Asynchronous Distributed Optimization with Stochastic Delays

Margalit Glasgow
Stanford University

Mary Wootters
Stanford University

Abstract

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 develop an algorithm ADSAGA based on SAGA for the distributed-data setting, in which the data is partitioned between many machines. We show that with m machines, under a natural stochastic delay model with an mean delay of m, ADSAGA converges in $\tilde{O}\left((n+\sqrt{m}\kappa)\log(1/\epsilon)\right)$ iterations, where n is the number of component functions, and κ is a condition number. This complexity sits squarely between the complexity $\tilde{O}\left((n+\kappa)\log(1/\epsilon)\right)$ of SAGA *without delays* and the complexity $\tilde{O}\left((n+m\kappa)\log(1/\epsilon)\right)$ of parallel asynchronous algorithms where the delays are *arbitrary* (but bounded by $O(m)$), and the data is accessible by all. Existing asynchronous algorithms with distributed-data setting and arbitrary delays have only been shown to converge in $\tilde{O}(n^2\kappa\log(1/\epsilon))$ iterations. We empirically compare the iteration complexity and wallclock performance of ADSAGA to existing parallel and distributed algorithms, including synchronous minibatch algorithms. Our results demonstrate the wallclock advantage of variance-reduced asynchronous approaches over SGD or synchronous approaches.

1 INTRODUCTION

In large scale machine learning problems, distributed training has become increasingly important. In this work, we consider a distributed setting governed by a central parameter server (PS), where the training data is partitioned among a set of machines, such that each machine can only access the data it stores locally. This is common in federated learning, where the machines may be personal devices or belong to different organizations (McMahan et al., 2017). Data-partitioning can also be used in data-centers to minimize stalls from loading data from remote file systems (Mohan et al., 2020).

Asynchronous algorithms — in which the machines do not serialize after sending updates to the PS — are an important tool in distributed training. Asynchrony can mitigate the challenge of having to wait for the slowest machine, which is especially important when compute resources are heterogeneous (Li et al., 2018). Perhaps surprisingly, there has been relatively little theoretical study of asynchronous algorithms in a distributed-data setting; most works have studied a shared-data setting where all of the data is available to all of the machines.

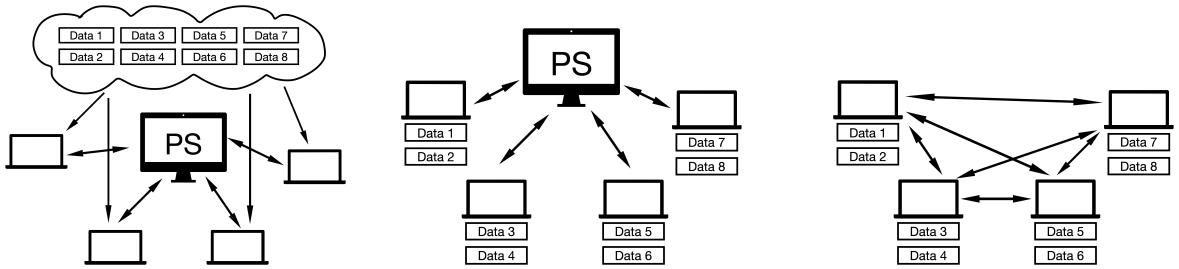
In this paper, we focus on asynchronous algorithms for the distributed-data setting, under a stochastic delay model. We consider the finite sum minimization problem common in many empirical risk minimization (ERM) problems:

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

where each f_i is convex and L-smooth, and f is L_f-smooth and μ-strongly convex. In machine learning, each f_i represents a loss function evaluated at a data point. A typical strategy for minimizing finite sums is using *variance-reduced* stochastic gradient algorithms, such as SAG (Roux et al., 2012), SVRG (Johnson and Zhang, 2013) or SAGA (Defazio et al., 2014). To converge to an ϵ-approximate minimizer x, (that is, some x such that $f(x) - \min_{x'} f(x') \leq \epsilon$), variance-reduced algorithms require $\tilde{O}((n + L/\mu)\log(1/\epsilon))$ iterations. In contrast, the standard stochastic gradient descent (SGD) algorithm yields a slower convergence rate that scales with $1/\epsilon$.

Many of these algorithms can be distributed across m machines who compute gradients updates *asyn-*

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.



(a) Shared Data, Central PS (b) Distributed Data, Central PS (c) Distributed Data, Decentralized

Figure 1: Settings for parallel optimization. (a) Shared-data setting, where ASAGA pertains (Leblond et al., 2018). (b) Distributed-data setting, where our work (ADSAGA) pertains. (c) Decentralized setting, in which there are a variety of algorithms with weaker guarantees.

chronously. In such implementations, a gradient step is performed on the k'th central iterate 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. That is, at iteration k, the PS performs the update

$$x^{k+1} = x^k - \eta U(i_k, x^{k-\tau(k)}),$$

where η 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 the SAGA algorithm — which forms the backbone of the algorithm we propose and analyze in this paper — the update used is

$$U(i,x) := \nabla f_i(x) - \alpha_i + \overline{\alpha}, \tag{2}$$

where α_i is the prior gradient computed of f_i, and $\overline{\alpha}$ denotes the average $\frac{1}{n}\sum_i \alpha_i$.[1]

