Machine Learning on Volatile Instances

Xiaoxi Zhang, Jianyu Wang, Gauri Joshi, Carlee Joe-Wong
Department of Electrical and Computer Engineering, Carnegie Mellon University, USA
Email:{xiaoxiz2, jianyuw1, gaurij, cjoewong}@andrew.cmu.edu

***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 that 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) affects SGD convergence and the time to train the model. By understanding these tradeoffs 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 [1]–[3]. 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 and a reasonable amount of training 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 [4]–[6]. Examples of such instances include Google Cloud Platform's preemptible instances [5] and Azure's low-priority virtual machines [6]; 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 [7]. 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 [8], [9], leading to significant temporal volatility in resource availability. SGD variants are also commonly used to train machine learning models in edge computing contexts, where resource volatility is a significant practical challenge [10], [11].

SGD algorithms can be run on volatile instances by deploying each worker on a single instance, and deploying the parameter server on an on-demand or reserved instance that is never interrupted [12]. 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 tradeoffs between error, cost, and training time for volatile instances*. We then use our analysis to propose *practical strategies for optimizing these tradeoffs* in realistic preemption environments. In particular, 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' probability of being preempted, as in GCP's preemptible instances and 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).* Bidding strategies for distributed machine learning jobs considering error convergence and the random runtime affected

by the bids have not been explored yet, to the best of our knowledge. 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 the case where each worker submits one of two distinct bids.

3) *Deriving the optimal number of workers (Section V).* For scenarios where users cannot control the preemption probability, we propose a general relationship to connect the number of provisioned workers to the expected reciprocal of the number of active workers which can capture practical preemption distributions. Using this, 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 [13] 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 [14]. Moreover, we implement two simple but effective dynamic strategies that reduce the cost and yield a better cost/completion time/error trade-off: (i) adding workers later in the job and re-optimizing the bids according to the realized error and training time so far, (ii) exponentially increasing the number of provisioned workers and running for a logarithmic number of iterations.

II. Related Work

Our work is broadly related to prior works on algorithm 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 [1], 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 [15]) 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 [10], [16]–[18], while others analyze the effect of the mini-batch size [15] or learning rate [19], [20] on SGD algorithms' training accuracy and convergence. Bottou et al. [20] analyze the convergence of training error in SGD algorithms but do not consider the running time of each iteration. Dutta et al. [19] analyze the trade-off between the training error and the (wall-clock) training time of distributed SGD, accounting for stochastic runtimes for the gradient computation at different workers. Our work is similar in spirit but focuses on spot instances, which introduces cost as another performance metric. We also go beyond [19], [20] 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. [12] 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 [21] or jointly optimize the use of spot and on-demand instances [22]. 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. [14] 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 [23]. Some software frameworks have been designed for running big data analytics on transient instances [24], 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 training error versus the number of SGD iterations as well as the runtime spent per iteration. Unlike most previous works in optimization theory literature that 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.

In Section III-B below, we quantify how the preemption probability 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 where we have 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 probability of preemption and the number of active worker nodes used for SGD training.

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 problem:

$$\text{minimize}: \quad \text{Expected total cost } \mathbb{E}[C] \tag{1}$$
$$\text{st.}: \quad \text{Expected training error } \mathbb{E}[\phi] \leq \epsilon, \tag{2}$$
$$\text{Expected completion time } \mathbb{E}[\tau] \leq \theta, \tag{3}$$

where ϵ and θ denote the maximum allowed error and the 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 stochastic gradient descent (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 $\mathbf{w}_{j+1} = \mathbf{w}_j - \alpha_j g(\mathbf{w}_j)$. where α_j is the 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. 1. 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. 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 [19]. While we focus on synchronous SGD in this paper, the insights could be extended to other distributed SGD variants.

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. 1. Let y_j denote the number of active (i.e., not preempted) workers in iteration j, such that $0 < y_j \leq n$ 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. When the number of active workers is 0 we do not consider it as an 'iteration' of SGD. However, having zero workers will

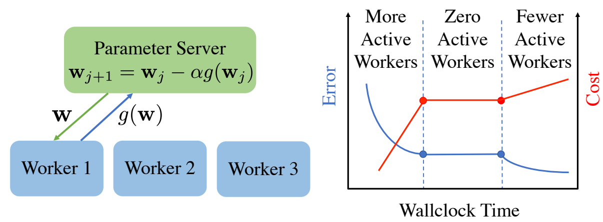


Fig. 1: Parameter Server Model and an illustration of how error and cost vary versus training time when the number of workers varies with time. Having more active workers results in a faster decrease in error, but a faster increase in cost.

increase the total training runtime, which we will account for in the runtime analysis in Section III-C.

B. SGD Error Convergence with Variable Number of Workers

Next we give an upper-bound on the expected training error in terms of y_j for $j = 1, \ldots J$. 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 [19], [20].

