Slow and Stale Gradients Can Win the Race: Error-Runtime Trade-offs in Distributed SGD

Sanghamitra Dutta* **Gauri Joshi*** **Soumyadip Ghosh**** **Parijat Dube**** **Priya Nagpurkar****
* Carnegie Mellon University ** IBM TJ Watson Research Center

Abstract

Distributed Stochastic Gradient Descent (SGD) when run in a synchronous manner, suffers from delays in waiting for the slowest learners (stragglers). Asynchronous methods can alleviate stragglers, but cause gradient staleness that can adversely affect convergence. In this work we present the first theoretical characterization of the speed-up offered by asynchronous methods by analyzing the trade-off between the error in the trained model and the actual training runtime (wall-clock time). The novelty in our work is that our runtime analysis considers random straggler delays, which helps us design and compare distributed SGD algorithms that strike a balance between stragglers and staleness. We also present a new convergence analysis of asynchronous SGD variants without bounded or exponential delay assumptions, and a novel learning rate schedule to compensate for gradient staleness.

1 INTRODUCTION

Stochastic gradient descent (SGD) is the backbone of most state-of-the-art machine learning algorithms. Thus, improving the stability and convergence rate of SGD algorithms is critical for making machine learning algorithms fast and efficient.

Traditionally SGD is run serially at a single node. However, for massive datasets, running SGD serially at a single server can be *prohibitively* slow. A solution that has proved successful in recent years is to parallelize the training across many learners (processing units). This method was first used at a large-scale in Google's

Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84. Copyright 2018 by the author(s).

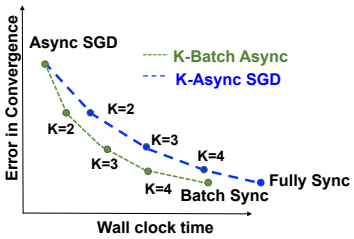

Figure 1: SGD variants span the error-runtime trade-off between fully Sync-SGD and fully Async-SGD. K is the number of learners or mini-batches the PS waits for before updating the model parameters, as we elaborate in Section 2.

DistBelief [Dean et al., 2012] which used a central parameter server (PS) to aggregate gradients computed by learner nodes. While parallelism dramatically speeds up training, distributed machine learning frameworks face several challenges such as:

Straggling Learners. In synchronous SGD, the PS waits for all learners to push gradients before it updates the model parameters. Random delays in computation (referred to as straggling) are common in today's distributed systems [Dean and Barroso, 2013]. Waiting for slow and straggling learners can diminish the speed-up offered by parallelizing the training.

Gradient Staleness. To alleviate the problem of stragglers, SGD can be run in an asynchronous manner, where the central parameters are updated without waiting for all learners. However, learners may return *stale* gradients that were evaluated at an older version of the model, and this can make the algorithm unstable.

The key contributions of this work are:

1. Most SGD algorithms optimize the trade-off between training error, and the number of iterations or epochs. However, the wallclock time per iteration is a random variable that depends on the gradient aggregation algorithm. We present the first rigorous analysis of the trade-off between error and the actual runtime (instead of iterations). This analysis is then used to compare different SGD variants such as K-sync SGD, K-async SGD and K-batch-async

SGD, as illustrated in Figure 1.

2. We present a new convergence analysis of asynchronous SGD and its variants, where we relax several commonly made assumptions such as bounded delays and gradients, exponential service times, and independence of the staleness process.

3. We propose a novel learning rate schedule to compensate for gradient staleness, and improve the stability and convergence of asynchronous SGD, while preserving its fast runtime.

1.1 RELATED WORKS

Single Node SGD: Analysis of gradient descent dates back to classical works [Boyd and Vandenberghe, 2004] in the optimization community. The problem of interest is the minimization of empirical risk of the form:

$$\min_{\mathbf{w}} \left\{ F(\mathbf{w}) \stackrel{\text{def}}{=} \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{w}, \xi_n) \right\}. \tag{1}$$

Here, ξ_n denotes the $n-$th data point and its label where $n = 1, 2, \ldots, N$, and $f(\mathbf{w}, \xi_n)$ denotes the composite loss function. Gradient descent is a way to iteratively minimize this objective function by updating the parameter $\mathbf{w}$ in the opposite direction of the gradient of $F(\mathbf{w})$ at every iteration, as given by:

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \eta \nabla F(\mathbf{w}_j) = \mathbf{w}_j - \frac{\eta}{N} \sum_{n=1}^{N} \nabla f(\mathbf{w}_j, \xi_n).$$

The computation of $\sum_{n=1}^{N} \nabla f(\mathbf{w}_j, \xi_n)$ over the entire dataset is expensive. Thus, stochastic gradient descent [Robbins and Monro, 1951] with mini-batching is generally used in practice, where the gradient is evaluated over small, randomly chosen subsets of the data. Smaller mini-batches result in higher variance of the gradients, which affects convergence and error floor [Dekel et al., 2012, Li et al., 2014, Bottou et al., 2016]. Algorithms such as AdaGrad [Duchi et al., 2011] and Adam [Kingma and Ba, 2015] gradually reduce learning rate to achieve a lower error floor. Another class of algorithms includes stochastic variation reduction techniques that include SVRG [Johnson and Zhang, 2013], SAGA [Roux et al., 2012] and their variants listed out in [Nguyen et al., 2017]. For a detailed survey of different SGD variants, refer to [Ruder, 2016].