There has been a great deal of work analyzing asynchronous algorithms in settings where the data is *shared* (or "i.i.d."), where any machine can access any of the data at any time, as in Figure 1(a). In particular, the asynchronous implementation of SAGA with shared data, called ASAGA (Leblond et al., 2018) is shown to converge in $\tilde{O}((n + \tau_{max}L/\mu)\log(1/\epsilon))$ iterations, under arbitrary delays that are bounded by τ_{max}. Other variance-reduced algorithms obtain similar results (Mania et al., 2015; Zhao and Li, 2016; Reddi et al., 2015; Zhou et al., 2018).

A key point in the analysis of asynchronous algorithms in the shared-data setting is the independence of the delay $\tau(k)$ at step k and the function f_{i_k} that is chosen at time k. This leads to an *unbiased gradient condition*, namely that the expected update is proportional to $\nabla f(x^{k-\tau(k)})$. This condition is central to the analyses of these algorithms. However, in the *distributed-data* (or "non-i.i.d.") setting, where each machine only has access to the partition of data is stores locally (Figure 1(b)), this condition does not naturally hold. For example, if the only assumption on the delays is that they are bounded, then using the standard SGD update $U(i,x) = \nabla f_i(x)$ may not even yield asymptotic convergence to x^*.

Due to this difficulty, the asynchronous landscape is far less understood when the data is distributed. Several works (Gurbuzbalaban et al., 2017; Vanli et al., 2018; Aytekin et al., 2016) analyze an asynchronous incremental aggregation gradient (IAG) algorithm, which can be applied to the distributed setting; those works prove that with arbitrary delays, IAG converges deterministically in $\tilde{O}(\frac{n^2L}{\mu}\log(1/\epsilon))$ iterations, significantly slower than the results available for the shared data model. The work of Xie et al. (2019) studies an asynchronous setting with distributed data and arbitrary delays and achieves an iteration complexity scaling polynomially with $1/\epsilon$. A line of work that considers a completely decentralized architecture without a PS (see Figure 1(c)) generalizes the distributed data PS setting of Figure 1(b). In this regime, with *stochastic* delays, Lian et al. (2018) established sublinear convergence rates of $\tilde{O}(L^2/\epsilon^2)$ using an SGD update; however variance-reduced algorithms, which could yield linear convergence rates (scaling with $\log(1/\epsilon)$) for finite sums, have not been studied in this setting.With arbitrary but bounded delays and strongly convex objectives, Tian et al. (2020) and Niwa et al. (2021) achieved a linear rate of convergence gradient tracking techniques; however their results depend exponentially on m.

We introduce the *ADSAGA algorithm*, a variant of SAGA designed for the distributed data setting (Figure 1(b)). For our analysis, we adopt the stochastic delay model from Lian et al. (2018) from the decen-

[1] This update is variance-reduced because it is an unbiased estimator of the gradient $\nabla f(x)$, and unlike the SGD update, its variance tends to 0 as x approaches the optimum of the objective (1).

tralized setting (Figure 1(c)), formalized below; this model is well-motivated (see Section 1.1), and allows us to prove strong convergence results despite the lack of independence between the data and the delays.

In this model, we show that ADSAGA converges in $\tilde{O}\left((n + L/\mu + \sqrt{mL_f L}/\mu)\log(1/\epsilon)\right)$ iterations. To the best of our knowledge, this is the first provable result for asynchronous algorithms in the distributed-data setting — under any delay model — that scales both logarithmically in $1/\epsilon$ and linearly in n. Moreover, our empirical results suggest that ADSAGA outperforms other distributed-data algorithms even when the delay distribution differs from our model.

1.1 Our Model and Assumptions

Assumptions on the functions f_i. We study the finite-sum minimization problem (1). As is standard in the literature on synchronous finite sum minimization (e.g. (Defazio et al., 2014; Leblond et al., 2018; Gazagnadou et al., 2019; Needell et al., 2014)), we assume that the functions f_i are convex and L-smooth:

$$|\nabla f_i(x) - \nabla f_i(y)|_2 \leq L|x - y|_2 \qquad \forall x, y, i,$$

and we similarly assume that the objective f is L_f-smooth. We further assume that f is μ-strongly convex:

$$\langle \nabla f(x) - \nabla f(y), x - y\rangle \geq \mu|x - y|_2^2 \qquad \forall x, y.$$

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

Distribution of the data. We assume the distributed-data model in Figure 1(b). The functions $f_1, \ldots, f_n$ are partitioned among the m machines into sets $\{S_j\}_{j\in[m]}$, such that each machine j has access to f_i for $i \in S_j$.[2]

Communication and delay model. The m machines are governed by a centralized PS. At timestep k, the PS holds an iterate x^k. We consider the following model for asynchronous interaction. Let $\mathcal{P} = (p_1, \ldots, p_m)$ denote a probability distribution on the m machines. Each machine j holds a (possibly stale) iterate x_j. At timestep k, a random machine j is chosen with probability p_j. This machine j sends an update h_j to the PS, based on x_j and the data it holds (that is, the functions f_i for $i \in S_j$). The PS sends machine j the current iterate, and machine j updates $x_j \leftarrow x^k$. Then the PS performs an update based on h_j to obtain x^{k+1}, the iterate for step $k + 1$. Then the process repeats, and a new machine is chosen independently from the distribution $(p_1, \ldots, p_m)$.