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

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

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

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

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

$$\mathbb{E}_{\mathcal{S}_j}\left[\| \nabla G(\mathbf{w}_j, \mathcal{S}_j) \|_2^2\right] \leq M + M_G \| \nabla G(\mathbf{w}) \|_2^2 . \tag{8}$$

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

Theorem 1 (SGD Error Bound)**.** *Suppose the objective function $G(\cdot)$ satisfies Assumptions 1– 2 and is c-strongly convex [25] with parameter $c \leq L$. For a fixed step size $0 < \alpha < \frac{\mu}{LM_G}$, the expected training error after J iterations is:*

$$\mathbb{E}[G(\mathbf{w}_J) - G^*] \leq (1 - \alpha c\mu)^J \mathbb{E}[G(\mathbf{w}_0)] + \frac{1}{2}\alpha^2 LM \sum_{j=1}^{J} (1 - \alpha c\mu)^{J-j} \mathbb{E}\left[\frac{1}{y_j}\right] \tag{9}$$

The proof is given in the Appendix. The above convergence bound can be extended to handle non-convex objective function $G(\cdot)$ and a diminishing step size, and we analyze the convergence speed to a stationary point. The extension is omitted for brevity purposes.

Remark 1 (Penalty for Using Volatile Instances)**.** The error bound in Theorem 1 is the smallest when the number of active workers is not a random variable, i.e., SGD is run on on-demand instead of volatile instances. This is because since y_j^{-1} is a convex function, 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 an 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 reduces the effective number of active workers and yields worse error convergence.

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 in 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. Fluctuations in computation time are common especially in cloud infrastructure due to background processes, node outages, network delays etc. [26]. Since the parameter server has to wait for all y_j workers to finish their gradient computations, the runtime per iteration is,

$$R(y_j) = \max_{k \in \mathcal{Y}_j} r_k + \Delta, \tag{10}$$

where Δ is the time taken by the parameter server to update $\mathbf{w}$ and push it to the y_j workers. The $\mathbb{E}[R(y_j)]$ 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 $\mathbb{E}[R(y_j)] \approx (\log y_j)/\mu + \Delta$. Adding this per-iteration runtime to the idle time when no workers are active, we can show that the expected time required to complete J SGD iterations is

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

For the case where each worker is preempted uniformly at random with probability q in each iteration (as described in Remark 2), then the expected completion time becomes $\mathbb{E}[\tau] = \sum_{j=1}^{J} \mathbb{E}\left[R(y_j)\right]/(1-q^n)$.

IV. Optimizing Spot Instance Bids

In this section, we use the results of Section III to derive the bid prices that minimize the cost of running distributed SGD with workers placed on spot instances in markets. 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 [12]. Let $f(\cdot)$ and $F(\cdot)$ denote the probability density function (PDF) [27] and the cumulative density function (CDF) [28] of the random variable 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 $\underline{p} \le b \le \bar{p}$. Whenever a worker is active ($b \ge p_t$), the per-time cost incurred for running it is equal to the prevailing spot price p_t (not the bid price).

A. Identical Worker Bids

Suppose we choose bid price b for each of the n workers. We first simplify the error and runtime in Section III for this case, and then solve the cost minimization problem (1)-(3).

Observe that the n workers 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 within an iteration. We can thus rewrite the error bound as a functions of J, the number of iterations required to reach error ϵ. Formally, we set $\hat{\phi}$ to be the right-hand side of (9) and $J \ge \hat{\phi}^{-1}(\epsilon)$, where $\hat{\phi}^{-1}(\epsilon)$ is the number of iterations required to ensure that 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 $\mathbb{E}\left[R(y_j)\right] = \mathbb{E}[R(n)]$. 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*

$$\mathbb{E}[\tau] = J\mathbb{E}[R(n)]/F(b), \tag{11}$$

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*

$$\mathbb{E}[C] = Jn\mathbb{E}[R(n)]\left(\underline{p} + \int_{\underline{p}}^{b}\left(1 - \frac{F(p)}{F(b)}\right)dp\right), \tag{12}$$

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 minimum time and cost of the volatile workers.

Optimizing the bid price. Having shown that $J = \hat{\phi}^{-1}(\epsilon)$, we now find the optimal bid b that minimizes the expected cost (see (1)).

According to Amazon's policy [4], 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 2 (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 2 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.

B. Optimal Heterogeneous Bids

We next extend our results from Section IV-A to find the optimal bidding strategy with two distinct bid prices b_1 and b_2 for two groups of 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 improves the training error but will not cost much. Formally, we place bids of b_1 for workers $1, \cdots, n_1$ and b_2 $(< b_1)$ for workers $n_1+1, \cdots, n$. We define the random variable $y(\vec{b}) \in \{n_1, n\}$ as the number of active workers when the bid prices are $\vec{b} = (b_1, b_2)$. Note that the times when 0 workers are active are not considered into an SGD 'iteration'. Thus, $y(\vec{b})$ can only be either n_1 (with probability $\frac{F(b_1)-F(b_2)}{F(b_1)}$) or n (with probability $F(b_2)/F(b_1)$) in each iteration.

Optimized bids. We initially assume that n_1, 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_1 and b_2 in Theorem 3, we discuss co-optimizing n_1 and J with the bids $\vec{b}$. The expected cost minimization problem is given as follows.

$$\min_{\vec{b}} \quad J\int_{\underline{p}}^{b_1} \mathbb{E}\Big[R(\vec{b},p)\Big] y(\vec{b}) p \frac{f(p)}{F(b_1)} \mathrm{d}p \tag{13}$$

$$\text{subject to:} \quad \mathbb{E}\Big[\hat{\phi}(\vec{b})\Big] \le \epsilon \quad \text{(Error constraint)} \tag{14}$$

$$\frac{J}{F(b_1)} \int_{\underline{p}}^{b_1} \mathbb{E}\Big[R(\vec{b},p)\Big] \frac{f(p)}{F(b_1)} \mathrm{d}p \le \theta \tag{15}$$

$$\bar{p} \ge b_1 \ge b_2 \ge \underline{p}, \ \forall i \le j \tag{16}$$

To derive the cost and completion time expressions in (13) and (15), we express the expected runtime of iteration j as $\mathbb{E}\Big[R(\vec{b},p)\Big]$, a function of the bids and price; y_j depends on $\vec{b}$ and thus is re-written as $y(\vec{b})$. 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 [29], 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.

