Asynchronous Distributed Optimization with Randomized Delays

Margalit Glasgow

Mary Wootters

October 2, 2020

Abstract

In this work, we study asynchronous finite sum minimization in a distributed-data setting with a central parameter server. While asynchrony is well understood in parallel settings where the data is accessible by all machines—e.g., modifications of variance-reduced gradient algorithms like SAGA work well—little is known for the distributed-data setting. We introduce a variant of SAGA called ADSAGA for the distributed-data setting where each machine stores a partition of the data. We show that with independent exponential work times—a common assumption in distributed optimization—ADSAGA converges in \( O((n + \sqrt{m\kappa}) \log(1/\epsilon)) \) iterations, where \( n \) is the number of component functions, \( m \) is the number of machines, and \( \kappa \) is a condition number. We empirically compare the iteration complexity of ADSAGA to existing parallel and distributed algorithms, including synchronous minibatch algorithms.

1 Introduction

In this paper we consider the finite sum minimization problem common in many empirical risk minimization (ERM) problems:

\[
\min_{x \in \mathbb{R}^d} f(x) := \sum_{i=1}^{n} f_i(x),
\]

where each \( f_i \) is convex and \( L \)-smooth, and \( f \) is \( L_f \)-smooth and \( \mu \)-strongly convex. For instance, in machine learning problems, each \( f_i \) might represent a loss function evaluated at a data point. A typical strategy for minimizing finite sums is variance-reduced stochastic gradient algorithms, such as SVRG [16] or SAGA [10], which achieve linear (geometric) convergence rates. In distributed memory settings with a central parameter server (PS), this minimization task is distributed among \( m \) cooperating machines [9, 19]. Each machine holds some subset of the function, and the central PS holds \( x \). This is common in cloud computing clusters and networks of mobile or IoT-enabled devices [3].

One approach in this distributed setting is to parallelize a stochastic gradient descent algorithm by computing minibatch gradient updates [35] [12]. At each iteration, the PS sends the current iterate \( x_k \) to each machine, and each machine computes a gradient update from the functions it holds, evaluated at \( x_k \). Each machine then sends this gradient update to the PS, which sums up the gradient updates and computes the next iterate.

While these parallel minibatch algorithms can speed up gradient descent, the synchrony requires the PS to wait for all of the machines to finish their computation before taking a gradient step, leading to idle time if there is any variance in work time. Due to the presence of stragglers and high variance in work time observed in [32, 8, 18], work beginning in [5, 26, 25] has proposed asynchronous parallel gradient descent algorithms to mitigate this issue.
In asynchronous algorithms, a gradient step is performed on $x_k$ as soon as a single machine completes its computation. Hence the gradient updates performed at the PS may come from delayed gradients computed at stale copies of the parameter $x$. We denote this stale copy by $x_{k - \tau(k)}$, meaning that it is $\tau(k)$ iterations old. Formally, at iteration $k$, the PS performs the update
\[
x_{k+1} = x_k - \eta U(i_k, x_{k - \tau(k)}),
\]
where $\eta$ is the learning rate, and $U(i_k, x_{k - \tau(k)})$ is an update computed from the gradient $\nabla f_{i_k}(x_{k - \tau(k)})$. For example, in SGD, we would have $U(i, x) = \nabla f_i(x)$.

In asynchronous settings where the data is shared, meaning any machine can access any function $f_i$ at any time (see Figure 1(a)), much is known. An asynchronous version of SAGA, called ASAGA [17] is shown to converge in $O(n + \frac{\tau_{max}}{\mu})$ iterations if all delays are bounded by $\tau_{max}$. This algorithm uses the update
\[
U(i, x) := \nabla f_i(x) - \alpha_i + \overline{\alpha},
\]
where $\alpha_i$ is the last computed gradient of $f_i$, and $\overline{\alpha}$ denotes the average $\frac{1}{n} \sum_i \alpha_i$. Other asynchronous variance-reduced algorithms are studied in [21, 33, 27, 34], achieving similar rates. In certain cases, when the gradient updates are sparse or the minimization problem is well-conditioned, these algorithms exhibit a parallel speed-up from increasing the number of machines.

The asynchronous landscape is far less understood when the data is distributed, meaning that each machine can only access the set of functions $f_i$ stored at that machine (as in Figure 1(b)). Several works [14, 31, 4] analyze an asynchronous incremental aggregation gradient (IAG) algorithm and prove it converges deterministically in $O(\frac{\tau_{max}}{\mu})$ iterations. A line of work which considers a completely decentralized architecture without a PS (see Figure 1(c)) generalizes the distributed data PS setting of Figure 1(b). In decentralized architectures, it is common to assume that each machine computes gradient updates and communicates with its neighbors at an independent exponential rate [25, 15, 20], also known as randomized gossip. The work [20] proved sublinear convergence rates for decentralized SGD with randomized gossip. Variance-reduced algorithms, which could yield better rates for strongly convex objectives, have not been studied in this model.

The discrepancy between convergence rates for algorithms with shared data versus distributed data, that is $O(n + \frac{\tau_{max}}{\mu})$ versus $O(\frac{n^2}{\mu})$, motivates the following question.

**Question 1.** Can asynchronous approaches with distributed data (Figure 1(b)) do as well as with shared data (Figure 1(a))?}

This question is challenging because a key property of the shared data setting is the independence of the delay $\tau(k)$ and the data sample $i_k$, which is chosen randomly by the machine computing the update. This leads to an unbiased gradient condition, central to the analyses of [21, 17, 33, 34, 27]:
\[
\mathbb{E}[U(i_k, x_{k - \tau(k)})|x_{k - \tau(k)}] = \nabla f(x_{k - \tau(k)}).
\]

When the functions $f_i$ are distributed among $m$ machines, it is impossible to achieve an unbiased gradient. If the only assumption on the delays is that they are bounded, then using the standard gradient update in $U(i, x) = \nabla f_i(x)$ may not even yield asymptotic convergence to $x^*$.

For this reason, we believe a better distributed data analog to the setting of [21, 17, 33, 34, 27] assumes some symmetry among the delays induced by each machine. That is, we assume that after a machine reads an iterate $x$ from the PS, its work time to compute a stochastic gradient and then write to the PS is a random variable, and the set of work times are i.i.d.

In this paper, we take a first step towards understanding Question 1 by studying settings where the work times are independent exponential random variables. Exponential work times have been studied before in the context of asynchronous optimization in [22, 20, 25, 15]. This delay model is tractable to analyze because it has the property that $i_k$ is independent from the iteration $k$. While not entirely realistic, experiments in parallel computing suggest that the shifted exponential distribution well estimates work times [18], so we conduct empirical simulations both with the exponential and the shifted exponential distributions.
1.1 Contributions

Our main technical contribution is the analysis of a SAGA-like algorithm, which we call ADSAGA, in an asynchronous distributed setting with random exponential work times. We show that with \( m \) machines, for \( \mu \)-strongly convex, \( L_f \)-smooth functions \( f \) with minimizer \( x^* \), when each \( f_i \) is \( L \)-smooth, we converge to

\[
|f(x_k) - f(x^*)| \leq \epsilon \text{ in } k = \tilde{O}\left( \left( n + \frac{L}{\mu} + \frac{\sqrt{mL_fL}}{\mu} \right) \log(1/\epsilon) \right)
\]  

(4)

iterations. Standard sequential SAGA achieves the same convergence in \( O\left( (n + \frac{L}{\mu}) \log(1/\epsilon) \right) \) iterations. This implies that our algorithm achieves a linear parallel speed-up in the number of machines \( m \) when \( m \leq \max\left( \frac{n^2 \mu^2}{L_fL}, \frac{L_f}{L} \right) \). On the other hand, when \( m \) exceeds this value and hence the third term in Equation (4) dominates, the convergence rate scales with the square root of the number of machines, or average delay.

