDB Endowment*, 2014, pp. 975–986.
[33] J. Chen, X. Pan, R. Monga, and S. Bengio, "Revisiting distributed synchronous SGD," in *Proc. ICLR Workshop Track*, 2016.
[34] H. Yu and R. Jin, "On the computation and communication complexity of parallel SGD with dynamic batch sizes for stochastic non-convex optimization," in *Proc. ICML*, vol. 97, PMLR, 2019, pp. 7174–7183.
[35] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.
[36] *Probability Density Function*. [Online]. Available: https://en.wikipedia.org/wiki/Probability_density_function
[37] *Cumulative Distribution Function*. [Online]. Available: https://en.wikipedia.org/wiki/Cumulative_distribution_function
[38] *How Spot Instances Work*. [Online]. Available: https://docs.aws.amazon.com/aws-technical-content/latest/cost-optimization-leveraging-ec2-spot-instances/how-spot-instances-work.html
[39] S. Dutta, V. Cadambe, and P. Grover, "Short-Dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 2100–2108.
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*. [Online]. Available: http://arxiv.org/abs/1512.03385
[41] G. E. H. A. Krizhevsky and I. Sutskever, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1097–1105.
[42] P. Moritz *et al.*, "Ray: A distributed framework for emerging ai applications," in *Proc. USENIX OSDI*, 2018, pp. 561–577.
[43] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX OSDI*, 2016, pp. 265–283.
[44] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2018, *arXiv:1706.02677*. [Online]. Available: https://arxiv.org/abs/1706.02677
[45] H. Karimi, J. Nutini, and M. Schmidt, "Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition," in *Proc. ECML PKDD*, 2016, pp. 795–811.
[46] M. T. Chao and W. E. Strawderman, "Negative moments of positive random variables," *J. Amer. Stat. Assoc.*, vol. 67, no. 338, pp. 429–431, Jun. 1972.

Xiaoxi Zhang (Member, IEEE) received the B.E. degree in electronics and information engineering from the Huazhong University of Science and Technology in 2013 and the Ph.D. degree in computer science from The University of Hong Kong in 2017. She is currently an Associate Professor with the School of Computer Science and Engineering, Sun Yat-sen University. Before joining SYSU, she was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, Carnegie Mellon University. She is broadly interested in optimization and algorithm design for networked systems, including cloud and edge computing networks, NFV systems, and distributed machine learning systems.

Jianyu Wang (Student Member, IEEE) received the B.E. degree in electronic engineering from Tsinghua University in 2017. He is currently pursuing the Ph.D. degree with Carnegie Mellon University, advised by Prof. Gauri Joshi and affiliated with the Parallel Data Laboratory. He worked as a Research Intern at Google Research and Facebook AI Research in 2019 and 2020, respectively. His research interests include federated learning, distributed optimization, and systems for large-scale machine learning. His research has been supported by Qualcomm Ph.D. Fellowship.

Li-Feng Lee received the B.S. degree in computer science from Tsinghua University in 2018 and the M.S. degree in electrical and computer engineering from Carnegie Mellon University in 2020. His research interests include cloud computing, and all things big data.



Tom Yang received the B.S. degree in electrical and computer engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 2019, and the M.S. degree in electrical and computer engineering from Carnegie Mellon University in 2020.

Akansha Kalra received the B.E. degree (Hons.) in electronics and communication from Panjab University, Chandigarh, and the M.S. degree from Carnegie Mellon University (CMU) in 2020. She is also a Research Intern with the Machine Learning Department, CMU. Her research interests include creating intelligent embodied agents using reinforcement learning and natural language.

Gauri Joshi (Member, IEEE) received the B.Tech. and M.Tech. degrees in electrical engineering from the Indian Institute of Technology (IIT) Bombay in 2010, and the Ph.D. degree from MIT EECS in June 2016. After her Ph.D., she worked as a Research Staff Member at IBM T. J. Watson Research Center. Since fall 2017, she has been an Assistant Professor with the ECE Department, Carnegie Mellon University. Her research spans distributed machine learning, large-scale parallel computing, and information theory. Her awards and honors include the NSF CAREER Award (2021), ACM Sigmetrics Best Paper Award (2020), NSF CRII Award (2018), Best Thesis Prize in Computer science at MIT (2012), and the Institute Gold Medal of IIT Bombay (2010).

Carlee Joe-Wong (Member, IEEE) received the A.B. degree *(magna cum laude)* in mathematics, and the M.A. and Ph.D. degrees in applied and computational mathematics from Princeton University in 2011, 2013, and 2016, respectively. From 2013 to 2014, she was the Director of Advanced Research at DataMi, a startup she co-founded from her research on mobile data pricing. She is currently the Robert E. Doherty Assistant Professor of electrical and computer engineering with Carnegie Mellon University. Her research interests lie in optimizing various types of networked systems, including applications of machine learning and pricing to cloud computing, mobile/wireless networks, and ridesharing networks. She received the NSF CAREER Award in 2018 and the ARO Young Investigator Award in 2019.