[2]We assume that all sets S_j have the same size, though if m does not divide n, we can reduce n until this is the case by combining pairs of functions to become one function.

Remark 1.1 (Relationship to prior delay models)**.** *Random delay models have been studied more generally in decentralized asynchronous settings (Lian et al., 2018; Ram et al., 2010; Jin et al., 2016) where they are often referred to as "random gossip". In a random gossip model, each machine has an exponentially distributed clock and wakes up to communicate an update with its neighbors each time it ticks.*

Our model is essentially the same as the random gossip model, restricted to a centralized communication graph, as in Jin et al. (2016). That is, our discrete delay model arises from a continuous-time model where each machine j takes T_j time to compute its update, where T_j is a random variable distributed according to an exponential distribution with parameter λ_j. After T_j time, machine j sends its update to the PS and receives an updated iterate $x_j \leftarrow x^k$. Then it draws a new (independent) work time T_j and repeats. The machines have independent work times but possibly different rates λ_j. Due to the memorylessness of exponential random variables, this continuous-time model is equivalent to the discrete-time model described above.

Similarly, our model generalizes the geometric delay model in (Mitliagkas et al., 2016) for centralized asynchronous gradient descent, which is the special case where the λ_j's are all the same.

We note that in our model, the PS has knowledge of the values p_i, and we use this in our algorithm. In practice, these rates can be estimated from the ratio between the number of updates from the machine j and the total number of iterations. Our empirical results also show that our approach performs well even an a vanilla implementation where the PS does not need to know the p_i.

Stochastic delays are well-motivated by applications. In the data-center setting, stochasticity in machine performance — and in particular, heavy tails in compute times — is well-documented (Dean and Barroso, 2013). Because we allow for the p_i's to be distinct for each machine, our model is well-suited to the example of federated learning, where we expect machines to have heterogeneous delays. Our particular model for stochastic delays is natural because it fits into the framework of randomized gossip and arises from independent exponentially distributed work times; however, we believe a more general model for stochastic delays could be more practical and merits further study.

Remark 1.2 (Blocking data)**.** *In our model, a machine sends an update computed from a single f_i in each round. In a practical implementation, we could block the data into blocks B_ℓ of size b, such that each machine computes b gradients before communicating with the PS. To apply our result, the set of functions*

Table 1: Comparison of related work for minimization of finite sums of n convex and L-smooth functions, whose average is μ-strongly convex and L_f-smooth. Here m denotes the number of machines or the minibatch size. We have substituted $O(m)$ for the maximum overlap bound τ in Leblond et al. (2018); Zhou et al. (2018), which is at least m, and $O(n)$ for the delay bound in IAG, which is as least n. We note that, as per Remark 1.1, our stochastic model is essentially a restriction of randomized gossip to a centralized communication graph.

Algorithm	Data Location	Delay Model	Convergence Guarantee	Iteration Complexity
This work (ADSAGA, Theorem B.1)	Distributed	Stochastic, $p_j = \Theta(\frac{1}{m})$ $\forall j$ (See Section 1.1)	$\mathsf{E}[f(x_k) - f(x^*)]$	$\tilde{O}\left(\left(n + \frac{L}{\mu} + \frac{\sqrt{mL_fL}}{\mu}\right)\log(1/\epsilon)\right)$
ASAGA, MiG	Shared	Arbitrary, bounded	$\mathsf{E}[f(x_k) - f(x^*)]$	$O\left((n + \frac{mL}{\mu})\log(1/\epsilon)\right)$
IAG	Distributed	Arbitrary, bounded	$f(x_k) - f(x^*)$	$O\left(\frac{n^2L}{\mu}\log(1/\epsilon)\right)$
Decentralized SGD (Lian et al., 2018)	Distributed	Random Gossip (see Remark 1.1)	$\mathsf{E}[f(x_k) - f(x^*)]$	$\tilde{O}(\frac{L^2}{\epsilon^2})$
FedAsync (Xie et al., 2019)	Distributed	Arbitrary, bounded	$\mathsf{E}[f(x_k) - f(x^*)]$	$\tilde{O}(\frac{L^2}{\epsilon^2})$
Minibatch SAGA (see supplementary material)	Shared or Distributed	Synchronous (No Delays)	$\mathsf{E}\|x^k - x^*\|_2^2$	$\tilde{O}\left(\left(n + \frac{L}{\mu} + \frac{mL_f}{\mu}\right)\log(1/\epsilon)\right)$

$\{f_i\}_i$ is replaced with $\{\sum_{i\in B_\ell} f_i\}_\ell$, yielding an iteration complexity of $\tilde{O}\left(n + bL/\mu + b\sqrt{mL_fL/\mu}\right)$.