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 we can have a $y(\vec{b})$ such that:

$$\mathbb{E}\left[\frac{1}{y(\vec{b})}\right] \le \frac{2c\mu\left(\epsilon - (1-\alpha c\mu)^J \mathbb{E}[G(\mathbf{w}_0)]\right)}{\alpha LM\left(1-(\alpha c\mu)^J\right)} \triangleq Q(\epsilon) \tag{17}$$

Further, we simplify $\mathbb{E}\Big[R(\vec{b},p)\Big]$ to be a function of the number of active workers: $\mathbb{E}[R(X)]$ is the expected runtime per iteration given X workers are active. We then provide closed-form expressions for the optimal bid prices through Theorem 3.

Theorem 3 (Optimal-Two Bids with a Fixed J)**.** *Suppose the objective function $G(\cdot)$ satisfies Assumptions 1–2. Given a number of iterations (J) that can guarantee $\frac{1}{n} < Q(\epsilon) \le \frac{1}{n_1}$ ($Q(\epsilon)$ is defined as the right-hand side of* (17)*), a fixed step size α, and a feasible deadline ($\theta \ge J\mathbb{E}[R(n)]$), we have the optimal bid prices b_1^* and b_2^*:*

$$b_1^* = F^{-1}\left(\frac{J}{\theta}\left(\left(\mathbb{E}[R(n)] - \mathbb{E}[R(n_1)]\right)\frac{\frac{1}{n_1} - Q(\epsilon)}{\frac{1}{n_1} - \frac{1}{n}} + \mathbb{E}[R(n_1)]\right)\right)$$

$$b_2^* = F^{-1}\left(\frac{\frac{1}{n_1} - Q(\epsilon)}{\frac{1}{n_1} - \frac{1}{n}} \times F(b_1^*)\right), \tag{18}$$

for any i.i.d. spot price and any i.i.d. running time per mini-batch, i.e.*, $F(\cdot)$ and $\mathbb{E}[R(n)]$ (or $\mathbb{E}[R(n_1)]$) do not change during the training process.*

For brevity, we use Figure 2 to illustrate our proof of Theorem 3. The key steps are: (i) change the variables of the optimization problem (13) to be $F(b_1)$ and $\gamma = \frac{F(b_2)}{F(b_1)}$; (ii) show that the expected cost, completion time, and error are monotonic w.r.t. to $F(b_1)$ 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 $F(b_1)$ and $F(b_2)$: γ. Formally, the error bound decreases with $\mathbb{E}\Big[y(\vec{b})^{-1}\Big]$. Applying $\mathbb{E}\Big[y(\vec{b})^{-1}\Big] = \frac{1}{F(b_1)}\left(\frac{F(b_1)-F(b_2)}{n_1} + \frac{F(b_2)}{n}\right) = \frac{1}{n_1} - \frac{1}{\gamma}\left(\frac{1}{n_1} - \frac{1}{n}\right)$ to (17) gives us the the optimal γ, since the expected cost increases with $F(b_1)$ and γ, respectively. We then choose $F(b_1^*)$ to the one that yields $\mathbb{E}[\tau] = \theta$ (tight (15)). Intuitively, $F(b_1^*)$ should be high enough to guarantee that some workers are active often enough that the job completes before the deadline.

Co-optimizing n_1 and $\vec{b}$. If n_1 is not a known input but a variable to be co-optimized with $\vec{b}$, we can write n_1 and b_2^* in terms of $F(b_1^*)$ according to (18) and plug them into (13)-(16) to solve for b_1^* first, and then derive b_2^* and the optimal n_1.

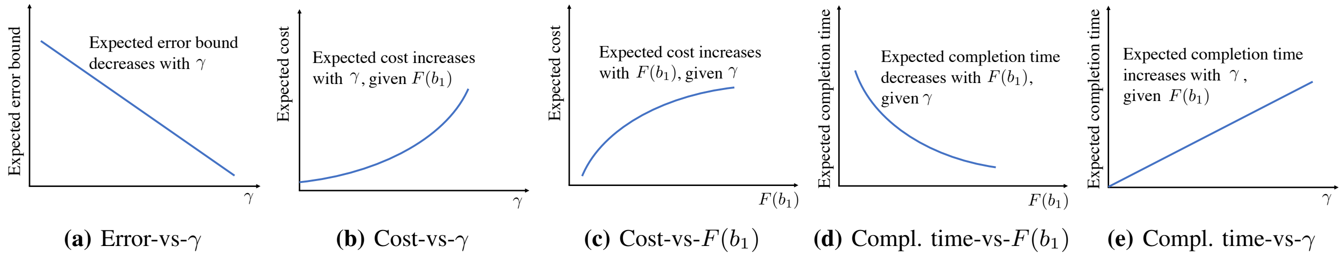


Fig. 2: Illustration of how the expected cost, completion time and error vary w.r.t. $F(b_1)$ and $\gamma = \frac{F(b_2)}{F(b_1)}$. As a larger γ leads to a smaller expected error (Fig. 2a) but a larger expected cost (Fig. 2b) and completion time (Fig. 2e), and the expected error is only controlled by γ, the optimal γ should be the smallest possible γ, *i.e.*, the one that yields error $= \epsilon$. The optimal $F(b_1)$ should be the one that yields the completion time equal to the deadline under the optimal γ (Fig. 2d).