Synchronous SGD and Stragglers: To process large datasets, SGD is parallelized across multiple learners with a central PS. Each learner processes one mini-batch, and the PS aggregates all the gradients. The convergence of synchronous SGD is same as mini-batch SGD, with a P-fold larger mini-batch, where P is the number of learners. However, the time per iteration grows with the number of learners, because some straggling learners that slow down randomly [Dean and Barroso, 2013]. Thus, it is important to juxtapose the error reduction per iteration with the runtime per iteration to understand the true convergence speed of distributed SGD.

To deal with stragglers and speed up machine learning, system designers have proposed several straggler mitigation techniques such as [Harlap et al., 2016] that try to detect and avoid stragglers. An alternate direction of work is to use redundancy techniques as proposed in [Wang et al., 2015, Joshi et al., 2014, Lee et al., 2017, Tandon et al., 2017, Dutta et al., 2016, Yang et al., 2017, Ye and Abbe, 2018, Karakus et al., 2017] to deal with the stragglers.

Asynchronous SGD and Staleness: A complementary approach to deal with the issue of straggling is to use asynchronous SGD. In asynchronous SGD, any learner can evaluate the gradient and update the central PS without waiting for the other learners. Asynchronous variants of existing SGD algorithms have also been proposed and implemented in systems [Dean et al., 2012, Gupta et al., 2016, Cipar et al., 2013].

In general, analyzing the convergence of asynchronous SGD with the number of iterations is difficult in itself because of the randomness of gradient staleness. There are only a few pioneering works such as [Tsitsiklis et al., 1986, Lian et al., 2015, Mitliagkas et al., 2016, Recht et al., 2011, Mania et al., 2017, Chaturapruek et al., 2015, Zhang et al., 2016] in this direction. In [Tsitsiklis et al., 1986], a fully decentralized analysis was proposed that considers no central PS. In [Recht et al., 2011], a new asynchronous algorithm called Hogwild was proposed and analyzed under bounded gradient assumption that has been followed upon by several works such as [Lian et al., 2015]. In Hogwild, every learner only updates a part of the central parameter vector $\mathbf{w}$ and is thus essentially different in spirit from conventional asynchronous SGD [Lian et al., 2015] where every learner operates on the entire $\mathbf{w}$.

1.2 OUR CONTRIBUTIONS

Existing machine learning algorithms mostly try to optimize the trade-off of error with the number of iterations, epochs or "work complexity" [Bottou et al., 2016]. Time to complete a task has traditionally been calculated in terms of work complexity measures [Sedgewick and Wayne, 2011], where the time taken to complete a task is a deterministic function of the size of the task (number of operations). However, due to straggling and synchronization bottle-necks in the system, the same task can often take different time to compute across different learners or iterations. To the best of

our knowledge, the theoretical trade-off of error with runtime, modelling runtimes as random variables has not been studied. We bring statistical perspective to the traditional work complexity analysis that incorporates the randomness introduced due to straggling. In this paper, we provide a systematic approach to analyze the error with runtime for both synchronous and asynchronous SGD, and some variants like K-sync, K-batch-sync, K-async and K-batch-async SGD.

We also propose a new error convergence analysis for Async and K-async SGD that holds for strongly convex objectives and can also be extended to non-convex formulations. In this analysis we relax the bounded delay assumption in [Lian et al., 2015] and the bounded gradient assumption in [Recht et al., 2011]. We also remove the assumption of exponential computation time and the staleness process being independent of the parameter values [Mitliagkas et al., 2016] as we will elaborate in Section 3.2. Interestingly, our analysis also brings out the regimes where asynchrony can be better or worse than synchrony in terms of speed of convergence. Further, we propose a new learning rate schedule to compensate for staleness, and stabilize asynchronous SGD.

The rest of the paper is organized as follows. Section 2 describes our problem formulation introducing the system model and assumptions. Section 3 provides the main results of the paper – analytical characterization of runtime, new convergence analysis for Async and K-async SGD and the proposed learning rate schedule to compensate for staleness. The analysis of runtime is elaborated further in Section 4. Proofs and detailed discussions are presented in the Supplement.

2 PROBLEM FORMULATION

Our objective is to minimize the risk function of the parameter vector $\mathbf{w}$ as mentioned in (1) given N training samples. Let S denote the total set of N training samples, *i.e.*, a collection of some data points with their corresponding labels or values. We use the notation ξ to denote a random seed $\in S$ which consists of either a single data and its label or a single mini-batch (m samples) of data and their labels.

2.1 SYSTEM MODEL

We assume that there is a central parameter server (PS) with P parallel learners as shown in Figure 2. The learners fetch the current parameter vector $\mathbf{w}_j$ from the PS as and when instructed in the algorithm. Then they compute gradients using one mini-batch and push their gradients back to the PS as and when instructed in the algorithm. At each iteration, the

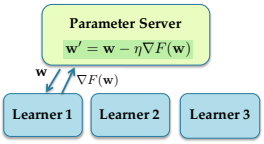

Figure 2: Parameter Server Model

PS aggregates the gradients computed by the learners and updates the parameter $\mathbf{w}$. Based on how these gradients are fetched and aggregated, we have different variants of synchronous or asynchronous SGD.

The time taken by a learner to compute gradient of one mini-batch is denoted by random variable X_i for $i = 1, 2, \ldots, P$. We assume that the X_is are i.i.d. across mini-batches and learners.

2.2 PERFORMANCE METRICS

There are two metrics of interest: Runtime and Error.

Definition 1 (Runtime). *The runtime of J iterations is the expected time to perform a total of J iterations.*

Definition 2 (Error). *The Error after j iterations is defined as $\mathbb{E}\left[F(\mathbf{w}_j) - F^*\right]$, the expected gap of the risk function from its optimal value.*

Our aim is to determine the trade-off between the error (measures the accuracy of the algorithm) and the runtime for the different SGD variants.