1.2 Contributions

Our main technical contribution is the development and analysis of a SAGA-like algorithm, which we call ADSAGA, in the model described in Section 1.1. We show that with m machines, for μ-strongly convex, L_f-smooth functions f with minimizer x^*, when each f_i is convex and L-smooth, ADSAGA achieves $\mathsf{E}[f(x_k) - f(x^*)] \leq \epsilon$ in

$$k = \tilde{O}\left(\frac{1}{mp_{\min}}\left(n + \frac{L}{\mu} + \frac{\sqrt{mL_fL}}{\mu}\right)\log(1/\epsilon)\right) \quad (3)$$

iterations, where $p_{\min} = \min(p_j)$ is the minimum update rate parameter. Standard sequential SAGA achieves the same convergence in $O\left((n + \frac{L}{\mu})\log(1/\epsilon)\right)$ iterations. This implies that when the machine update rates vary by no more than a constant factor, if the term $n + L/\mu$ in our iteration complexity (3) dominates the final term $\sqrt{mL_fL/\mu}$, the ADSAGA with stochastic delays achieves the same iteration complexity as SAGA with no delays. On the other hand, when the $\sqrt{mL_fL/\mu}$ term dominates in (3), the convergence rate of ADSAGA scales with the *square root* of the number of machines, or average delay. We provide a quantitative comparison to related works in Table 1.

Remarkably, due to this $\sqrt{m}$ dependence, the convergence rate in (3) is dramatically *faster* than the rates proved for ASAGA (the analog in the shared-data setting with arbitrary delays), which as discussed above is at best $O(n + \frac{Lm}{\mu})$,[3] or even the rate of synchronous minibatch SAGA, which is $\tilde{O}(n + \frac{L}{\mu} + \frac{mL_f}{\mu})$. This is possible because of the stochastic delay model. Due to the occasional occurrence of very short delays (e.g., a machine j is drawn twice in quick succession), after the same number of iterations, the parallel depth of ADSAGA in our model is far deeper than that of a synchronous minibatch algorithm or than some instantiations of ASAGA with bounded delays. As evidenced by standard lower bounds in optimization, a high parallel depth is a prerequisite for convergence (Nesterov et al., 2018). For instance, with arbitrary bounded delays, the complexity would scale at least linearly with m due to the lower-bound zero-chain techniques. We believe this intuition and our result suggests optimistic news for asynchronous algorithms under more general stochastic delays: the iteration complexity may improve if the delays vary greatly, and are occasionally short.

The proof of our result uses a novel potential function to track our progress towards the optimum. In addition to including typical terms such as $f(x_k) - f(x^*)$ and $|x_k - x^*|_2^2$, our potential function includes a quadratic term that takes into account the dot product of $x_k - x^*$ and the expected next stale gradient update. This

[3] Note that, in ASAGA, the convergence rate scales with the maximum delay τ_{max}, which is lower bounded by m.

quadratic term is similar to the one that appears in the potential analysis of SAG(Roux et al., 2012) and in Assran and Rabbat (2020). Key to our analysis is a new *unbiased trajectory* lemma, which states that in expectation, the expected stale update moves towards the true gradient.

We support our theoretical claims with numerical experiments. In our first set of experiments, we simulate ADSAGA as well as other algorithms in the model of Section 1.1, and show that ADSAGA achieves better iteration complexity than other algorithms that have been analyzed in the distributed-data setting (IAG, SGD). These observations extend to a broader and more realistic shifted exponential delay model, proposed in Lee et al. (2017b). In our second set of experiments, we implement ADSAGA and other state-of-the-art distributed algorithms in a distributed compute cluster. We observe that in terms of wallclock time, ADSAGA performs similarly to IAG, and both of these asynchronous algorithms are over 60% faster with 30 machines than the synchronous alternatives (minibatch-SAGA and minibatch-SGD) or asynchronous SGD.

We refer the reader to Appendix A for an extended discussion of related work; see also the survey of Assran et al. (2020).

Remark 1.3 (Extensions to non-strongly convex objectives, and acceleration)**.** *We remark that the AdaptReg reduction in Allen-Zhu and Hazan (2016) can be applied to ADSAGA to extend our result to non-strongly convex objectives f. In this case, the convergence rate becomes $\tilde{O}(n + L/\epsilon + \sqrt{mL_f L}/\epsilon)$. Further, applying the black-box acceleration reduction in Lin et al. (2015) or Frostig et al. (2015) yields a convergence rate of $\tilde{O}(n + \sqrt{nL/\mu} + \sqrt{n\sqrt{mL_f L}/\mu})$. Both of the reductions use an outer loop around ADSAGA which requires breaking the asynchrony to serialize every $\Omega(n)$ iterations. We expect that using the outer loops without serializing would yield the same results.*

2 THE ADSAGA ALGORITHM

In this section, we describe the algorithm ADSAGA, a variant of SAGA designed for an the asynchronous, distributed data setting. Each machine maintains a local copy of the iterate x, which we denote x_j, and also stores a vector α_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 PS stores the current iterate x and maintains the average $\overline{\alpha}$ of the α_i. Additionally, to handle the case of heterogeneous update rates, the PS maintains a variable u_j for each machine, which stores a weighted history of updates from machine j.