Remarkably, because of this \( \sqrt{m} \) dependence, the convergence rate in Equation (4) is dramatically faster than the rates proved for the shared data analog ASAGA \([17]\), \( O(n + \frac{m \tau_{\max}}{\mu}) \), or even the rate of synchronous minibatch SAGA, \( \tilde{O}(n + \frac{L_f}{\mu} + \frac{m L_f}{\mu}) \) \([24]\). Indeed, in ASAGA, the convergence rate scales with the maximum delay \( \tau_{\max} \), which is lower bounded by \( m \). The fast convergence rate we give for ADSAGA is possible because of the assumption of exponential work times. Because of the occasional occurrence of super short work times, after the same number of iterations, the parallel depth of ADSAGA with exponential work times is far deeper than that of a synchronous minibatch algorithm or than some instantiations of ASAGA with bounded delays.

The proof of our result uses a novel Lyapunov function to track our progress towards the optimum. In addition to including typical terms such as \( f(x_t) - f(x^*) \) and \( |x_t - x^*|^2 \), our potential function includes a quadratic term that takes into account the dot product of \( x_t - x^* \) and the expected next stale gradient update. This quadratic term is similar to the one that appears in the Lyapunov analysis of SAG \([28]\). Key to our analysis is a new unbiasedness condition, which states that in expectation, the expected stale update moves towards the true gradient.

We support our theoretical claims with numerical experiments, which show how the convergence rate changes both when the random delays deviate from our theoretical exponential model, and when the number of machines changes. We show empirically that in linear regression, where frequently \( L_f \ll L \), we achieve parallel speedups from increasing \( m \) as large as \( n \). Our empirical results — based on i.i.d. work times from shifted exponential distributions — suggest that asynchronous distributed SAGA (ADSAGA) performs nearly as well as ASAGA, and better than previously proposed algorithms for the distributed setting such as IAG \([14]\) and SGD \([20]\).

1.2 Related Work

In this section, we survey the most related gradient-based asynchronous algorithms for strongly convex optimization. See Table [4] for a quantitative summary of the most relevant comparisons to our work.

**Synchronous Parallel Stochastic Algorithms.** Synchronous parallel stochastic gradient descent algorithms can be thought of as minibatch variants of their non-parallel counterparts. minibatch SGD
is analyzed in [35, 11]. For finite sum minimization, minibatch SAGA is analyzed in [12, 6, 13], achieving a convergence rate of $O\left((n + \frac{L}{\mu} + \frac{mL}{\mu}) \log(1/\epsilon)\right)$ for a minibatch size of $m$ [24]. Katyusha [1] presents an accelerated, variance reduced parallelizable algorithm for finite sums with convergence rate $O\left((n + \sqrt{\frac{nL}{\mu}} + m\sqrt{\frac{L}{\mu}}) \log(1/\epsilon)\right)$; this rate is proved to be near-optimal in [24].

**Asynchronous Centralized Algorithms (Figure 1(a))** Centralized asynchronous algorithms often arise in shared-memory architectures or in compute systems with a central parameter server. The seminal textbook [5] shows asymptotic convergence for stochastic optimization in *totally asynchronous* settings which may have unbounded delays. In the *partially asynchronous* setting, where delays are arbitrary but bounded by some value $\tau$, sublinear convergence rates of $O(\frac{1}{\epsilon})$ matching those of SGD were achieved for strongly convex stochastic optimization in [20] (under sparsity assumptions) and [7]. For finite sum minimization, linear (geometric) convergence is proved for asynchronous variance-reduced algorithms in [21, 33, 27, 17, 34]; the best known rate of $O\left((n + \frac{L}{\mu} + \tau) \log(1/\epsilon)\right)$ is achieved by ASAGA [17] and MIG [34], though these works provide stronger guarantees under sparsity assumptions. Note that most of these works can be applied to lock-free shared-memory architectures as they do not assume consistent reads of the central parameter. In [2], the authors consider the setting where all delays are exactly equal to $\tau$. All of the works discussed above assume that each gradient update is computed from a uniformly random function $f_i$.

**Asynchronous Centralized Algorithms with Distributed Data (Figure 1(b))** Several works [14, 4, 31] consider incremental aggregation algorithms, which use the update $U(i, x) = \nabla f_i(x) + \sum_{i' \neq i} \alpha_{i'}$, and can be applied to the distributed data setting. All of these works yield convergence rates that are quadratic in the maximum delay between computations of $\nabla f_i$. Note that this delay is lower bounded by $n$ if a single new gradient is computed at each iteration. The bounds in these works are deterministic, and hence cannot leverage any stochasticity in the gradient computed locally at each machine, which is natural when $n > m$ and each machine holds many functions $f_i$.

**Asynchronous Decentralized Algorithms with Distributed Data (Figure 1(c))** In the decentralized setting, the network of machines is represented as a graph $G$, and machines communicate ("gossip") with their neighbors. Many works (25, 15, 20) have considered the setting of *randomized* gossip, where each machine has an exponential clock and wakes up to communicate with its neighbors each time it ticks. In [20] a convergence rate of $O(1/\epsilon^2)$ is achieved for non-convex objectives $f$, matching the rate of SGD. We remark that by choosing the graph $G$ to be the complete graph, this result extends to our distributed data, centralized PS setting with random exponential work times. For strongly convex objectives $f$, [30] studied a decentralized setting with arbitrary but bounded delays, and achieved a linear rate of convergence using a gradient tracking technique.

We summarize the most relevant comparisons to our work in Table 1. We refer the reader to [3] for a recent survey on asynchronous parallel optimization algorithms for a more complete discussion of compute architectures and asynchronous algorithms such as coordinate decent methods that are beyond the scope of this section.

### 1.3 Organization

The rest of the paper is organized as follows. In Section 2 we formally set up the problem and our algorithm. In Section 3 we state and prove our main results. In Section 4 we provide empirical simulations. We conclude and discuss future directions in Section 5.

### 2 Formal Set-up

In this work, we study the following finite-sum minimization problem:

$$\min_{x \in \mathbb{R}^d} f(x) := \sum_{i=1}^{n} f_i(x).$$

\[\text{(5)}\]
We make the following assumptions, which are standard in the literature on synchronous finite sum minimization [10, 17, 12, 23]. The functions $f_i$ are convex and $L$-smooth, that is,

$$|\nabla f_i(x) - \nabla f_i(y)|^2 \leq L|x-y|^2 \quad \forall x, y, i.$$ 

Further the objective $f$ is $L_f$-smooth and $\mu$-strongly convex, that is,

$$(\nabla f(x) - \nabla f(y), x-y) \geq \mu|x-y|^2 \quad \forall x, y.$$ 

Note that because $f$ is an average of the $f_i$, we have $L_f \leq L$.