2.3 VARIANTS OF SGD

We now describe the SGD variants considered in this paper. Please refer to Figure 3 and Figure 4 for a pictorial illustration.

Figure 3: For $K = 2$ and $P = 3$, we illustrate the K-sync and K-batch-sync SGD in comparison with fully synchronous SGD. Lightly shaded arrows indicate straggling gradient computations that are cancelled.

K-sync SGD: This is a generalized form of synchronous SGD, also suggested in [Gupta et al., 2016, Chen et al., 2016] to offer some resilience to straggling as the PS does not wait for all the learners to finish. The PS only waits for the first K out of P learners to push their gradients. Once it receives K gradients, it updates $\mathbf{w}_j$ and cancels the remaining learners. The updated parameter vector $\mathbf{w}_{j+1}$ is sent to all P learners

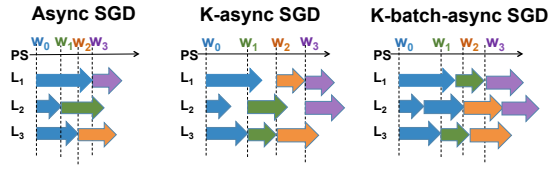

Figure 4: For $K = 2$ and $P = 3$, we illustrate the K-async and K-batch-async algorithms in comparison with fully asynchronous SGD.

for the next iteration. The update rule is given by:

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \frac{\eta}{K} \sum_{l=1}^{K} g(\mathbf{w}_j, \xi_{l,j}). \tag{2}$$

Here $\xi_{l,j}$ denotes the mini-batch of m samples used by the l-th learner at the j-th iteration and $g(\mathbf{w}_j, \xi_{l,j}) = \frac{1}{m} \sum_{\xi \in \xi_{l,j}} \nabla f(\mathbf{w}_j, \xi)$ denotes the average gradient of the loss function evaluated over the mini-batch $\xi_{l,j}$ of size m. For $K = P$, the algorithm is exactly equivalent to a fully synchronous SGD with P learners.

K-batch-sync: In K-batch-sync, all the P learners start computing gradients with the same $\mathbf{w}_j$. Whenever any learner finishes, it pushes its update to the PS and evaluates the gradient on the next mini-batch at the same $\mathbf{w}_j$. The PS updates using the first K mini-batches that finish and cancels the remaining learners. Theoretically, the update rule is still the same as (2) but here l now denotes the index of the mini-batch instead of the learner. However K-batch-sync will offer advantages over K-sync in runtime as no learner is idle.

K-async SGD: This is a generalized version of asynchronous SGD, also suggested in [Gupta et al., 2016]. In K-async SGD, all the P learners compute their respective gradients on a single mini-batch. The PS waits for the first K out of P that finish first, but it does not cancel the remaining learners. As a result, for every update the gradients returned by each learner might be computed at a stale or older value of the parameter $\mathbf{w}$. The update rule is thus given by:

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \frac{\eta}{K} \sum_{l=1}^{K} g(\mathbf{w}_{\tau(l,j)}, \xi_{l,j}). \tag{3}$$

Here $\xi_{l,j}$ is one mini-batch of m samples used by the l-th learner at the j-th iteration and $\tau(l,j)$ denotes the iteration index when the l-th learner last read from the central PS where $\tau(l,j) \leq j$. Also, $g(\mathbf{w}_{\tau(l,j)}, \xi_{l,j}) = \frac{1}{m} \sum_{\xi \in \xi_{l,j}} \nabla f(\mathbf{w}_{\tau(l,j)}, \xi_{l,j})$ is the average gradient of the loss function evaluated over the mini-batch $\xi_{l,j}$ based on the stale value of the parameter $\mathbf{w}_{\tau(l,j)}$. For $K = 1$, the algorithm is exactly equivalent to fully asynchronous SGD, and the update rule can be simplified as:

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \eta g(\mathbf{w}_{\tau(j)}, \xi_j). \tag{4}$$

Here ξ_j denotes the set of samples used by the learner that updates at the j-th iteration such that $|\xi_j| = m$ and $\tau(l,j)$ denotes the iteration index when that particular learner last read from the central PS. Note that $\tau(j) \leq j$.

K-batch-async: Observe in Figure 4 that K-async also suffers from some learners being idle while others are still working on their gradients until any K finish. In K-batch-async (proposed in [Lian et al., 2015]), the PS waits for K mini-batches before updating itself but irrespective of which learner they come from. So wherever any learner finishes, it pushes its gradient to the PS, fetches current parameter at PS and starts computing gradient on the next mini-batch based on the current value of the PS. Surprisingly, the update rule is again similar to (3) theoretically except that now l denotes the indices of the K mini-batches that finish first instead of the learners and $\mathbf{w}_{\tau(l,j)}$ denotes the version of the parameter when the learner computing the $l-$th mini-batch last read from the PS. While the error convergence of K-batch-async is similar to K-async, it reduces runtime as no learner is idle.

2.4 ASSUMPTIONS

Closely following [Bottou et al., 2016], we also make the following assumptions:

1. $F(\mathbf{w})$ is an $L-$ smooth function. Thus,

$$||\nabla F(\mathbf{w}_1) - \nabla F(\mathbf{w}_2)||_2 \leq L||\mathbf{w}_1 - \mathbf{w}_2||_2. \tag{5}$$

2. $F(\mathbf{w})$ is strongly convex with parameter c. Thus,

$$2c(F(\mathbf{w}) - F^*) \leq ||\nabla F(\mathbf{w})||_2^2 \quad \forall\ \mathbf{w}. \tag{6}$$

   Refer to Section 6 in Supplement for discussion on strong convexity. Our results also extend to non-convex objectives, as discussed in Section 3.