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 (17)). We can then increase $\mathbb{E}\left[\frac{1}{y(\vec{b})}\right]$ by submitting lower bids b_2, making it less likely that workers $n_1+1,\dots,n$ will be active, while still satisfying (17). A lower b_2 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 1 (Relationship of J and $\vec{b}$)**.** *To guarantee a training error $\leq \epsilon$, the number of iterations J should be at least*

$$J = \log_{(1-\alpha c\mu)} \frac{\epsilon - \frac{\alpha LM}{2c\mu}\mathbb{E}\left[\frac{1}{y(\vec{b})}\right]}{\mathbb{E}[G(\mathbf{w}_0)] - \frac{\alpha LM}{2c\mu}\mathbb{E}\left[\frac{1}{y(\vec{b})}\right]} \tag{19}$$

For brevity, we show the idea of co-optimizing J and $\vec{b}$: We first replace J in (13) and (15) by (19). Constraint (14) is already guaranteed by (19) and can be removed. We then solve for the remaining optimization variables, the bids $\vec{b}$.

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 [6] and preemptible instances from Google Cloud Platform [5]. 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. 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 [5]. To better quantify the relationship between the number of active workers y_j and the number of provisioned workers n, we consider the two preemption distributions in Lemma 3. We will make use of the fact that for both distributions, there exists a parameter $\chi > 0$ such that $\mathbb{E}\left[\frac{1}{y_j}\right] \propto \frac{1}{n^\chi}$. 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 $\mathbb{E}\left[\frac{1}{y_j}\right] \leq O\left(n_j^{\frac{-1}{2}}\right)$; if each worker is preempted with probability q each iteration, we have $\mathbb{E}\left[\frac{1}{y_j}\right] \leq O\left(\frac{1}{n_j^\chi}\right)$, where there exists a $\chi \in (0,1)$.*

We then find closed-form solutions for the optimal number of workers n and iterations J when $\chi \geq 1$ in Theorem 4 and then provide a dynamic strategy with error analysis for any $\chi > 0$ in Theorem 5.

Theorem 4 (Co-optimizing n and J)**.** *Suppose $\mathbb{E}[y_j] \propto n$ and $\mathbb{E}\left[\frac{1}{y_j}\right] \leq \frac{d}{n}$ $(d > 0)$, 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 $J \leq \theta\delta$ 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 $\beta = 1 - \alpha c\mu$, $A = \mathbb{E}[G(\mathbf{w}_0)]$, and $B = \frac{\alpha^2 LMd}{2}$.

A strategy with dynamic numbers of workers. While Theorem 4 gives us the exact optimal expression for n when the provisioned number of workers is fixed over iterations, ML practitioners often increase the number of workers over time [30]–[32]. Intuitively, in the later stages of the model training 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, we observe in Theorem 1 that $\mathbb{E}\left[\frac{1}{y_j}\right]$'s contribution to the error bound increases exponentially with j by $\frac{1}{1-\alpha c\mu}$.

Inspired by these observations, we propose to decrease $\mathbb{E}\left[\frac{1}{y_j}\right]$ over iterations by controlling the provisioned number of workers: we dynamically set the number of workers to be

$n_j = \lceil n_0 \eta^{j-1} \rceil$ for each iteration j. One can similarly exponentially increasing the batch size of each worker while using the same number of workers over iterations [33], but doing so will exponentially increase the runtime of each iteration. We prove in Theorem 5 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 during the entire training.

Theorem 5. *Suppose that 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 having $\lceil n_0 \eta^{j-1} \rceil$ workers in iteration j and running SGD for $\lceil \log_{\eta^{\chi}} (1 + (\eta - 1)J) \rceil$ iterations achieves an error bound no larger than having a fixed set of n_0 workers running J iterations when J is sufficiently large.*

In the proof of Theorem 5, we also show that our dynamic strategy achieves an error bound that converges to 0 asymptotically with J, while when using a static number of workers the error bound in Theorem 1 converges to a positive constant.

We then optimize η to minimize the expected cost, subject to the error and completion time constraints. If we ignore straggler effects, we can define $\mathbb{E}\left[R(y_j)\right] = R,\ \forall j$. 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 $q^{n_0 \eta^j}$. Assuming $\mathbb{E}[y_j] \propto n_j = n_0 \eta^{j-1}$ and $\mathbb{E}\left[\frac{1}{y_j}\right] \leq \frac{d}{n_j^{\chi}}$, our cost minimization problem can be modified as follows.

$$\text{minimize}_{\eta} \quad (1 - \eta^J)/(1 - \eta) \tag{20}$$

$$\text{subject to}: \quad \sum_{j=1}^{J} R/(1 - q^{n_0 \eta^j}) \leq \theta \tag{21}$$

$$A\beta^J + \frac{B\beta^{J-1}\left(1 - (\frac{1}{\beta\eta^{\chi}})^J\right)}{n_0^{\chi}\left(1 - \frac{1}{\beta\eta^{\chi}}\right)} \leq \epsilon \tag{22}$$

$$\eta^{\chi} > 1/\beta, \tag{23}$$

where $\beta = 1 - \alpha c \mu$, $A = \mathbb{E}\left[G(\mathbf{w}_0)\right]$, 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 [25]). Therefore, we can use standard algorithms for convex optimization to solve for the optimal η.