In ADSAGA which we analyze, we partition the $n$ functions equally among the $m$ machines into sets $\{S_j\}_{j \in [m]}$, such that each machine $j$ has access to $f_i$ for $i \in S_j$. Each machine maintains a local copy of the iterate $x$, which we denote $x_j$, and also stores a vector $\alpha_i$ for each $i \in S_j$, which contains the last gradient of $f_i$ computed at machine $j$ and sent to the PS. The central PS stores the current iterate $x$ and maintains the average $\overline{\alpha}$ of the $\alpha_i$.

Asynchronously and in parallel, each machine performs the following steps, formally described in Algorithm 1.

1. Sample a random function $i \in S_j$, and compute the gradient of $f_i$ at the local copy of the iterate, $x_j$. Set $h_j = \nabla f_i(x_j) - \alpha_i$, and update $\alpha_i = \nabla f_i(x_j)$.

2. Update $x_j$ to be the current iterate $x$ at the PS and send $h_j$ to the PS. These two operations are performed atomically; that is, we assume that no other machine communicates with the PS in between $x$ being read and $h_j$ being sent and used at the PS.\(^3\)

Meanwhile, each time the PS receives a gradient update $h_j$, it performs a SAGA iteration to update $x$:

$$x \leftarrow x - \eta(h_j + \overline{\alpha}),$$ \hspace{1cm} (6)

where $\eta$ is a step size. Next, to maintain the invariant $\overline{\alpha} = E_i[\alpha_i]$, the PS updates $\overline{\alpha} \leftarrow \overline{\alpha} + \frac{1}{n}h_j$.

We provide a logical view of ADSAGA in Algorithm 2. Crucially, because we assume exponential work times, we are able to view our algorithm logically as choosing a random function $i$ in each iteration, and performing the stale SAGA update using this function. To make notation clearer for the analysis, in Algorithm 2 we index the central parameter with a superscript of the iteration counter $k$. Note that in this case $\overline{\alpha}$ is an average of the $\alpha_i$.

\(^3\)This assumption of atomicity is reasonable in a parameter server setup, where each machine spends the bulk of its time in Step 1.

![Table 1: Comparison of Related Work. Note that we have substituted $O(m)$ for the maximum overlap bound $\tau$ in \[17, 34\], which is at least $m$.](image)
algorithm, we also introduce the auxiliary variables $\beta_j$ and $i_j$ to aid with the analysis. We emphasize that under our model of exponential work times, the logical view of Algorithm 2 is equivalent to the implementation in Algorithm 1.

| Algorithm 1 Asynchronous Distributed SAGA (ADSAGA): Implementation |
|-----------------------------------------------|
| **procedure** ADSAGA($x, \eta, \{f_i\}, t$) |
| $h_j = 0$ for $i \in [m]$ | $\triangleright$ Initialize updates to 0 at each machine |
| $\alpha_i = 0$ for $i \in [n]$ | $\triangleright$ Initialize last gradients to 0 at each machine |
| $\overline{\alpha} = 0$ | $\triangleright$ Initialize last gradient averages at PS |
| repeat in parallel at each machine $j$: |
| **Atomically:** |
| $x_j \leftarrow x$ | $\triangleright$ Read current iterate from PS |
| $x \leftarrow x - \eta(h_j + \overline{\alpha})$ | $\triangleright$ Apply stale SAGA update at PS |
| $\overline{\alpha} \leftarrow \overline{\alpha} + \frac{1}{n}h_j$ | $\triangleright$ Update gradient averages at PS |
| **Locally:** |
| $i_j \sim \text{Uniform}(S_j)$ | $\triangleright$ Choose a stochastic function at machine $j$ |
| $h_j \leftarrow \nabla f_i(x_j) - \alpha_i$ | $\triangleright$ Evaluate gradient of $f_i$ at iterate at machine $j$ |
| $\alpha_i \leftarrow \nabla f_i(x_j)$ | $\triangleright$ Update last gradient locally |
| until $t$ total updates have been made |
| return $x$ |
| end procedure |

| Algorithm 2 Asynchronous Distributed SAGA (ADSAGA): Logical View with Exponential Work Times |
|-----------------------------------------------|
| **procedure** ADSAGA($x^0, \eta, \{f_i\}, t$) |
| $g_j, h_j = 0$ for $i \in [m]$ | $\triangleright$ Initialize updates to 0 at each machine |
| $i_j \sim S_j$ for $j \in [m]$ | $\triangleright$ Randomly initialize last gradient indicator at each machine |
| $\alpha_i = 0$ for $i \in [n]$ | $\triangleright$ Initialize last gradients to 0 at each machine |
| $\overline{\alpha} = 0$ | $\triangleright$ Initialize last gradient averages at PS |
| for $k = 1$ to $t$ do |
| $i \sim \text{Uniform}(n); j \leftarrow j(i)$ | $\triangleright$ Randomly choose a function, this fixes a machine $j(i)$ to wake up |
| $x_j \leftarrow x^k$ | $\triangleright$ Read current iterate from PS |
| $x^{k+1} \leftarrow x^k - \eta(h_j + \overline{\alpha})$ | $\triangleright$ Apply stale SAGA update at PS |
| $\overline{\alpha} \leftarrow \overline{\alpha} + \frac{1}{n}h_j$ | $\triangleright$ Update gradient averages at PS |
| $\alpha_{ij} \leftarrow g_{ij}$ | $\triangleright$ Update last gradient locally |
| $g_j \leftarrow \nabla f_i(x_j)$ | $\triangleright$ Evaluate gradient of $f_i$ at iterate at machine $j$ |
| $\beta_j \leftarrow \alpha_i$ | $\triangleright$ Update auxiliary variable $\beta_j$ |
| $h_j \leftarrow g_j - \beta_j$ | $\triangleright$ Prepare next update to be sent to PS |
| $i_j \leftarrow i$ | $\triangleright$ Update auxiliary variable $i_j$ |
| end for |
| return $x^t$ |
| end procedure |

3 Convergence

In this section, we prove our main result, which yields a convergence rate of

$$\hat{O} \left( \left( n + \frac{L}{\mu} + \frac{\sqrt{m}L_f L}{\mu} \right) \log(1/\epsilon) \right).$$
Theorem 3.1. After

\[ k = \left( 4n + 158 \frac{L}{\mu} + 22 \sqrt{\frac{nL}{\mu}} \right) \log \left( \frac{1 + \frac{1}{2mL}}{\epsilon} \right) \]

iterations of Algorithm 2 with \( \eta = \frac{1}{59L + 8 \sqrt{\frac{mL}{\mu}}} \), we have

\[ \mathbb{E}[f(x^k) - f(x^*)] \leq \epsilon, \]

where \( \sigma^2 = \mathbb{E}_i [\|\nabla f_i(x^*)\|^2] \).

Let \( y^k \) denote the value \( x^k - x_* \), and let \( G^k \) be the matrix whose \( j \)th column contains the vector \( g_j \). Let \( H^k \) be the matrix whose \( j \)th column contains the vector \( h_j = g_j - \beta_j \) at the start of iteration \( k \). Let \( \alpha_i^k \) and \( \beta_i \) denote the values of \( \alpha_i \) and \( \beta_i \) respectively at the start of iteration \( k \), and likewise \( \bar{\alpha}_i^k := \mathbb{E}_i[\alpha_i^k] \). When the iteration \( k \) is clear from context, we will eliminate the superscripts \( k \). To further simplify, we will use the following definitions: \( \alpha_i^* := \alpha_i - \nabla f_i(x^*) \), \( \beta_j^* := \beta_j - \nabla f_j(x^*) \), and \( g_j^* := g_j - \nabla f_j(x^*) \).