We describe the algorithm formally in Algorithm 1. At a high level, each machine j chooses a function f_i randomly in S_j, and computes the variance-reduced stochastic gradient $h_j := \nabla f_i(x_j) - \alpha_i$, that is, the gradient of f_i at the current local iterate minus the prior gradient this machine computed for f_i. Meanwhile, upon receiving this vector h_j at the PS, the PS takes a gradient step in roughly the direction $h_j + \overline{\alpha}$.

If the machine update rates p_j are equal, $u_j = h_j$ always, so our algorithm is precisely an asynchronous implementation of the SAGA algorithm with delays. In our experiments, we implement this as a "Vanilla" version of ADSAGA. This is shown in Algorithm 2. The reader may wish to look at Algorithm 2 first, before looking at the more complicated Algorithm 1.

If the machine update rates p_j are heterogeneous, our algorithm differs in two ways from the vanilla algorithm. First, we use machine-specific step sizes η_j which scale inversely with the machine's update rate, p_j. Intuitively, this compensates for less frequent updates with larger weights for those updates. Second, we use the variables u_j so that the trajectory of the expected update to x tends towards the full gradient $\nabla f(x)$ (see Lemma 3.3).

We provide a logical view of ADSAGA (in the delay model described in Section 1.1) in Algorithm 3. We emphasize that Algorithm 1 is equivalent to Algorithm 3 given our stochastic delay model; we introduce Algorithm 3 only to aid the analysis. In the logical view, each logical iteration tracks the steps performed by the PS when h_j is sent to the PS from some machine j, followed by the steps performed by the machine j upon receiving the iterate x in return. We choose this sequence for a logical iteration because it implies that iterate x_j used to compute the local gradient in step M.2 equals the iterate x from the PS. Because the variable u_j is only modified in iterations which concern machine j, we are able to move step PS.2 to later in the logical iteration; this eases the analysis. For similar reasons, we move the step M.3 which updates α_i to the start of the logical iteration.

To make notation clearer for the analysis, in Algorithm 3, we index the central parameter with a superscript of the iteration counter k. Note that in this algorithm, we also introduce the auxiliary variables g_j, β_j and i_j to aid with the analysis. The variable i_j tracks the index of the function used to compute the update h_j at machine j, $g_j := \nabla_i(x_j)$, and $\beta_j := \alpha_{i_j}$.

3 CONVERGENCE RESULT AND PROOF OVERVIEW

In this section, we state our main result and sketch the proof. We use $\tilde{O}$-notation to hide logarithmic factors

Algorithm 1 Asynchronous Distributed SAGA (ADSAGA)

Process PARAMETER SERVER$(\eta, \{p_j\}, x^{(0)}, t)$:
$u_j = 0$ for $j \in [m]$. Initialize update variables
$\overline{\alpha} = 0$. Initialize last gradient averages
repeat
C.1) Receive h_j from machine j
C.2) Send x to machine j
PS.1) $u_j \leftarrow u_j \left(1 - \frac{p_{\min}}{p_j}\right) + h_j$. First update to u_j
PS.2) $x \leftarrow x - \eta_j \left(u_j + \overline{\alpha}\right)$
. Apply stale gradient update using step size $\eta_j = \frac{p_{\min}}{p_j}$
PS.3) $\overline{\alpha} \leftarrow \overline{\alpha} + \frac{1}{n} h_j$. Update gradient averages
PS.4) $u_j \leftarrow u_j - \frac{m}{n} h_j$. Second update to u_j
until t iterations have passed
Return x

Process WORKER MACHINE$(\{f_i\}_{i \in S_j}, x_j)$:
$h_j = 0$ for $j \in [m]$. Initialize updates
$\alpha_i = 0$ for $i \in [n]$. Initialize last gradients
repeat
C.1) Send h_j to PS
C.2) Receive current iterate x from PS and set $x_j \leftarrow x$
M.1) $i \sim \text{Uniform}(S_j)$. Choose a random function
M.2) $g_j \leftarrow \nabla f_i(x_j)$. Compute the gradient of f_i
M.3) $h_j \leftarrow g_j - \alpha_i$
. Reduce variance by subtracting last time's f_i
M.3) $\alpha_i \leftarrow g_j$. Update last gradient locally
until terminated by PS

in n or in constants depending on the functions f_i. Logarithmic factors are standard for algorithms that use a constant step size, though we expect they could be removed by using a decreasing step size.