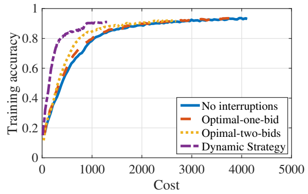
We can capture the effect of straggling workers by replacing the constant per-iteration runtime R in (3) with $\mathbb{E}\left[R(y_j)\right] = \frac{1}{\lambda}\left(\log n_0 + (j-1)\log \eta\right)$ in the completion time constraint (3). This constraint accounts for the fact that as we have more active workers in each iteration, the per-iteration runtime will likely increase because we need to wait for the slowest worker to finish. As in the case without stragglers, we then observe that our optimization problem is convex in η for each fixed J, and moreover that there exists a finite maximum number of iterations J for which (3) 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.

VI. Experimental Validation

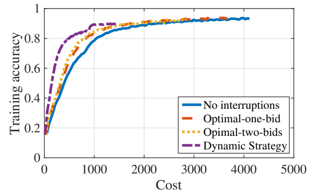
We evaluate our bidding strategy from Section IV-B on the CIFAR-10 image classification benchmark dataset, using $J = 5000$ iterations on ResNet-50 [34] and $J = 10000$ on a small Convolutional Neural Network (CNN) [35] with two convolutional layers and three fully connected layers; the distributed SGD algorithms under both datasets are implemented based on Ray [36] and Tensorflow [37]. We run the former experiments on a local cluster with GPU servers and the latter on Amazon EC2's c5.xlarge spot instances.

Choosing the experiment parameters. We set the deadline (θ) to be twice the estimated runtime of using 8 workers to process J iterations without interruptions. 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 then 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.

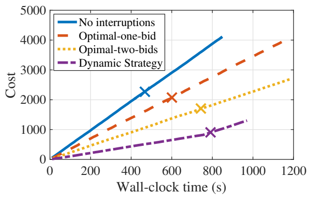
Superiority of our bidding strategies. We evaluate the bidding strategies with both the optimal single bid price for all workers (**Optimal-one-bid**) and the optimal bid prices for two groups of workers derived in Theorem 3 (**Optimal-two-bids**) against an aggressive **No-interruptions** strategy that 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 in: we subtract the consumed time from the original deadline θ and take J to be the number of remaining iterations. One could further divide the training and re-optimization into more stages. Frequent re-optimizing will likely incur significant interruption overheads, but infrequent optimization may reduce the cost with tolerable overhead. Figures 3 and 4 compare the performance of our strategies on synthetic and real spot prices, respectively. Figures 3a and 3b 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 3c and 3d, we indicate the cumulative cost as we run the jobs. The use markers to indicate the costs where we achieve 98% accuracy; while the no interruptions benchmark achieves this accuracy much faster, it *costs nearly three times as much as our dynamic strategy and twice as much as our optimal-two-bids strategy*.



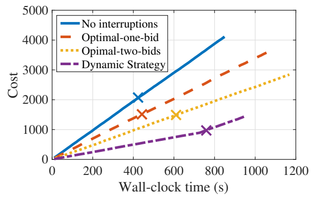
(a) Accuracy-vs-cost, uniform spot price distribution



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



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



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

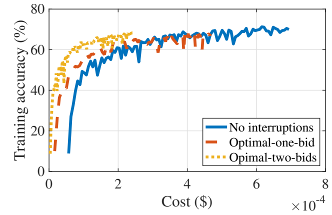
Fig. 3: The dynamic strategy (a,b) achieves the highest test accuracy under any given cost under synthetic spot prices. The markers on the curves in (c,d) show the cost when achieving a 98% test accuracy; at which point No-interruptions, Optimal-one-bid, and Optimal-two-bids respectively increase the cost by 134%, 82%, 46% under the uniform distribution, and 103%, 101%, 43% under the Gaussian distribution relative to the dynamic strategy.

Figures 4a and 4b show that our optimal-one-bid and optimal-two-bids strategies can significantly save cost under the real spot prices while achieving almost the same training accuracy as the no interruptions benchmark.

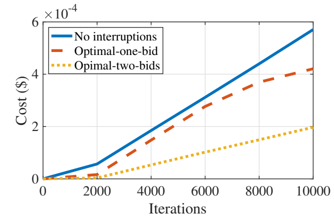
Superiority of our choices of the number of workers. To verify our results in Section V, we simulate **No preemption** by running 2 workers for 10000 iterations without preemption and observe that the final accuracy can approach 63%. We then estimate the optimal n under $J = 10000$ to be 4 when each instance is preempted with probability $p = 0.5$, aiming at the same accuracy 65% (cff. Theorem 4, the estimated optimal n is proportional to $1/(1-p)$ even if the combination of n and J is not the joint optimal solution). Figure 5a shows that using our estimated n achieves a better accuracy per dollar than randomly choosing n. We further show in Figure 5b that our strategy **Dynamic** 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 5, achieves a better accuracy per dollar, compared with using 1 worker for $J = 10000$ iterations (**Static** $n = 1$).