3. The stochastic gradient is an unbiased estimate of the true gradient:

$$\mathbb{E}_{\xi_j|\mathbf{w}_k}\left[g(\mathbf{w}_k, \xi_j)\right] = \nabla F(\mathbf{w}_k) \quad \forall\ k \leq j. \tag{7}$$

   Observe that this is slightly different from the common assumption that $\mathbb{E}_{\xi_j}\left[g(\mathbf{w}, \xi_j)\right] = \nabla F(\mathbf{w})$ for all $\mathbf{w}$. Observe that, all $\mathbf{w}_j$ for $j > k$ is actually not independent of the data ξ_j. We thus make the assumption more rigorous by conditioning on $\mathbf{w}_k$ for $k \leq j$. Our requirement $k \leq j$ means that $\mathbf{w}_k$ is the value of the parameter at the PS before the data ξ_j was accessed and can thus be assumed to be independent of the data ξ_j.

4. Similar to the previous assumption, we also assume that the variance of the stochastic update given $\mathbf{w}_k$

Table 1: List of Notations

Mini-batch Size	m
Total Iterations	J
Number of learners (Processors)	P
Number of learners to wait for	K
Learning rate	η
Lipschitz Constant	L
Strong-convexity parameter	c
Runtime of a learner for one mini-batch	X_i
Total runtime	T

at iteration k before the data point was accessed is also bounded as follows:

$$\mathbb{E}_{\xi_j|\mathbf{w}_k}\left[||g(\mathbf{w}_k,\xi_j)-\nabla F(\mathbf{w}_k)||_2^2\right] \leq \frac{\sigma^2}{m}+\frac{M_G}{m}||\nabla F(\mathbf{w}_k)||_2^2 \;\forall\; k\leq j. \quad (8)$$

3 MAIN RESULTS

3.1 RUNTIME ANALYSIS

We compare the theoretical wall clock runtime of the different SGD variants to illustrate the speed-up offered by different asynchronous and batch variants. A detailed discussion is provided in Section 4.

Theorem 1. *Let the wall clock time of each learner to process a single mini-batch be i.i.d. random variables $X_1, X_2, \ldots, X_P$. Then the ratio of the expected time of synchronous to asynchronous SGD is*

$$\frac{\mathbb{E}\left[T_{Sync}\right]}{\mathbb{E}\left[T_{Async}\right]} = P\frac{\mathbb{E}\left[X_{P:P}\right]}{\mathbb{E}\left[X\right]}$$

where $X_{(P:P)}$ is the P^{th} order statistic of P i.i.d. random variables $X_1, X_2, \ldots, X_P$.

This is the first result that analytically characterizes the speed-up offered by asynchronous SGD. To prove this result, we use ideas from renewal theory as we discuss in Section 4. In the following corollary, we highlight this speed-up for the special case of exponential computation time.

Corollary 1. *Let the wall clock time of each learner to process a single mini-batch be i.i.d. exponential random variables $X_1, X_2, \ldots, X_P \sim \exp(\mu)$. Then the ratio of the expected time of synchronous to asynchronous is approximately given by $P\log P$.*

Thus, the speed-up scales with P and can diverge to infinity for large P. We illustrate the speed-up for different distributions in Figure 5.

The next result illustrates the advantages offered by K-batch-sync and async over their corresponding counterparts K-sync and K-async respectively.

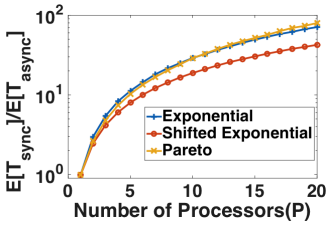

Figure 5: Plot of the speed-up using asynchronous over synchronous: $\log\frac{\mathbb{E}[T_{Sync}]}{\mathbb{E}[T_{Async}]}$ with P for different distributions - $\exp(1)$, $1+\exp(1)$ and $Pareto(2,1)$.

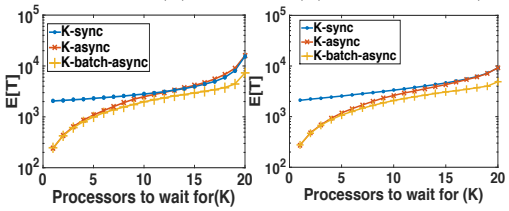

Figure 6: Plot of runtime $\mathbb{E}[T]$ for 2000 iterations: (Left) Pareto distribution $Pareto(2,1)$ and (Right) Shifted exponential distribution $1+\exp(1)$.

Theorem 2. *Let the wall clock time of each learner to process a single mini-batch be i.i.d. exponential random variables $X_1, X_2, \ldots, X_P \sim \exp(\mu)$. Then the ratio of the expected time of K-async (or sync) SGD to K-batch-async (or sync) SGD is given by*

$$\frac{\mathbb{E}\left[T_{K-async}\right]}{\mathbb{E}\left[T_{K-batch-async}\right]} = \frac{P\mathbb{E}\left[X_{K:P}\right]}{K\mathbb{E}\left[X\right]} \approx \frac{P\log\frac{P}{P-K}}{K}$$

where $X_{K:P}$ is the K^{th} order statistic of i.i.d. random variables $X_1, X_2, \ldots, X_P$.

To prove this, we derive an exact expression for the runtime of K-batch-async SGD using renewal theory, for *any* given i.i.d. distribution of X_is, not necessarily exponential. The runtime of K-batch-async SGD is given by $\frac{JK\mathbb{E}[X]}{P}$ (see Lemma 4 in Section 4). The proof of Theorem 2 is also provided in in Section 4.