Theorem 3.1. *Let $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$ be an L_f-smooth and μ-strongly convex function. Suppose that each f_i is L-smooth and convex. Let $r := \frac{8\ 76+168\ \frac{p_{max}}{p_{min}}^{\ 2}\ \frac{m}{n}}{3}$. For any partition of the n functions to m machines, after*

$$k = \frac{1}{m p_{min}} \left(4n + 2r\frac{L}{\mu} + 2\sqrt{r}\frac{\sqrt{mL_f L}}{\mu}\right) \times \log\left(\frac{\left(1 + \frac{1}{2m\mu\eta}\right)\left(f(x^0) - f(x^*)\right) + \frac{n\sigma^2}{2L}}{\epsilon}\right)$$

iterations of Algorithm 1 at the PS with $\eta = \frac{1}{2rL + 2\sqrt{rmL_f L}}$, we have $\mathsf{E}\left[f(x^k) - f(x^*)\right] \le \epsilon$, where $\sigma^2 = \frac{1}{n}\sum_i |\nabla f_i(x^*)|_2^2$.

Algorithm 2 Vanilla ADSAGA (all p_j equal)

Process PARAMETER SERVER$(\eta, x^{(0)}, t)$:
$\overline{\alpha} = 0$. Initialize last gradient averages
repeat
C.1) Receive h_j from machine j
C.2) Send x to machine j
PS.1) $x \leftarrow x - \eta\left(h_j + \overline{\alpha}\right)$
. Apply stale gradient update
PS.2) $\overline{\alpha} \leftarrow \overline{\alpha} + \frac{1}{n} h_j$. Update gradient averages
until t iterations have passed
Return x

Process WORKER MACHINE$(\{f_i\}_{i \in S_j}, x_j)$:
. Identical to WORKER MACHINE in Algorithm 1

Theorem 3.1 yields a convergence rate of $\tilde{O}\left(\left(n + \frac{L}{\mu} + \frac{\sqrt{mL_f L}}{\mu}\right)\log(1/\epsilon)\right)$ when $p_j = \Theta\left(\frac{1}{m}\right)$ for all j, that is, all machines perform updates with the same frequency up to a constant factor.

The proof of Theorem 3.1 can be found in the supplementary material. The key elements of our proof are a new *Unbiased Trajectory Lemma* (Lemma 3.3) and a novel potential function which captures progress both in the iterate x and in the stale gradients.

We begin by introducing some notation which will be used in defining the potential function. We use I_d to denote the identity matrix in $\mathsf{R}^{d \times d}$, and $\mathbf{1}$ to denote the all-ones vector. Let H, G, and U be the matrices whose jth columns contain the vectors $h_j = g_j - \beta_j$, g_j, and u_j respectively. We use the superscript k to denote the value of any variable from Algorithm 1 at the *beginning* of iteration k (at the PS). When the iteration k is clear from context, we will eliminate the superscripts k. To further simplify, we will use the following definitions: $\alpha_i^* := \alpha_i - \nabla f_i(x^*)$, $\beta_j^* := \beta_j - \nabla f_{i_j}(x^*)$, and $g_j^* := g_j - \nabla f_{i_j}(x^*)$, where i_j is the index of the function used by machine j to compute g_j and β_j, as above.

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

$$\phi(x, G, H, U, \alpha, \beta) := \sum_{\ell} \phi_\ell(x, G, H, U, \alpha, \beta),$$

Algorithm 3 Asynchronous Distributed SAGA (ADSAGA): Logical View, in the model in Section 1.1

Input: $x^0, \eta, \{f_i\}, \{S_j\}, \mathcal{P}, t$
$g_j, h_j, u_j, \beta_j = 0$ for $j \in [m]$. Initialize variables to 0
$i_j \sim \text{Uniform}(S_j)$ for $j \in [m]$
. Initialize last gradient indicators
$\alpha_i = 0$ for $i \in [n]$. Initialize last gradients
$\overline{\alpha} = 0$. Initialize last gradient averages at PS
for $k = 0$ **to** t **do**
$j \sim \mathcal{P}$; . Choose machine j with probability p_j
M.3) $\alpha_{i_j} \leftarrow g_j$. Update last gradient locally
PS.2) $x^{k+1} \leftarrow x^k - \eta_j (u_j + \overline{\alpha})$
. Take gradient step using $_j = \frac{p_{\min}}{p_j}$
PS.3) $\overline{\alpha} \leftarrow \overline{\alpha} + \frac{1}{n} h_j$. Update gradient averages at PS
PS.4) $u_j \leftarrow u_j - \frac{m}{n} h_j$. First update to u_j
M.1) $i \sim \text{Uniform}(S_j)$
. Choose uniformly a random function at machine j
$g_j \leftarrow \nabla f_i(x^k)$. Update auxiliary variable g_j
$\beta_j \leftarrow \alpha_i$. Update auxiliary variable $_j$
M.2) $h_j \leftarrow \nabla f_i(x_j) - \alpha_i$
. Prepare next update to be sent to PS
PS.1) $u_j \leftarrow u_j \left(1 - \frac{p_{\min}}{p_j}\right) + \frac{p_{\min}}{p_j} h_j$
. Second update of u_j at PS; this could also be done locally
$i_j \leftarrow i$. Update auxiliary variable i_j
end for
Return x^t