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

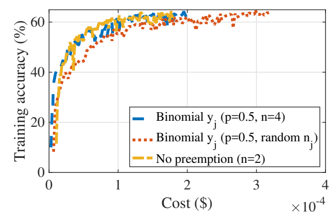


(a) Accuracy-vs-cost

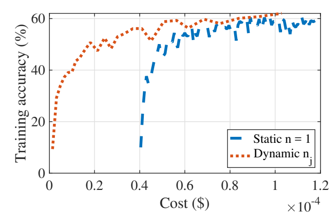


(b) Cost-vs-iterations

Fig. 4: Under historical price traces of the c5x.large spot instances in the region of us-west-2a (Oregon), Optimal-one-bid and Optimal-two-bids can save cost by 26.27% and 65.46% respectively compared with No-interruptions (Figure 4b) while achieving 96.78% and 96.46% of the training accuracy that No-interruptions achieves (Figure 4a).



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



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

Fig. 5: Using n estimated based on Theorem 4 achieves higher accuracy per dollar than randomly setting n (Figure 5a); 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 5 with $\chi = 1$ achieves higher accuracy per dollar on EC2 spot instances.

a higher cost and a longer completion time or higher training error. We quantify these tradeoffs and derive new bounds on the training error with using time-variant numbers of workers. We finally use these results to derive optimized bidding strategies for users on spot instances and propose practical strategies for scenarios without controlling the preemption of the 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.

VIII. Acknowledgments

This work was supported by NSF grants CNS-1751075, CNS-1909306, CCF-1850029, and a 2018 IBM Faculty Research Award. We also thank Fangjing Wu for her help.

Appendix

Proof of Theorem 1. $G(\mathbf{w}_{j+1})$ is at most:

$$G(\mathbf{w}_j)+\nabla G(\mathbf{w}_j)\cdot(\mathbf{w}_{j+1}-\mathbf{w}_j)+\frac{L}{2}||\mathbf{w}_{j+1}-\mathbf{w}_j||_2^2 \quad (24)$$

due to Assumption 1. Combining (5), Assumption 2, and (24),

$$\begin{aligned}&\mathbb{E}[G(\mathbf{w}_{j+1})-G(\mathbf{w}_j)]\\ &\leq -\alpha||\nabla G(\mathbf{w}_j)||_2^2\left(\mu-\frac{\alpha L M_G}{2}\right)+\mathbb{E}\left[\frac{\alpha^2 LM}{2y_j}\right] \quad (25)\\ &\leq -\frac{1}{2}\alpha\mu||\nabla G(\mathbf{w}_j)||_2^2+\mathbb{E}\left[\frac{\alpha^2 LM}{2y_j}\right], \quad (26)\end{aligned}$$

where (26) follows from our choice of $0<\alpha<\frac{\mu}{LM_G}$. If $G(\cdot)$ is c-strong convex with $c\leq L$, then it satisfies the Polyak-Lojasiewicz condition $||\nabla G(\mathbf{w}_j)||_2^2\geq 2c\left(G(\mathbf{w}_j)-G^*\right), \forall \mathbf{w}_j$ (Appendix B of [38]). Substituting this into (26) and subtracting G^* on both sides, we have:

$$\mathbb{E}[G(\mathbf{w}_{j+1})]\leq(1-\alpha c\mu)\left(G(\mathbf{w}_j)-G^*\right)+\mathbb{E}\left[\frac{\alpha^2 LM}{2y_j}\right]$$

Applying the above inequality recursively over all iterations leads to (9), and the theorem follows. □

Proof of Lemma 2. The objective function (1) takes the sum of price multipled by the runtime over all J iterations with at least one active worker. Therefore, we have $\mathbb{E}[C]=\frac{J\int_{\underline{p}}^{b} n\mathbb{E}[R(n)]pf(p)\mathrm{d}p}{F(b)}$, which equals $\frac{Jn\mathbb{E}[R(n)]}{F(b)}\int_{\underline{p}}^{b}\left((pF(p))'-F(p)\right)\mathrm{d}p$ and thus $\frac{Jn\mathbb{E}[R(n)]}{F(b)}\left(bF(b)-bF(\underline{p})-\int_{\underline{p}}^{b}F(b)\mathrm{d}p\right)$. The lemma follows as $F(\underline{p})=0$. □

Proof of Theorem 2. Note that $\mathbb{E}[C]$ is non-increasing with b, the optimal number of iterations equals $\hat{\phi}^{-1}(\epsilon)$, and the expected cost in non-decreasing with b, the optimal bid price has $\mathbb{E}[\tau]=\theta$. Setting the right-hand side of (11) to be equal to θ and taking $J=\hat{\phi}^{-1}(\epsilon)$, we can conclude that the optimal b should be equal to $F^{-1}\left(\frac{\hat{\phi}^{-1}(\epsilon)\mathbb{E}[R(n)]}{\theta}\right)$. □