Theorem 2 shows that as $\frac{K}{P}$ increases, the speed-up using K-batch-async increases and can be upto $\log P$ times higher. For non-exponential distributions, we simulate the behaviour of $\mathbb{E}[T]$ in Figure 6 for K-sync, K-async and K-batch-async respectively for Pareto and Shifted Exponential.

3.2 ERROR ANALYSIS UNDER FIXED LEARNING RATE

Theorem 3 below gives a convergence analysis of K-async SGD for fixed η, relaxing the following assumptions in existing literature.

- [Mitliagkas et al., 2016] assumes that X_i's are exponentially distributed. Our analysis holds for any general service time X_i.

- [Mitliagkas et al., 2016] also assumes that the staleness process is independent of $\mathbf{w}$. While this assumption simplifies the analysis greatly, it is not true in practice. For instance, for a two learner case, the parameter $\mathbf{w}_2$ after 2 iterations depends on whether the update from $\mathbf{w}_1$ to $\mathbf{w}_2$ was based on a stale gradient at $\mathbf{w}_0$ or the current gradient at $\mathbf{w}_1$, depending on which learner finished first. In this work, we remove this independence assumption.

- Instead of the bounded delay assumption in [Lian et al., 2015], we use a general staleness bound $\mathbb{E}\left[||\nabla F(\mathbf{w}_j) - \nabla F(\mathbf{w}_{\tau(l,j)})||_2^2\right] \leq \gamma \mathbb{E}\left[||\nabla F(\mathbf{w}_j)||_2^2\right]$, which allows for large, but rare delays.

- In [Recht et al., 2011], the norm of the gradient is assumed to be bounded. However, if $||\nabla F(\mathbf{w})||_2^2 \leq M$ for some constant M, then using (6) we obtain $||\mathbf{w} - \mathbf{w}^*||_2^2 \leq \frac{2}{c}(F(\mathbf{w}) - F^*) \leq \frac{M}{c^2}$ implying that the range of $\mathbf{w}$ is bounded which is a very strong and restrictive assumption, that we relax in this result.

Theorem 3. *Suppose the objective $F(\mathbf{w})$ is c-strongly convex and the learning rate $\eta \leq \frac{1}{2L(\frac{M_G}{Km}+\frac{1}{K})}$. Also assume that $\mathbb{E}\left[||\nabla F(\mathbf{w}_j) - \nabla F(\mathbf{w}_{\tau(l,j)})||_2^2\right] \leq \gamma \mathbb{E}\left[||\nabla F(\mathbf{w}_j)||_2^2\right]$, for some $\gamma \leq 1$. Then, the error of K-async SGD after J iterations is,*

$$\mathbb{E}\left[F(\mathbf{w}_J)\right] - F^* \leq \frac{\eta L \sigma^2}{2c\gamma' Km} + (1 - \eta c\gamma')^J \left(\mathbb{E}\left[F(\mathbf{w}_0)\right] - F^* - \frac{\eta L \sigma^2}{2c\gamma' Km}\right) \quad (9)$$

where $\gamma' = 1 - \gamma + \frac{p_0}{2}$ and p_0 is a lower bound on the conditional probability that $\tau(l,j) = j$, given all the past delays and parameters.

Here, γ is a measure of staleness of the gradients returned by learners; smaller γ indicates a less staleness.

We use Lemma 1 to prove Theorem 3. See Section 8.1 and Section 8.2 in Supplement for the full proof.

Lemma 1. *Suppose that $p_0^{(l,j)}$ is the conditional probability that $\tau(l,j) = j$ given all the past delays and all the previous $\mathbf{w}$, and $p_0 \leq p_0^{(j)}$ for all j. Then,*

$$\mathbb{E}\left[||\nabla F(\mathbf{w}_{\tau(l,j)})||_2^2\right] \geq p_0 \mathbb{E}\left[||\nabla F(\mathbf{w}_j)||_2^2\right]. \quad (10)$$

Proof. By the law of total expectation,

$$\begin{aligned}\mathbb{E}\left[||\nabla F(\mathbf{w}_{\tau(l,j)})||_2^2\right] &= p_0^{(l,j)} \mathbb{E}\left[||\nabla F(\mathbf{w}_{\tau(l,j)})||_2^2 | \tau(j) = j\right] \\ &\quad + (1 - p_0^{(l,j)}) \mathbb{E}\left[||\nabla F(\mathbf{w}_{\tau(l,j)})||_2^2 | \tau(j) \neq j\right] \\ &\geq p_0 \mathbb{E}\left[||\nabla F(\mathbf{w}_j)||_2^2\right].\end{aligned}$$

□

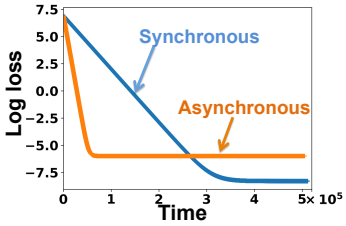

Figure 7: Theoretical error-runtime trade-off for Sync and Async-SGD with same η. Async-SGD has faster decay with time but a higher error floor.

For the exponential distribution, p_0 is equal to $\frac{1}{P}$ (see Section 8.1.1 in Supplement). For non-exponential distributions, it is a constant in $[0, 1]$. For some special classes of distributions like new-longer-than-used (new-shorter-than-used) defined in Definition 3, we can formally show that p_0 lies in $[0, \frac{1}{P}]$ ($[\frac{1}{P}, 1]$) respectively. Refer to Section 8.1.1 in Supplement for the proof.

For K-batch-async, the update rule is same as K-async except that the index l denotes the index of the mini-batch. Thus, the error analysis will be exactly similar. Our analysis can also be extended to non-convex $F(\mathbf{w})$ as we show in Section 8.2.1 in the Supplement.