We will analyze the expectation of the following potential function \( \phi(x, G, H, \alpha, \beta) \):

\[ \phi(x, G, H, \alpha, \beta) := \phi_2(x, G, H, \alpha, \beta) + \phi_3(x, G, H, \alpha, \beta), \]

where

\[ \phi_2(x, G, H, \alpha, \beta) := \frac{\epsilon}{2} \left( \frac{1}{\eta(H \mathbb{1} + m \bar{\alpha})} \right)^{\frac{1}{2}} \left( \frac{x - x^*}{\eta(H \mathbb{1} + m \bar{\alpha})} \right)^T \left( \begin{array}{cc} c_1 & c_2 \\ c_3 & c_4 \end{array} \right) \left( \frac{x - x^*}{\eta(H \mathbb{1} + m \bar{\alpha})} \right), \]

\[ \phi_3(x, G, H, \alpha, \beta) := \eta^2 c_4 \sum_j |g_j^*|^2, \]

\[ \phi_4(x, G, H, \alpha, \beta) := \eta^2 c_5 \sum_i |\alpha_i^*|^2 - \eta^2 c_5 \sum_j |\beta_j^*|^2. \]

Note that \( c_1, c_2, c_3 \) represent the \( d \times d \) matrices \( c_1 \mathbb{1}, c_2 \mathbb{1}, \) and \( c_3 \mathbb{1} \) respectively. Later, we will choose the \( m_i \) as follows:

\[
\begin{align*}
c_0 &= 2m\eta c_3 = 4m\eta, \\
c_1 &= 1, \\
c_2 &= -1, \\
c_3 &= 2, \\
c_4 &= \frac{4}{3} \left( \frac{11}{5} + \frac{m}{n} \right), \\
c_5 &= \frac{8}{3} \left( \frac{11}{5} + \frac{m}{n} \right) \frac{3nL_f \eta + 4 + \frac{4m}{n}}{\frac{2}{5} + \frac{m}{n}}, \\
c_6 &= \frac{3nL_f \eta + 4 + \frac{4m}{n}}{\frac{2}{5} + \frac{m}{n}}, \\
c_7 &= \frac{3nL_f \eta + 8 + \frac{8m}{n}}{\frac{2}{5} + \frac{m}{n}}.
\end{align*}
\]

It is easy to check that with these values, the potential function is non-negative. In the rest of the text, we abbreviate the potential \( \phi(x^k, H^k, \alpha^k, \beta^k) \) at the \( k \)th iteration by \( \phi(k) \).

This potential function captures not only progress in \( f(x^k) - f(x^*) \) and \( |x^k - x^*|^2 \), but also the extent to which the expected stale update, \( \frac{1}{n} H \mathbb{1} + \bar{\mathbb{1}} \), is oriented in the direction of \( x^k - x^* \). While some steps of asynchronous gradient descent may take us in expectation further from the optimum, those steps will position us for later progress by better orienting \( \frac{1}{n} H \mathbb{1} + \bar{\mathbb{1}} \). The values of \( m_i \) are chosen to cancel extraneous quantities that arise when evaluating the expected difference in potential between steps.

The following proposition is our main technical proposition.
Proposition 3.2. In Algorithm 2, for a step size \( \eta \leq \frac{1}{59L + 8\sqrt{mL/L}} \),

\[
\mathbb{E}[\phi(k + 1)] \leq \left(1 - \frac{\gamma}{n}\right) \phi(k)
\]

where \( \gamma = \min \left(\frac{1}{4}, \frac{3n\mu m}{8}\right) \).

Before we prove this proposition, we prove Theorem 3.1 which follows from Proposition 3.2. We restate the theorem for the reader’s convenience.

Theorem 3.3. After

\[
k = \left(4n + 158 \frac{L}{\mu} + 22 \sqrt{\frac{mL_L}{\mu}}\right) \log \left(\frac{\frac{1}{2} + \frac{1}{2\mu}}{\frac{1}{2} + \frac{1}{2mp}} \left(\frac{f(x^0) - f(x^*)}{\epsilon} + \frac{n\sigma^2}{4mL^2}\right)\right)
\]

iterations of Algorithm 2 with \( \eta = \frac{1}{59L + 8\sqrt{mL/L}} \), we have

\[
\mathbb{E}[f(x^k) - f(x^*)] \leq \epsilon,
\]

where \( \sigma^2 = \mathbb{E}_i [\|\nabla f_i(x^*)\|^2] \).

Proof. Upon initialization, the expected potential \( \phi(0) \) (over the random choices of \( i_j \)) equals

\[
2m(f(x^0) - f(x^*)) + |x^0 - x^*|^2 + \eta^2 \left(\frac{4}{3} + \frac{10}{3} \frac{11}{\mu} + \frac{m}{n} c_5 + n \frac{8}{5} c_5\right) \mathbb{E}_i [\|\nabla f_i(x^*)\|^2]
\]

\[
\leq \left(2m + \frac{2}{\mu}\right) (f(x^0) - f(x^*)) + \eta^2(6n)(14 + 10mL_{f\eta}) \mathbb{E}_i [\|\nabla f_i(x^*)\|^2].
\]

For \( \eta = \frac{1}{59L + 8\sqrt{mL/L}} \), we have

\[
\mathbb{E}[\phi(0)] \leq \left(2m + \frac{2}{\mu}\right) (f(x^0) - f(x^*)) + \left(\frac{n}{L^2}\right) \mathbb{E}_i [\|\nabla f_i(x^*)\|^2].
\]

With \( \gamma, \eta \) and \( k \) as in the corollary statement, we have

\[
(1 - \frac{\gamma}{n})^k \leq \exp \left(-\frac{\gamma k}{n}\right) \leq \exp \left(-\log \left(\frac{\frac{1}{2} + \frac{1}{2mp}}{\frac{1}{2} + \frac{1}{2mp}} \left(\frac{f(x^0) - f(x^*)}{\epsilon} + \frac{n\sigma^2}{4mL^2}\right)\right)\right)
\]

\[
\leq \frac{\epsilon}{\left(\frac{1}{2} + \frac{1}{2mp}\right) (f(x^0) - f(x^*)) + \frac{n\sigma^2}{4mL^2}} \leq \frac{4m\epsilon}{\mathbb{E}[\phi(0)]}
\]

It follows from Proposition 3.2 that

\[
\mathbb{E}[\phi(k)] \leq 4m\epsilon
\]

Since \( \phi(k) \geq 4m(f(x^k) - f(x_*)) \), we have \( \mathbb{E}[f(x^k) - f(x^*)] \leq \epsilon \) as desired.

\[ \square \]

Proof. (Proposition 3.2) To abbreviate, let \( M = \begin{pmatrix} c_1 & c_2 & c_3 \end{pmatrix} \). We also abbreviate \( \nabla_i := \nabla f_i(x^k) \), and let \( \nabla \) be the matrix whose \( ith \) column is \( \nabla_i \). All expectations are over the random choice of \( i \) in Algorithm 2 which induces a choice of \( j(i) \), the unique machine holding function \( i \). We will often abbreviate \( j(i) \) by \( j \).