Proof of Theorem 4. Given that y_j is i.i.d. across all iterations with $\mathbb{E}[y_j\mid y_j>0]\propto n$, it suffices to minimize $J\cdot n$ subject to $A\beta^J+\frac{B\left(1-\beta^J\right)}{n(1-\beta)}\leq\epsilon$. Suppose the n^* is a feasible solution that is not least integer that makes the error constraint tight, *i.e.*, satisfying $A\beta^{J^*}+\frac{B\left(1-\beta^{J^*}\right)}{(n^*-1)(1-\beta)}\leq\epsilon$, there exists a feasible solution $n'=n^*-1$ such that the objective value $J^*\cdot n'$ is strictly smaller than $J^*\cdot n^*$, a contradiction. Therefore, we can replace the objective function $J\cdot n$ by $\frac{BJ(1-\beta^J)}{(1-\beta)(\epsilon-A\beta^J)}$. Letting its derivative to be zero leads to $\frac{A\beta^{\tilde{J}}\left(\tilde{J}\ln\frac{1}{\beta}+1-\beta^{\tilde{J}}\right)}{1+\beta^{\tilde{J}}(\tilde{J}\ln\frac{1}{\beta}-1)}$ (denoted by $H(\tilde{J})$) $=\epsilon$ where $\tilde{J}$ can be fractional. One can verify that $H(\tilde{J})$ monotonically decreases with $\tilde{J}$ and the objective function is smooth. Thus, J^* should be among: the least integer no smaller than $\tilde{J}$, the largest integer no larger than $\tilde{J}$, and $\lfloor\theta\delta\rfloor$, whichever that yields the smallest objective value, the theorem follows. □

Proof of Theorem 5. Based on our Theorem 1, the error bound of using $\lceil n_0\eta^{j-1}\rceil$ workers in iteration j and running the SGD for J' iterations is at most:

$$\begin{aligned}&(1-\alpha c\mu)^{J'}\mathbb{E}[G(\mathbf{w}_0)]+B\sum_{j=1}^{J'}\frac{(1-\alpha c\mu)^{J'-j}}{(n_0\eta^{j-1})^{\chi}}\\ =&(1-\alpha c\mu)^{J'}\mathbb{E}[G(\mathbf{w}_0)]+\frac{B}{n_0^{\chi}}\cdot\sum_{j=1}^{J'}\frac{(1-\alpha c\mu)^{J'-1}}{[\eta^{\chi}(1-\alpha c\mu)]^{j-1}}\\ =&(1-\alpha c\mu)^{J'}\mathbb{E}[G(\mathbf{w}_0)]+\frac{B}{n_0^{\chi}}\cdot(1-\alpha c\mu)^{J'-1}\cdot\frac{1-x^{J'}}{1-x},\end{aligned} \quad (27)$$

where we define $x=\frac{1}{\eta^{\chi}(1-\alpha c\mu)}$ and B is a constant linear with $\frac{\alpha^2 LM}{2}$. Given our choice of $\eta^{\chi}>(1-\alpha c\mu)^{-1}$, the error bound will exponentially decrease with J'. In comparison, if using n_0 workers for J iterations, the error is at most:

$$(1-\alpha c\mu)^J\mathbb{E}[G(\mathbf{w}_0)]+\frac{B}{n_0}\cdot\frac{1-(1-\alpha c\mu)^J}{\alpha c\mu} \quad (28)$$

Based on (27), (28), and our choice of η, the error decay rate is no smaller than $(1-\alpha c\mu)$ in the dynamic strategy (bound (27)) and equals $(1-\alpha c\mu)$ in the static strategy. Moreover, when $J\to+\infty$, the error bound of the dynamic strategy approaches $\frac{B\beta^{J'-1}\left(1-(\frac{1}{\beta\eta^{\chi}})^{J'}\right)}{n_0^{\chi}\left(1-\frac{1}{\beta\eta^{\chi}}\right)}$, where $\beta:=1-\alpha c\mu$, while that of the static strategy (28) approaches $\frac{B}{(1-\beta)n_0}$. Putting $J'=\log_{\eta}(1+(\eta-1)J)$ into the former, it becomes $\frac{B[(\eta^{\chi}+1)J+1]^{\log_{\eta^{\chi}}\beta}}{n\beta(1-\frac{1}{\beta\eta^{\chi}})}$ which is smaller than $\frac{B}{(1-\beta)n_0}$ (error bound of the static strategy) when J is sufficiently large due to $\log_{\eta^{\chi}}\beta<0$, the theorem follows. □

Proof of Lemma 3. For such a uniform y_j, we have:

$$\mathbb{E}\left[\frac{1}{y_j}\right]=\sum_{k=1}^{n_j}\frac{1}{k}\cdot\frac{1}{n_j}\leq\frac{\ln n_j+1}{n_j}\leq O\left(\frac{1}{n_j^{1/2}}\right)$$

If each worker is preempted with probability q, it suffices to show that for a constant $d>0$, any $q\in[\frac{1}{2},1)$, and $\gamma\in(0,1)$, $\left|\mathbb{E}\left[\frac{1}{y_j}\right]-\mathbb{E}\left[\frac{1}{y_j+1}\right]\right|\leq dn^{-\gamma}$ is at most

$$\begin{aligned}&\leq\frac{1}{1-q^n}\left(\sum_{y=1}^{n^{\gamma}}\frac{1}{y(y+1)}\binom{n}{y}q^n+\sum_{y=n^{\gamma}+1}^{n}\frac{1}{y(y+1)}\binom{n}{y}q^n\right)\\ &\leq\frac{1}{1-q^n}\left(n^{\gamma}\left(qn^{n^{\gamma-1}}\right)^n+\frac{1}{n^{2\gamma-1}}\right)\leq\frac{d}{n^{2\gamma-1}}\end{aligned}$$

and $\mathbb{E}\left[\frac{1}{y_j+1}\right]=\frac{1-q^{n+1}}{(1+n)(1-q)}$ according to [39]. The result also holds for $q\in(0,\frac{1}{2})$ by applying the derivation on $1-q$ which is $\in[\frac{1}{2},1)$), rather than on q, the lemma follows. □

REFERENCES

[1] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[2] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT*, 2010.