Now let us compare with K-sync SGD. We observe that the analysis of K-sync SGD is same as serial SGD with mini-batch size Km. Thus,

Lemma 2 (Error of K-sync)**.** *[Bottou et al., 2016] Suppose that the objective $F(\mathbf{w})$ is c-strongly convex and learning rate $\eta \leq \frac{1}{2L(\frac{M_G}{Km}+1)}$. Then, the error after J iterations of K-sync SGD is*

$$\mathbb{E}\left[F(\mathbf{w}_J) - F^*\right] \leq \frac{\eta L \sigma^2}{2c(Km)} + (1 - \eta c)^J \left(F(\mathbf{w}_0) - F^* - \frac{\eta L \sigma^2}{2c(Km)}\right).$$

Can stale gradients win the race? For the same η, observe that the error given by Theorem 3 decays at the rate $(1 - \eta c(1 - \gamma + \frac{p_0}{2}))$ for K-async or K-batch-async SGD while for K-sync, the decay rate with number of iterations is $(1 - \eta c)$. Thus, depending on the values of γ and p_0, the decay rate of K-async or K-batch-async SGD can be faster or slower than K-sync SGD. The decay rate of K-async or K-batch-async SGD is faster if $\frac{p_0}{2} > \gamma$. As an example, one might consider an exponential or new-shorter-than-used service time where $p_0 \geq \frac{1}{P}$ and γ can be made smaller by increasing K. It might be noted that asynchronous SGD can still be faster than synchronous SGD with respect to wall clock time even if its decay rate with respect to number of iterations is lower as every iteration is much faster in asynchronous SGD (Roughly $P \log P$ times faster for exponential service times).

The maximum allowable learning rate for syn-

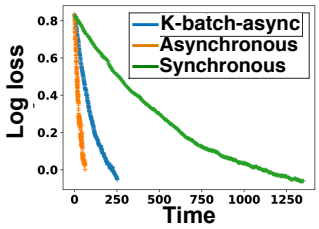

Figure 8: Error-runtime trade-off comparison of different SGD variants for logistic regression on MNIST, with $X_i \sim \exp(1)$, $P = 8$, $K = 4$, $\eta = 0.01$ and $m = 1$. K-batch-async gives intermediate performance, between Async and sync-SGD (see Section 9 in Supplement).

chronous SGD is $\max\{\frac{1}{c}, \frac{1}{2L(\frac{M_G}{Pm}+1)}\}$ which can be much higher than that for asynchronous SGD,*i.e.*, $\max\{\frac{1}{c(1-\gamma+\frac{p_0}{2})}, \frac{1}{2L(\frac{M_G}{m}+1)}\}$. Similarly the error-floor for synchronous is $\frac{\eta L\sigma^2}{2cPm}$ as compared to asynchronous whose error floor is $\frac{\eta L\sigma^2}{2c(1-\gamma+\frac{p_0}{2})m}$.

In Figure 7, we compare the theoretical trade-offs between synchronous ($K = P$ in Lemma 2) and asynchronous SGD ($K = 1$ in Theorem 3). Async-SGD converges very quickly, but to a higher floor. Figure 8 shows the same comparison on the MNIST dataset, along with K-batch-async SGD.

3.3 VARIABLE LEARNING RATE FOR STALENESS COMPENSATION

The staleness of the gradient is random, and can vary across iterations. Intuitively, if the gradient is less stale, we want to weigh it more while updating the parameter $\mathbf{w}$, and if it is more stale we want to scale down its contribution to the update. With this motivation, we propose the following condition on the learning rate at different iterations.

$$\eta_j \mathbb{E}\left[||\mathbf{w}_j - \mathbf{w}_{\tau(j)}||_2^2\right] \leq C \tag{11}$$

for a constant C. This condition is also inspired from our error analysis in Theorem 3, because it helps remove the assumption $\mathbb{E}\left[||\nabla F(\mathbf{w}_j) - \nabla F(\mathbf{w}_{\tau(j)})||_2^2\right] \leq \gamma \mathbb{E}\left[||\nabla F(\mathbf{w}_j)||_2^2\right]$. Our convergence result is as follows:

Theorem 4. *Suppose the learning rate in the j-th iteration $\eta_j \leq 1/2L(\frac{M_G}{m}+1)$, and $\eta_j \mathbb{E}\left[||\mathbf{w}_j - \mathbf{w}_{\tau(j)}||_2^2\right] \leq C$ for some constant C. Then, we have*

$$\mathbb{E}\left[F(\mathbf{w}_J)\right] - F^* \leq \Delta + \left(\mathbb{E}\left[F(\mathbf{w}_0)\right] - F^*\right) \prod_{j=1}^{J} (1 - \rho_j)$$

where $\rho_j = \eta_j(1 + \frac{p_0}{2})c$, and the error floor $\Delta = \Delta_J + (1-\rho_J)\Delta_{J-1} + \cdots + \prod_{j=1}^{J}(1-\rho_j)\Delta_0$, with $\Delta_j = \frac{\eta_j^2 L\sigma^2}{2m} + \frac{CL^2}{2}$.

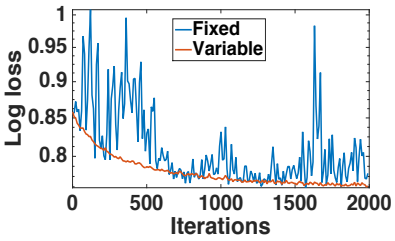

Figure 9: Async-SGD on CIFAR10 dataset, with $X \sim \exp 20$, mini-batch size $m = 250$ and $P = 40$ learners. We compare fixed $\eta = 0.01$, and the variable schedule given in (13) for $\eta_{max} = 0.01$ and $C = 0.005\eta_{max}$. The proposed schedule can give fast convergence, and also maintain stability, while the fixed η algorithm becomes unstable (see Section 9 in Supplement for setup details).