Let

\[
\Delta_i := \begin{cases} 
\frac{g_j(i) - \beta_j(i) + \alpha_i}{n} & i \neq j(i); \\
\frac{g_j(i) - \beta_j(i) + \alpha_i}{n} & i = j(i).
\end{cases}
\]

\[
\Delta_i := \begin{cases} 
\frac{g_j(i) - \beta_j(i) + \alpha_i}{n} & i \neq j(i); \\
\frac{g_j(i) - \beta_j(i) + \alpha_i}{n} & i = j(i).
\end{cases}
\]
such that
\[ E[\phi_2(k + 1)] = E_i \left[ \left( \left( \eta (H \mathbb{1} + m \overline{\alpha}) \right) - \eta \Delta_i \right)^T c_1 c_2 c_3 \left( \left( \eta (H \mathbb{1} + m \overline{\alpha}) \right) - \eta \Delta_i \right) \right] \]

Observe that \( \beta_{j(i)} = \alpha_{j(i)} \), and that we maintain the invariant \( \overline{\alpha} = E_i [\alpha_i] \) at the end of every iteration. Hence
\[ \mathbb{E}_i [\Delta_i] = \frac{1}{m} \left( \frac{H \mathbb{1} + m \overline{\alpha}}{n} \right) \]  \hspace{1cm} (13)

Consider the difference
\[ \phi_2(k) - E[\phi_2(k + 1)] = 2 \eta \left( \eta (H \mathbb{1} + m \overline{\alpha}) \right)^T M \left( \eta (H \mathbb{1} + m \overline{\alpha}) - \frac{m}{n} \nabla \mathbb{1} \right) - \eta^2 \mathbb{E}_i [\Delta_i^T M \Delta_i] \]
\[ = \frac{2 \eta c_1 c_2}{n} (H \mathbb{1})^T \nabla \mathbb{1} - \frac{2 \eta^2 c_3}{n} (H \mathbb{1})^T \nabla \mathbb{1}. \]  \hspace{1cm} (14)

The following claims bound the differences in expectation of the other \( \phi_\ell \).

**Claim 3.4.**
\[ \phi_1(k) - E[\phi_1(k + 1)] \geq \frac{c_{0\eta}}{mn} (H \mathbb{1} + \overline{\alpha})^T \nabla \mathbb{1} \]
\[ - \frac{3 c_0 L_f \eta^2}{2} \left( E_i [\eta g_j^* | \Delta^2] + c_1 [\beta_j^* | \Delta^2] + E_i [\alpha_i^* | \Delta^2] \right) \]  \hspace{1cm} (15)

**Proof.** Using convexity and \( L_f \)-smoothness,
\[ \phi_1(k) - E[\phi_1(k + 1)] = c_0 f(x) - c_0 E_i [f(x - \eta (g_j - \beta_j + \overline{\alpha}))] \]
\[ \geq \frac{c_{0\eta}}{mn} (H \mathbb{1} + \overline{\alpha})^T \nabla \mathbb{1} - \frac{c_0 L_f \eta^2}{2} E_i [g_j - \beta_j + \overline{\alpha} | \Delta^2]. \]  \hspace{1cm} (16)

Now by Jenson’s inequality,
\[ E_i [g_j - \beta_j + \overline{\alpha} | \Delta^2] \leq 3 E_i [g_j^* | \Delta^2] + 3 E_i [\beta_j^* | \Delta^2] + 3 |\overline{\alpha}|^2 \]
\[ \leq 3 E_i [g_j^* | \Delta^2] + 3 E_i [\beta_j^* | \Delta^2] + 3 E_i [\alpha_i^* | \Delta^2]. \]  \hspace{1cm} (17)

**Lemma 3.5.**
\[ E_i [\nabla_i - \nabla_i(x^*) | \Delta^2] \leq L_y^T \nabla \mathbb{1}. \]  \hspace{1cm} (18)

**Proof.** By the convexity of each \( f_i \) and their \( L \)-smoothness,
\[ E_i [\nabla_i - \nabla_i(x^*) | \Delta^2] \leq E_i [L_y^T (\nabla_i - \nabla_i(x^*))] = \frac{L_y^T \nabla \mathbb{1}}{n}. \]  \hspace{1cm} (19)

**Claim 3.6.**
\[ \phi_3(k) - E[\phi_3(k + 1)] \geq \eta^2 c_4 E_i [g_j^* | \Delta^2] - \frac{\eta^2 c_4 L_y^T \nabla \mathbb{1}}{n}. \]  \hspace{1cm} (20)
Proof.

\[
\phi_3(k) - E[\phi_3(k + 1)] = \eta^2 c_4 \left( E_i \left[ |g_j - \nabla_i (x^*)|_2^2 \right] - E_i \left[ |\nabla_i - \nabla_i(x^*)|_2^2 \right] \right). 
\] (21)

Plugging in Lemma 3.5 yields the claim. □

Claim 3.7.

\[
\phi_4(k) - E_i[\phi_4(k + 1)] = \frac{1}{4n} \phi_4(k) + \eta^2 c_5 \left( \frac{3}{5} - \frac{m}{4n} \right) E_i[|\alpha^*_i|_2^2] \\
+ \eta^2 c_5 \left( \frac{3}{5} - \frac{m}{4n} \right) E_j \left[ |\beta^*_j|_2^2 \right] \\
- \frac{8\eta^2 c_5}{5} E_j \left[ |g^*_j|_2^2 \right].
\] (22)

Proof. If gradient \( i \) is chosen in the \( k + 1 \)th iteration, then

\[
\phi_4(k + 1) - \phi_4(k) = \eta^2 c_5(1 + c_6) \left( |g^*_j|_2^2 - |\alpha^*_{i_{j(i)}}|_2^2 \right) \\
+ \eta^2 c_5 \left( |\beta^*_{j(i)}|_2^2 - |\alpha^*_{i_{j(i)}}|_2^2 + I(i = i_{j(i)})(|\alpha^*_{i_{j(i)}}|_2^2 - |g^*_{j(i)}|_2^2) \right) \\
= -\eta^2 c_5 \left( |\alpha^*_{i_{j(i)}}|_2^2 + c_6 |\beta^*_{j(i)}|_2^2 \right) + \eta^2 c_5(1 + c_6) |g^*_{j(i)}|_2^2 \\
+ \eta^2 c_5 I(i = i_{j(i)})(|\beta^*_{j(i)}|_2^2 - |g^*_{j(i)}|_2^2). 
\] (23)

Taking the expectation over \( i \) yields

\[
E_i[\phi_4(k + 1)] - \phi_4(k) = -\eta^2 c_5 \left( E_i \left[ |\alpha^*_{i_{j(i)}}|_2^2 \right] + c_6 E_j \left[ |\beta^*_{j(i)}|_2^2 \right] \right) \\
+ \eta^2 c_5(1 + c_6) E_j |g^*_{j(i)}|_2^2 \\
+ \eta^2 c_5 \frac{m}{n} \left( E_j \left[ |\beta^*_{j(i)}|_2^2 \right] - E_j \left[ |g^*_{j(i)}|_2^2 \right] \right) \\
= -\frac{1}{4n} \phi_4(k) - \eta^2 c_5 \left( 1 - \frac{c_6 + 1}{4} \right) E_i[|\alpha^*_{i_{j(i)}}|_2^2] \\
- \eta^2 c_5 \left( c_6 - \frac{5m}{4n} \right) E_j \left[ |\beta^*_{j(i)}|_2^2 \right] \\
+ \eta^2 c_5 \left( 1 + c_6 - \frac{m}{n} \right) E_j \left[ |g^*_{j(i)}|_2^2 \right]. \] (24)

Plugging in \( c_6 = \frac{3}{5} + \frac{m}{n} \) yields the claim. □

Finally, the next claim bounds the quadratic term in the difference \( \phi_2(k) - E[\phi_2(k)] \) in Equation (14).