[3] J. D. et al., "Large scale distributed deep networks," in *International Conference on Neural Information Processing Systems (NIPS)*, vol. 1, 2012, pp. 1223–1231.

[4] Amazon EC2, "Amazon ec2 spot instances," https://aws.amazon.com/ec2/spot/, 2019.

[5] Google Cloud Platform, "Preemptible virtual machines," https://cloud.google.com/preemptible-vms/, 2019.

[6] Microsoft Azure, "Announcing low-priority vms on scale sets now in public preview," https://azure.microsoft.com/en-us/blog/low-priority-scale-sets/, 2018.

[7] Amazon EC2, "Spot price overrides," https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html#spot-price-overrides, 2019.

[8] F. Yang and A. A. Chien, "Zccloud: Exploring wasted green power for high-performance computing," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 1051–1060.

[9] A. A. Chien, F. Yang, and C. Zhang, "Characterizing curtailed and uneconomic renewable power in the mid-continent independent system operator," *arXiv preprint arXiv:1702.05403*, 2016.

[10] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[11] Z. Tao and Q. Li, "esgd: Communication efficient distributed deep learning on the edge," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[12] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *Proc. ACM SIGCOMM*, 2015.

[13] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," https://www.cs.toronto.edu/~kriz/cifar.html.

[14] P. Sharma, D. Irwin, and P. Shenoy, "How not to bid the cloud," in *Proc. USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, 2016.

[15] O. S. Ofer Dekel, Ran Gilad-Bachrach and L. Xiao., "Optimal distributed online prediction using mini-batches." *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 165–202, 2012.

[16] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *International conference on machine learning*, 2014, pp. 1000–1008.

[17] M. Kamp, L. Adilova, J. Sicking, F. Hüger, P. Schlicht, T. Wirtz, and S. Wrobel, "Efficient decentralized deep learning by dynamic model averaging," *arXiv preprint arXiv:1807.03210*, 2018.

[18] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of AISTATS*, 2017. [Online]. Available: http://arxiv.org/abs/1602.05629

[19] 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 *Proceedings of AISTATS*, 2018.

[20] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[21] M. Zafer, Y. Song, and K.-W. Lee, "Optimal bids for spot vms in a cloud for deadline constrained jobs," in *Proc. of IEEE CLOUD*, 2012.

[22] A. Harlap, A. Tumanov, A. Chung, G. R. Ganger, and P. B. Gibbons, "Proteus: Agile ml elasticity through tiered reliability in dynamic resource markets," in *Proc. of European Conference on Computer Systems*, 2017.

[23] K. Lee and M. Son, "Deepspotcloud: leveraging cross-region gpu spot instances for deep learning," in *Proceedings of IEEE CLOUD*. IEEE, 2017, pp. 98–105.

[24] Y. Yan, Y. Gao, Y. Chen, Z. Guo, B. Chen, and T. Moscibroda, "Tr-spark: Transient computing for big data analytics," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 2016, pp. 484–496.

[25] S. Boyd and L. Vandenberghe, "Convex optimization," *Cambridge university press*, 2014.

[26] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[27] "Probability density function." [Online]. Available: https://en.wikipedia.org/wiki/Probability_density_function

[28] "Cumulative distribution function." [Online]. Available: https://en.wikipedia.org/wiki/Cumulative_distribution_function

[29] "How spot instances work," https://docs.aws.amazon.com/aws-technical-content/latest/cost-optimization-leveraging-ec2-spot-instances/how-spot-instances-work.html.

[30] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD," in *Proc. of SysML Conference*, 2019.

[31] 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. of VLDB Endowment*, 2014.

[32] J. Chen, X. Pan, R. Monga, and S. Bengio, "Revisiting distributed synchronous sgd," in *Proc. of ICLR Workshop Track*, 2016.

[33] 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. of ICML*, 2019.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[35] G. E. H. Alex Krizhevsky, IIya Sutskever, "Imagenet classification with deep convolutional neural networks," in *Proceedings of NIPS*, 2012.

[36] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging ai applications," in *In Proceedings of USENIX OSDI*, 2018.

[37] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, , and X. Zheng., "Tensorflow: A system for large-scale machine learning." in *In Proceedings of USENIX OSDI*, 2016.

[38] H. Karimi, J. Nutini, and M. Schmidt, "Linear convergence of gradient and proximal-gradient methods under the polyak-ojasiewicz condition," in *Proc. of ECML PKDD*, 2016.

[39] M. T. Chao and W. E. Strawderman, "Negative moments of positive random variables," *Journal of the American Statistical Association*, vol. 67, no. 338, pp. 429–431, 1972.