The proof is provided in Section 8.3 in the Supplement. In our analysis of Asynchronous SGD, we observe that the term $\frac{\eta}{2}\mathbb{E}\left[||\nabla F(\mathbf{w}_j) - \nabla F(\mathbf{w}_{\tau(j)})||_2^2\right]$ is the most difficult to bound. For fixed learning rate, we had assumed that $\mathbb{E}\left[||\nabla F(\mathbf{w}_j) - \nabla F(\mathbf{w}_{\tau(j)})||_2^2\right]$ is bounded by $\gamma||\nabla F(\mathbf{w}_j)||_2^2$. However, if we impose the condition (11) on η, we do not require this assumption. Our proposed condition actually provides a bound for the staleness term as follows:

$$\frac{\eta_j}{2}\mathbb{E}\left[||\nabla F(\mathbf{w}_j) - \nabla F(\mathbf{w}_{\tau(j)})||_2^2\right] \leq \frac{\eta_j L^2}{2}\mathbb{E}\left[||\mathbf{w}_j - \mathbf{w}_{\tau(j)}||_2^2\right] \leq \frac{CL^2}{2}. \tag{12}$$

Proposed Algorithmic Modification Inspired by this analysis, we propose the learning rate schedule,

$$\eta_j = \min\left\{\frac{C}{||\mathbf{w}_j - \mathbf{w}_{\tau(j)}||_2^2}, \eta_{max}\right\}, \tag{13}$$

where η_{max} is a suitably large ceiling on learning rate. It ensures stability when the first term in (13) becomes large due to the staleness $||\mathbf{w}_j - \mathbf{w}_{\tau(j)}||_2$ being small. The C is chosen of the same order as the desired error floor. To implement this schedule, the PS needs to store the last read model parameters for every learner. In Figure 9 we illustrate how this schedule can stabilize asynchronous SGD.

4 RUNTIME ANALYSIS

In this section, we provide our results on the runtime of different variants of SGD. These lemmas are then used in the proofs of Theorem 1 and Theorem 2.

4.1 RUNTIME OF K-SYNC SGD

Lemma 3 (Runtime of K-sync SGD)**.** *The expected time taken by K-sync SGD to complete J iterations is,*

$$\mathbb{E}\left[T\right] = J\mathbb{E}\left[X_{K:P}\right] \tag{14}$$

where $X_{K:P}$ is the K^{th} order statistic of P i.i.d. random variables $X_1, X_2, \ldots, X_P$.

The proof is provided in Section 7.1 in the Supplement.

Remark: For $X_i \sim \exp(\mu)$, the runtime for J iterations is $\mathbb{E}[T] = \frac{J}{\mu}\sum_{i=P-K+1}^{P}\frac{1}{i} \approx \frac{J}{\mu}\left(\log\frac{P}{P-K}\right)$ [Sheldon, 2002]. Refer to Section 7.1.1 in Supplement.

The runtime of K-batch-sync SGD is not tractable in general, but for $X_i \sim \exp(\mu)$, the time per iteration is distributed as $Erlang(K, P\mu)$. Thus, $\mathbb{E}[T] = J\frac{K}{P\mu}$.

4.2 RUNTIME OF K-BATCH-ASYNC SGD

Lemma 4 (Runtime of K-batch-async SGD)**.** *The expected time taken by K-batch-async SGD for J iterations is given by:*

$$\mathbb{E}[T] = J\frac{K\mathbb{E}[X]}{P}. \tag{15}$$

To prove the result we use ideas from renewal theory. A brief background on renewal theory is provided in Section 7.2 in the Supplement for completeness.

Proof of Lemma 4. For the i-th learner, let $\{N_i(t), t > 0\}$ be the number of times the i-th learner pushes its gradient to the PS over in time t. The time between two pushes is an independent realization of X_i. Thus, the inter-arrival times $X_i^{(1)}, X_i^{(2)}, \ldots$ are i.i.d. with mean inter-arrival time $\mathbb{E}[X_i]$. Using the elementary renewal theorem [Gallager, 2013, Chapter 5] we have,

$$\lim_{t\to\infty}\frac{\mathbb{E}[N_i(t)]}{t} = \frac{1}{\mathbb{E}[X_i]}. \tag{16}$$

Thus, the rate of gradient pushes by the i-th learner is $1/\mathbb{E}[X_i]$. As there are P learners, the rate of gradient pushes to the PS is

$$\lim_{t\to\infty}\sum_{i=1}^{P}\frac{\mathbb{E}[N_i(t)]}{t} = \sum_{i=1}^{P}\frac{1}{\mathbb{E}[X_i]} = \frac{P}{\mathbb{E}[X]}. \tag{17}$$

Every K pushes are one iteration. Thus, the time to complete J iterations or effectively JK pushes is given by $\mathbb{E}[T] = \frac{JK\mathbb{E}[X]}{P}$. □

Now, we use these lemmas to prove Theorem 1 below.

Proof of Theorem 1. For $K = 1$, K-batch-async reduces to asynchronous SGD, and its runtime $\mathbb{E}[T] = J\frac{\mathbb{E}[X]}{P}$. By taking the ratio of the runtimes in Lemma 3 with $K = P$ and Lemma 4 with $K = 1$, we get the result in Theorem 1. □

Corollary 1 also follows by substituting in Theorem 1 that for $X_i \sim \exp(\mu)$, $\mathbb{E}[X_{P:P}] = \sum_{i=1}^{P}\frac{1}{i\mu} \approx \frac{\log P}{\mu}$.