Claim 3.8.

\[
E_i \left[ \Delta_i^T M \Delta_i \right] \leq 4 \left( c_1 + c_2 + \left( 1 + \frac{m}{n} \right) c_3 \right) \left( E_i \left[ |g^*_j|_2^2 \right] + E_i \left[ |\beta^*_j|_2^2 \right] \right) + 4c_3 E \left[ |\alpha^*_i|_2^2 \right] + \frac{4c_3L}{n} y^T \nabla 1. \] (25)

10
Proof.

\[ \mathbb{E}_i [\Delta_i^T M \Delta_i] = \mathbb{E}_i \left[ \left( (I(i = i_j) + 1 - \frac{m}{n}) (g_j - b_j) - \nabla_i + \alpha_i \right)^T M \left( (I(i = i_j) + 1 - \frac{m}{n}) (g_j - b_j) - \nabla_i + \alpha_i \right) \right] \]

\[
\leq 4 \left( 1 - \frac{m}{n} \right) \mathbb{E}_i \left[ \left( \frac{g_j}{n} \right)^T M \left( \frac{g_j}{n} \right) + \frac{m}{n} \mathbb{E}_i \left[ \left( \frac{g_j}{n} \right)^T M \left( \frac{g_j}{n} \right) \right] + 4 \mathbb{E}_i \left[ \left( \frac{\alpha_i - \nabla_i(x^*)}{\alpha_i - \nabla_i(x^*)} \right)^T M \left( \frac{\alpha_i - \nabla_i(x^*)}{\alpha_i - \nabla_i(x^*)} \right) \right] \right] 
\]

\[
\leq 4 \left( c_1 + c_2 + \left( 1 + \frac{m}{n} - \frac{m^2}{n^2} \right) c_3 \right) \left( \mathbb{E}_i [g_j^2] + \mathbb{E}_i [\beta^2_j] \right) + 4c_3 \mathbb{E}_i [\alpha_i^2] + 4c_3 \mathbb{E}_i [\nabla_i \beta_j^2] 
\]

where the first inequality is by Jensen’s inequality. Plugging in Lemma 3.5 yields the claim. \( \square \)

We now combine Equation (14), Claim 3.6, Claim 3.7, Claim 3.4 and Claim 3.8 to find the total expected difference in potential. Note that we plug in our choice of \( c_0 = 2m\eta c_3 \), which was chosen so that the \((H1 + m\alpha)^T \nabla 1\) terms cancel.

\[
\phi(k) - \mathbb{E}[\phi(k + 1)] \geq \frac{2}{m} \left( \eta(H1 + m\alpha) \right)^T \left( \frac{0}{c_1 + c_2} \frac{c_1 + c_2}{2} \frac{c_1 + c_2}{c_2 + c_3} \right) \left( \eta(H1 + m\alpha) \right) - \left( 2\eta c_2 + 4\eta^2 c_3 L + \eta^2 c_4 L \right) \left( \frac{y^T \nabla 1}{n} \right) 
\]

\[
+ \mathbb{E}_i [g_j^2] \eta^2 \left( c_4 - 3mL \eta c_3 - 4 \left( c_1 + c_2 + \left( 1 + \frac{m}{n} \right) c_3 \right) - c_5 \frac{8}{5} \right) 
\]

\[
+ \mathbb{E}_i [\beta^2_j] \eta^2 \left( c_5 \left( \frac{3}{5} - \frac{m}{4n} \right) - 3mL \eta c_3 - 4 \left( c_1 + c_2 + \left( 1 + \frac{m}{n} \right) c_3 \right) \right) 
\]

\[
+ \mathbb{E}_i [\alpha_i^2] \eta^2 \left( c_5 \left( \frac{3}{5} - \frac{m}{4n} \right) - 3mL \eta c_3 - 4c_3 \right) 
\]

\[
+ \frac{1}{4n} \phi_4(k). 
\]

To further cancel terms, we plug in

\[ c_2 = -c_1, \]

and

\[ c_5 = c_3 \left( \frac{3mL \eta + 4 + \frac{4m}{n}}{3 \frac{5}{5} + \frac{m}{n}} \right) \]

such that we obtain
\[
\phi(k) - \mathbb{E}[\phi(k + 1)] \geq \frac{2}{m} \left( \eta(H\mathbb{1} + m\mathbb{1}) \right)^T \begin{pmatrix} y & 0 & 0 \\ 0 & (c_2 + c_3) & 0 \end{pmatrix} \left( \eta(H\mathbb{1} + m\mathbb{1}) \right) \\
- \left( \frac{2\eta c_2 + 4\eta^2 c_3 L + \eta^2 c_4 L}{n} \right) (y^T \nabla 1) \\
+ \mathbb{E}_i ||g^{*}_j||_2^2 \eta^2 \left( c_4 - c_5 \left( \frac{11}{5} + \frac{m}{n} \right) \right) \\
+ \frac{1}{4n} \phi_4(k).
\]

(28)

Now because \( f \) is convex, \( f(x) - f(x^*) \leq \frac{y^T \nabla 1}{n} \), and so rearranging terms, we have
\[
\phi(k) - \mathbb{E}[\phi(k + 1)] \geq \frac{2}{m} \left( \eta(H\mathbb{1} + m\mathbb{1}) \right)^T \begin{pmatrix} y & 0 & 0 \\ 0 & (c_2 + c_3) & 0 \end{pmatrix} \left( \eta(H\mathbb{1} + m\mathbb{1}) \right) \\
- \left( \frac{2\eta c_2 + 4\eta^2 c_3 L + \eta^2 c_4 L}{n} \right) (y^T \nabla 1) \\
+ \mathbb{E}_i ||g^{*}_j||_2^2 \eta^2 \left( c_4 - c_5 \left( \frac{11}{5} + \frac{m}{n} \right) \right) \\
+ \frac{1}{4n} \phi_4(k) \\
+ \frac{c_0}{4n} (f(x) - f(x^*)).
\]

(29)

Further, using the strong convexity of \( f \), we have \( y^T \nabla 1 \geq n\mu ||y||^2_2 \), and so assuming that (which is proved in Equation (34) below)
\[
- \left( \frac{2\eta c_2 + 4\eta^2 c_3 L + \eta^2 c_4 L + \eta \frac{m c_3}{2n}}{n} \right) > 0,
\]

we have
\[
\phi(k) - \mathbb{E}[\phi(k + 1)] \geq \frac{1}{m} \left( \eta(H\mathbb{1} + m\mathbb{1}) \right)^T \begin{pmatrix} y & 0 & 0 \\ 0 & (c_2 + c_3) & 0 \end{pmatrix} \left( \eta(H\mathbb{1} + m\mathbb{1}) \right) \\
- \left( \frac{2\eta c_2 + 4\eta^2 c_3 L + \eta^2 c_4 L + \eta \frac{m c_3}{2n}}{n} \right) \mu \left( \frac{2(c_2 + c_3)}{2(c_2 + c_3)} \right) \left( \eta(H\mathbb{1} + m\mathbb{1}) \right) \\
+ \mathbb{E}_i ||g^{*}_j||_2^2 \eta^2 \left( c_4 - c_5 \left( \frac{11}{5} + \frac{m}{n} \right) \right) \\
+ \frac{1}{4n} \phi_4(k) \\
+ \frac{c_0}{4n} (f(x) - f(x^*)).
\]