4.3 RUNTIME OF K-ASYNC SGD

The runtime of K-async SGD is not tractable for non-exponential X_i, but we obtain an upper bound on it for "new-longer-than-used" distributions, defined below.

Definition 3 (New-longer-than-used)**.** *A random variable is said to have a new-longer-than-used distribution if the following holds for all $t, u \geq 0$:*

$$\Pr(U > u + t | U > t) \leq \Pr(U > u)$$

Most of the continuous distributions we encounter like normal, exponential, gamma, beta are new-longer-than-used. Alternately, the hyper exponential distribution is new-shorter-than-used and it satisfies $\Pr(U > u+t|U > t) \geq \Pr(U > u)$ for all $t, u \geq 0$.

Lemma 5 (Runtime of K-async SGD)**.** *Suppose that each X_i has a new longer than used distribution. Then, the expected time taken to complete J iterations by K-async is upper-bounded as*

$$\mathbb{E}[T] \leq J\mathbb{E}[X_{K:P}] \tag{18}$$

where $X_{K:P}$ is the K^{th} order statistic of P i.i.d. random variables $X_1, X_2, \ldots, X_P$.

The proof is provided in Section 7.3. We use this lemma to prove Theorem 2 below. A simulation comparing the error-runtime trade-off of K-async and K-batch-async variants is provided in Section 7.3.2 in the Supplement.

Proof of Theorem 2. For the exponential X_i, equality holds in (18) in Lemma 5, as we justify in Section 7.3.1 in the Supplement. The expectation can thus be derived as $\mathbb{E}[X_{K:P}] = \sum_{i=P-K+1}^{P}\frac{1}{i\mu} \approx \frac{\log(P/P-K)}{\mu}$. For exponential X_i, the runtime of K-batch-async is $\mathbb{E}[T] = J\frac{K\mathbb{E}[X]}{P} = J\frac{K}{\mu P}$ from Lemma 4. □

5 CONCLUSIONS

The speed of distributed SGD depends on the error reduction per iteration, as well as the runtime per iteration. To the best of our knowledge, this paper presents the first runtime analysis of synchronous and asynchronous SGD, and their variants. When juxtaposed with the error analysis, we get error-runtime trade-offs that can be used to compare different SGD algorithms. We also give a new analysis of asynchronous SGD by relaxing some commonly made assumptions, and a novel learning rate schedule to compensate for gradient staleness. In the future we plan to explore methods to gradually increase synchrony, so that we can achieve fast convergence as well as low error floor.

Acknowledgements: The work was done when Gauri Joshi was a Research Staff Member at IBM and Sanghamitra Dutta was an intern.

References

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *arXiv:1606.04838*, 2016.

S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

S. Chaturapruek, J. C. Duchi, and C. Ré. Asynchronous stochastic convex optimization: the noise is in the noise and sgd don't care. In *Advances in Neural Information Processing Systems*, pages 1531–1539, 2015.

J. Chen, R. Monga, S. Bengio, and R. Józefowicz. Revisiting distributed synchronous SGD. *CoRR*, abs/1604.00981, 2016. URL http://arxiv.org/abs/1604.00981.

J. Cipar, Q. Ho, J. K. Kim, S. Lee, G. R. Ganger, G. Gibson, K. Keeton, and E. Xing. Solving the straggler problem with bounded staleness. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2013.

J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13 (1):165–202, 2012.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2, July 2011.

S. Dutta, V. Cadambe, and P. Grover. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pages 2100–2108, 2016.

R. Gallager. *Stochastic Processes: Theory for Applications*. Cambridge University Press, 1st edition, 2013.

S. Gupta, W. Zhang, and F. Wang. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *Proceedings of the International Conference on Data Mining*, pages 171–180, Dec. 2016.

A. Harlap, H. Cui, W. Dai, J. Wei, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing. Addressing the straggler problem for iterative convergent parallel ml. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, pages 98–111, 2016.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

G. Joshi, Y. Liu, and E. Soljanin. On the delay-storage trade-off in content download from coded distributed storage systems. *IEEE Journal on Selected Areas in Communications*, 32(5):989–997, 2014.

C. Karakus, Y. Sun, S. Diggavi, and W. Yin. Straggler mitigation in distributed optimization through data encoding. In *Advances in Neural Information Processing Systems*, pages 5440–5448, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

D. M. Kreps. *A course in microeconomic theory*, volume 41. JSTOR, 1990.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 2017.

M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 661–670, 2014.

X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.

I. Mitliagkas, C. Zhang, S. Hadjis, and C. Ré. Asynchrony begets momentum, with an application to deep learning. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*, pages 997–1004. IEEE, 2016.

L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. In *International Conference on Machine Learning*, pages 2613–2621, 2017.

B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

N. L. Roux, M. Schmidt, and F. R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley Professional, 2011.

R. Sheldon. *A first course in probability*. Pearson Education India, 2002.

R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376, 2017.

J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.

D. Wang, G. Joshi, and G. Wornell. Using straggler replication to reduce latency in large-scale parallel computing. *ACM SIGMETRICS Performance Evaluation Review*, 43(3):7–11, 2015.

Y. Yang, P. Grover, and S. Kar. Coded distributed computing for inverse problems. In *Advances in Neural Information Processing Systems*, pages 709–719, 2017.

M. Ye and E. Abbe. Communication-computation efficient gradient coding. *arXiv preprint arXiv:1802.03475*, 2018.

W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-sgd for distributed deep learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2350–2356. AAAI Press, 2016.