(30)

(31)

Recall that our goal is to find some \( \gamma \leq \frac{1}{4} \) such that
\[
\mathbb{E}[\phi(k + 1)] \leq \left( 1 - \frac{\gamma}{n} \right) \phi(k).
\]

We will do this by finding some \( \gamma \) that satisfies for all \( y, G, H, \alpha, \) and \( \beta \):

1. \[
\frac{\eta^2 \left( c_4 - c_5 \left( \frac{11}{5} + \frac{m}{n} \right) \right) \mathbb{E}_i ||g^{*}_j||^2_2}{\eta^2 c_4 \sum_j ||g^{*}_j||^2_2} \geq \frac{\gamma}{n}.
\]
By plugging in
\[ c_4 = \frac{4}{3} c_5 \left( \frac{11}{5} + \frac{m}{n} \right) = \frac{8}{3} \left( \frac{11}{5} + \frac{m}{n} \right) \frac{3nL_f \eta + 4 + \frac{4m}{n}}{3 + \frac{m}{n}}, \]
the first statement holds for \( \gamma \leq \frac{1}{4} \). For the second statement, we must have
\[
\begin{pmatrix}
\frac{y}{\eta(H_1 + m\bar{\alpha})} & 0 \\
0 & \frac{0}{\eta(H_1 + m\bar{\alpha})}
\end{pmatrix}
\begin{pmatrix}
y \\
0
\end{pmatrix}
\begin{pmatrix}
\frac{c_1}{c_2} & \frac{c_2}{c_3} & \frac{c_1}{c_3}
\end{pmatrix}
\begin{pmatrix}
\frac{y}{\eta(H_1 + m\bar{\alpha})} & 0 \\
0 & \frac{0}{\eta(H_1 + m\bar{\alpha})}
\end{pmatrix}
\geq \frac{\gamma}{2} \begin{pmatrix}
y \\
0
\end{pmatrix}
\begin{pmatrix}
\frac{c_1}{c_2} & \frac{c_2}{c_3} & \frac{c_1}{c_3}
\end{pmatrix}
\begin{pmatrix}
y \\
0
\end{pmatrix}
\begin{pmatrix}
\frac{c_1}{c_2} & \frac{c_2}{c_3} & \frac{c_1}{c_3}
\end{pmatrix}^{-1/2}
\begin{pmatrix}
y \\
0
\end{pmatrix}
\begin{pmatrix}
\frac{c_1}{c_2} & \frac{c_2}{c_3} & \frac{c_1}{c_3}
\end{pmatrix} \geq \gamma I \quad (32)
\]
Equivalently, we must have
\[
\begin{pmatrix}
\frac{c_1}{c_2} & \frac{c_2}{c_3} & \frac{c_1}{c_3}
\end{pmatrix}^{-1/2}
\begin{pmatrix}
y \\
0
\end{pmatrix}
\begin{pmatrix}
\frac{c_1}{c_2} & \frac{c_2}{c_3} & \frac{c_1}{c_3}
\end{pmatrix}
\begin{pmatrix}
y \\
0
\end{pmatrix}
\begin{pmatrix}
\frac{c_1}{c_2} & \frac{c_2}{c_3} & \frac{c_1}{c_3}
\end{pmatrix} \geq \gamma I \quad (32)
\]
We can check the smallest eigenvalue of this matrix product by evaluating the trace and determinant of the 2 × 2 matrices that are the basis for the block matrices above.

- **Determinant:**
  \[ D := \frac{2n^2 \eta \mu (c_4 - c_1)(2c_1 - \frac{mc_3}{2n} - 4\eta c_3 L - \eta c_4 L)}{c_1(c_3 - c_1)}. \]

- **Trace:**
  \[ T := \frac{\eta \mu c_3(2c_1 - \frac{mc_3}{2n} - 4\eta c_3 L - \eta c_4 L) + 2n c_1(c_4 - c_1)}{c_1(c_3 - c_1)} = \frac{m}{n} D + \frac{2n}{m}. \]

Now since \( c_3 + c_2 > 0 \) and by the result of Equation (34), the trace and determinant are positive, and hence the smallest eigenvalue of the matrix product in Equation (32) is at least the quotient of the determinant and the trace. This quotient equals
\[
\frac{D}{\frac{2n}{m}} \geq \min \left( \frac{1}{4}, 3Dm \right) \quad (33)
\]
It is easy to check that \( c_4 < \frac{32}{3} (3mL_f \eta + 4) \), so for \( \eta \leq \frac{1}{59L + 8\sqrt{mL_f L}} \), we have
\[
D = \frac{n^2 \eta \mu}{m} \left( 4 - 2\frac{m}{n} - 16\eta L - 2\eta L \left( 32mL_f \eta + \frac{128}{3} \right) \right) \geq \frac{n^2 \eta \mu (2 - 59L \eta - 64mL_f L \eta^2)}{m} > \frac{n^2 \eta \mu}{m}. \quad (34)
\]
Plugging this into Equation (33) and recalling our bound that \( \gamma \leq \frac{1}{4} \), we can choose
\[ \gamma = \min \left( \frac{1}{4}, \frac{3\eta \mu}{8} \right). \]
\[\square\]
4 Experiments

Our experiments empirically validate our theoretical results and give empirical evidence for understanding Question 1. We study ADSAGA for the least squares problems, and compare the convergence rate as we vary the number of machines and the delay distribution. Further, we compare the performance of ADSAGA to other state-of-the-art parallelizable algorithms: minibatch synchronous SAGA, ASAGA [17], distributed SGD [20], and distributed incremental aggregate gradient (IAG) [14]. We show that with up to \( m = 60 \) machines, with the work time distributions we study, all algorithms perform similarly besides SGD, which is unable to converge quickly due to the variance of the gradient updates. As the number of machines grows very large, we see that ADSAGA performs better than IAG. However, if the work times are close to constant, ADSAGA is slower than ASAGA and minibatch SAGA.

More precisely, inspired by modeling of work times in [18], we study work time distributions \( w \sim s + \text{Exp}(1) \) for \( s \in \{0, 0.1, 1, 10\} \). When \( s = 0 \) this corresponds to exponential work times, which we have analyzed theoretically in this paper. As \( s \) grows large, the work times and the delays become close to constant.

We simulate ADSAGA, minibatch SAGA, ASAGA, SGD, and IAG on a least squares problem

\[
\min_x |A\hat{x} - b|^2,
\]

where \( A \in \mathbb{R}^{N \times d} \) is chosen randomly with i.i.d. rows from \( \mathcal{N}(0, \frac{1}{d}I_d) \), and \( x \sim \mathcal{N}(0, I_d) \). The observations \( b \) are noisy observations of the form \( b = Ax + Z \), where \( Z \sim \mathcal{N}(0, I_N) \). We use \( N = n = 120 \), and run ADSAGA, minibatch SAGA, ASAGA, SGD, and IAG on \( \hat{x} \) with \( m \) machines for \( m \in \{10, 20, 40, 60, 120\} \).

To be fair to all algorithms, for all experiments below, we use a grid search to find the best step size in \( \{0.05 \times i \}_{i \in [40]} \), ensuring that none of the best step sizes were at the boundary of this set.

We plot the expected iteration complexity to achieve \( |\hat{x} - x^*|^2 \leq \epsilon \) in Figure 1(a) for exponential work times, and in Figure 1(b) for shifted exponential work times where are close to constant. Here \( x^* := \min_{\hat{x}} |A\hat{x} - b|^2 \) is the empirical risk minimizer.

Further, we plot in Figure 2 the iteration complexity of ADSAGA, minibatch SAGA, ASAGA for varying shifts of the exponential distributions. For large shifts with many machines, ASAGA outperforms ADSAGA.
5 Conclusion

In this paper, we analyze ADSAGA, a SAGA-like algorithm in a distributed asynchronous setting. We showed that when the work times are distributed exponentially, we achieve convergence in $\tilde{O} \left( \left( n + \frac{L}{\mu} + \sqrt{mL/f} \right) \log(1/\epsilon) \right)$ iterations. This work leaves open several interesting questions:

1. If we assume i.i.d. work times which induce bounded delays, can we show a convergence rate of $O \left( \left( n + \frac{mL}{\mu} \right) \log(1/\epsilon) \right)$ for ADSAGA, equal to that of ASAGA？ Under what assumption on the work times can we achieve rates that scale sublinearly in the number of machines $m$, akin to with exponential work times, such that increasing the number of machines is always advantageous in terms of wall clock time?

2. For arbitrary but bounded delays in the distributed setting (studied in [14, 4, 31]), is the dependence on $n^2$ optimal? When $m < n$ and each machine holds many functions, if we use a stochastic gradient at each machine, can we achieve convergence rates better than $O \left( \frac{n^2 L}{\mu} \log(1/\epsilon) \right)$?

3. Can the rate of ASAGA with delays bounded by $O(m)$ (either in the original, randomly sampled data setting, or in a distributed setting with identically distributed delays) be improved to $O \left( n + \frac{L}{\mu} + \frac{mL}{\mu} \right)$, such that the dependence on $m$ scales with $L_f$ instead of $L$? This would show that asynchronous SAGA performs as well its parallel, synchronous counterpart. There is reason to think this is possible due to asynchronous convergence rates for quadratic objectives which have terms that scale with $mL_f$ [2, 29]. Our empirical results also suggest that ASAGA performs as well as minibatch SAGA, which achieves this rate [24].

4. In the decentralized random gossip setting (which implies independent exponential work times), what rates does a SAGA-like algorithm achieve?

Any work time distribution on $[1, \tau/m - 1]$ induces delays bounded by $\tau$. 

Figure 2: Comparison of ADSAGA, ASAGA for varying shifts of the exponential distribution.
Acknowledgements

The authors thank Ioannis Mitliagkas, Kevin Tian, and Aaron Sidford for helpful conversations and pointers to the literature.

References

[1] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. The Journal of Machine Learning Research, 18(1):8194–8244, 2017.

[2] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In Algorithmic Learning Theory, pages 111–132, 2020.

[3] Mahmoud Assran, Arda Aytekin, Hamid Feyzmahdavian, Mikael Johansson, and Michael Rabbat. Advances in asynchronous parallel and distributed optimization. arXiv preprint arXiv:2006.13838, 2020.

[4] Arda Aytekin, Hamid Reza Feyzmahdavian, and Mikael Johansson. Analysis and implementation of an asynchronous optimization algorithm for the parameter server. arXiv preprint arXiv:1610.05507, 2016.

[5] Dimitri P Bertsekas and John N Tsitsiklis. Parallel and distributed computation: numerical methods, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

[6] Adel Bibi, Alibek Sailanbayev, Bernard Ghanem, Robert Mansel Gower, and Peter Richtárik. Improving saga via a probabilistic interpolation with gradient descent. arXiv preprint arXiv:1806.05633, 2018.

[7] Sorathan Chaturapruek, John C Duchi, and Christopher 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.

[8] Jeffrey Dean and Luiz André Barroso. The tail at scale. Communications of the ACM, 56(2):74–80, 2013.

[9] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc’aulério Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In Advances in neural information processing systems, pages 1223–1231, 2012.

[10] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In Advances in neural information processing systems, pages 1646–1654, 2014.

[11] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. The Journal of Machine Learning Research, 13:165–202, 2012.

[12] Nidham Gazagnadou, Robert Gower, and Joseph Salmon. Optimal mini-batch and step sizes for saga. In International Conference on Machine Learning, pages 2142–2150, 2019.

[13] Robert M Gower, Peter Richtárik, and Francis Bach. Stochastic quasi-gradient methods: Variance reduction via jacobian sketching. arXiv preprint arxiv:1805.02632, 2018.

[14] Mert Gurbuzbalaban, Asuman Ozdaglar, and Pablo A Parrilo. On the convergence rate of incremental aggregated gradient algorithms. SIAM Journal on Optimization, 27(2):1035–1048, 2017.

[15] Peter H Jin, Qiaochu Yuan, Forrest Iandola, and Kurt Keutzer. How to scale distributed deep learning? arXiv preprint arXiv:1611.04581, 2016.

[16] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In Advances in neural information processing systems, pages 315–323, 2013.
[17] Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. Improved asynchronous parallel optimization analysis for stochastic incremental methods. *The Journal of Machine Learning Research*, 19(1):3140–3207, 2018.

[18] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.

[19] Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, page 2, 2013.

[20] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052, 2018.

[21] Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *arXiv preprint arXiv:1507.06970*, 2015.

[22] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 997–1004. IEEE, 2016.

[23] Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in neural information processing systems*, pages 1017–1025, 2014.

[24] Atsushi Nitanda, Tomoya Murata, and Taiji Suzuki. Sharp characterization of optimal minibatch size for stochastic finite sum convex optimization. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 488–497. IEEE, 2019.

[25] S Sundhar Ram, Angelia Nedić, and Venu V Veeravalli. Asynchronous gossip algorithm for stochastic optimization: Constant stepsize analysis. In *Recent Advances in Optimization and its Applications in Engineering*, pages 51–60. Springer, 2010.

[26] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[27] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alexander J Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in neural information processing systems*, pages 2647–2655, 2015.

[28] Nicolas L Roux, Mark Schmidt, and Francis 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.

[29] Atal Narayan Sahu, Aritra Dutta, Aashutosh Tiwari, and Peter Richtárik. On the convergence analysis of asynchronous sgd for solving consistent linear systems. *arXiv preprint arXiv:2004.02163*, 2020.

[30] Ye Tian, Ying Sun, and Gesualdo Scutari. Achieving linear convergence in distributed asynchronous multi-agent optimization. *IEEE Transactions on Automatic Control*, 2020.

[31] N Denizcan Vanli, Mert Gurbuzbalaban, and Asuman Ozdaglar. Global convergence rate of proximal incremental aggregated gradient methods. *SIAM Journal on Optimization*, 28(2):1282–1300, 2018.

[32] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Osdi*, volume 8, page 7, 2008.

[33] Shen-Yi Zhao and Wu-Jun Li. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *AAAI*, pages 2379–2385, 2016.
[34] Kaiwen Zhou, Fanhua Shang, and James Cheng. A simple stochastic variance reduced algorithm with fast convergence rates. In *International Conference on Machine Learning*, pages 5980–5989, 2018.

[35] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.