where $\phi_i = \phi_i(x, G, H, U, \alpha, \beta)$ are given by

$$\phi_1 := 4m\eta \left(f(x) - f(x^*)\right),$$

$$\phi_2 := \begin{pmatrix} x - x^* \\ \eta(U\mathbf{1} + m\overline{\alpha}) \end{pmatrix}^T \begin{pmatrix} I_d & -I_d \\ -I_d & 2I_d \end{pmatrix} \begin{pmatrix} x - x^* \\ \eta(U\mathbf{1} + m\overline{\alpha}) \end{pmatrix},$$

$$\phi_3 := \eta^2 c_3 \sum_j \frac{p_{\min}}{p_j} |g_j^*|_2^2,$$

$$\phi_4 := \eta^2 c_4 \left(2 \sum_i \frac{p_{\min}}{p_j} |\alpha_i^*|_2^2 - \sum_j \frac{p_{\min}}{p_j} |\beta_j^*|_2^2 \right),$$

$$\phi_5 := \eta^2 c_5 \sum_j |u_j|_2^2,$$

The exact values of the constants c_3, c_4 and c_5 are given in the supplementary material.

This potential function captures not only progress in $f(x^k) - f(x^*)$ and $|x^k - x^*|_2^2$, but also the extent to which the expected stale update, $\frac{1}{m} U\mathbf{1} + \overline{\alpha}$, 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 $\frac{1}{m} U\mathbf{1} + \overline{\alpha}$. The exact coefficients in the potential function are chosen to cancel extraneous quantities that arise when evaluating the expected difference in potential between steps. In the rest of the text, we abbreviate the potential $\phi(x^k, G^k, H^k, U^k, \alpha^k, \beta^k)$, given by the variables at the start of the kth iteration, by $\phi(k)$. Below, in the expectations we implicitly condition on the history $\{x^k, G^k, H^k, U^k, \alpha^k, \beta^k\}$.

The following is our main technical proposition.

Proposition 3.2. *In ADSAGA, for any step size* $\eta \leq \frac{1}{2rL + 2\sqrt{rmL_f L}}$, $\mathsf{E}_{i,j}[\phi(k+1)] \leq (1 - \gamma)\,\phi(k)$, *where* $\gamma = mp_{min} \min\left(\frac{1}{4n}, \mu\eta\right)$, *and* r *is a constant defined in Theorem 3.1 dependent only on* $\frac{p_{max}}{p_{min}}$.

We sketch the proof of this proposition. For ease of presentation only, we assume that $p_j = \Theta\left(\frac{1}{m}\right)$ for all j. The formal proof, which contains precise constants and the dependence on $p_{\min}$, is given in the supplementary material. We begin by stating the Unbiased Trajectory lemma, which shows that the expected update to x moves in expectation towards the gradient $\nabla f(x)$.

Lemma 3.3 (Unbiased Trajectory)**.** *At iteration* k *of ADSAGA at the PS,*

$$\mathsf{E}_{i,j}[x^{k+1}] = x^k - \eta p_{min} \left(U^k\mathbf{1} + m\overline{\alpha}^k\right),$$

and

$$\mathsf{E}_{i,j}\left[U^{k+1}\mathbf{1} + m\overline{\alpha}^{k+1}\right] = p_{min}\left(1 - \frac{1}{n}\right)\left(U^k\mathbf{1} + m\overline{\alpha}^k\right) + mp_{min}\nabla f(x^k).$$

Essentially, the Unbiased Trajectory lemma states that, in expectation, the update approaches the "desired" update, which is proportional to $\nabla f(x)$. Using this condition, we can control the expected change in $\phi_1 + \phi_2$. It turns out that the first-order terms of this contribution yield a significant decrease in the potential ϕ. However, there are some second order terms, involving $|\alpha_i^*|^2, |u_j|^2, |g_j^*|^2$, and $|\beta_j^*|^2$, which go in the wrong direction and complicate matters. This is the reason for the terms ϕ_3, ϕ_4, and ϕ_5 in the potential function. We are able to choose the constants in those terms so that they exactly cancel the problematic second-order terms from $\phi_1 + \phi_2$. Combining all of this eventually yields Proposition 3.2.

4 EXPERIMENTS

We conduct experiments to compare the convergence rates of ADSAGA to other state-of-the-art algorithms: SGD, IAG, ASAGA, and minibatch SAGA. In our first set of experiments, we simulate the stochastic delay model of Section 1.1. In our second set, we implement these algorithms in a distributed compute cluster. The main takeaways of our experiments are the following:

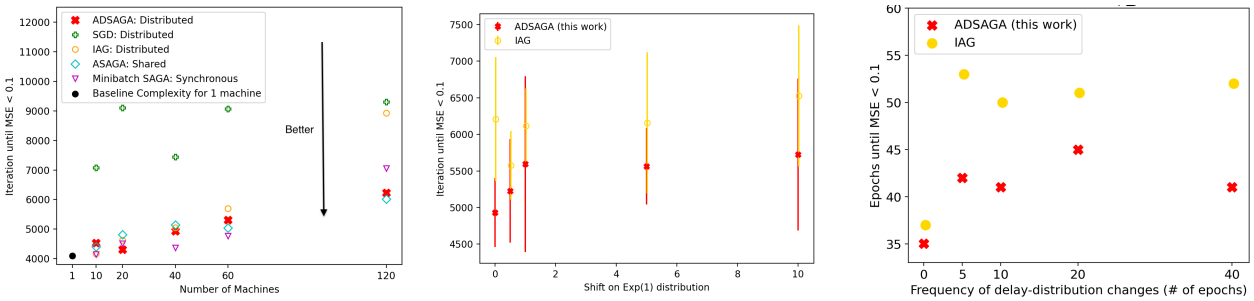


Figure 2: (a) Comparison of ADSAGA (this work), ASAGA (Leblond et al., 2018), minibatch SAGA (Gazagnadou et al., 2019), SGD (Lian et al., 2018), and IAG (Gurbuzbalaban et al., 2017): Iteration complexity to achieve $|x_k - x^*|_2^2 \leq 0.1$, averaged over 8 runs. (b) Comparison of IAG and ADSAGA with shifted-exponential delays for 60 machines. (c) The exponential update rates are resampled uniformly at random from [1, 10] with varying frequencies (given on x-axis). $x = 0$ corresponds to no heterogeneity. Results shown for 60 nodes.

(a) (b)

Figure 3: (a) Convergence accuracy after 100 epochs on 20 machines. (b) Wallclock time to achieve $|x_k - x^*|_2^2 \leq 10^{-10}$, averaged over 8 runs.

1. In experiments on a cluster, ADSAGA is comparable to IAG and outperforms all other algorithms in wall-clock time.

2. In experiments with simulated (shifted) exponential delays, ADSAGA outperforms IAG in iteration complexity.

3. ADSAGA performs well even without knowledge of the update rates $\{p_j\}$, even with significant machine heterogeneity.

Data. For all experiments, we simulate these algorithms on a randomly-generated least squares problem $\min_{\hat{x}} |A\hat{x} - b|_2^2$. Here $A \in \mathsf{R}^{n \times d}$ 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 $Z \sim \mathcal{N}(0, \sigma^2 I_n)$. In the first set of experiments with simulated delays, we choose $n = 120$, $d = 60$, and $\sigma = 1$. For the the second set of experiments on the distributed cluster, we choose a larger 10GB least squares problem with $n = 600000$ and $d = 200000$, and $\sigma = 100$.

Remark 4.1. *We chose to perform our experiments on least squares problem to better understand how different algorithms were affected by the distributed-data setting, without introducing complexity that would be hard to interpret. Since performing asynchronous algorithm on a distributed cluster can already produce results that are difficult to reproduce, we choose a simple dataset to allow our experiments to be more consistent and interpretable.*

Simulated Delays. In our first set of experiments, we empirically validate our results by simulating the stochastic delay model described in Section 1.1, and in a more general delay model. First we simulate ADSAGA, SGD, IAG in the distributed data setting in the delay

model of Section 1.1 with all p_i equal; we simulate ASAGA in this model in the shared-data setting; and we also compare to minibatch SAGA assuming a *synchronous* implementation with a minibatch size m equal to the number of machines. We run all five algorithms on $\hat{x}$ with m machines for $m \in \{10, 20, 40, 60, 120\}$. To be fair to all algorithms, we use a grid search to find the best step size in $\{0.05 \times i\}_{i\in[40]}$, observing that none of the best step sizes were at the boundary of this set.

In Figure 2, we plot expected iteration complexity to achieve $|\hat{x}-x^*|_2^2 \leq 0.1$, where $x^* := \min_{\hat{x}} |A\hat{x}-b|_2^2$. Figure 2(a) demonstrates that, in the model in Section 1.1, ADSAGA outperforms the other algorithms for distributed data, SGD and IAG, especially as the number of machines m grows. In a more practical setting with shifted-exponential delays (no longer memoryless, but still heavy-tailed), with 60 machines, ADSAGA still outperforms IAG (Figure 2(b)). Shifted exponential or heavy-tailed work-time distributions are observed in practice (Dean and Barroso (2013); Lee et al. (2017b)). In Figure 2(c), we simulate with time-varying delays. Here, the update rates are chosen to be exponentially distributed with heterogeneous parameters, chosen uniformly at random from $[1, 10]$. These parameters are resampled during training with varying frequencies.

Distributed Experiments. In our second set of experiments, we run the five data-distributed algorithms (ADSAGA, SGD, IAG, minibatch-SAGA, and minibatch-SGD) in a distributed compute cluster and compare their wallclock times to convergence.[4] In the three asynchronous algorithms, the PS waits to receive a gradient update from a node; sends the current parameter back to that node; and performs the appropriate update. In synchronous minibatch-SGD, the PS waits until updates have been received by all nodes before performing an update and sending the new parameter to all nodes. We run all algorithms with m worker nodes for $m \in \{5, 10, 15, 20, 30\}$. The supplementary material has more details about our implementation.

We implemented the vanilla version of ADSAGA, Algorithm 2, which does not require knowing the update rates p_j. While we measured substantial heterogeneity in the update rates of each machine — some machines made to twice as many updates as others — we observed that the vanilla ADSAGA implementation worked as well as the full implementation of Algorithm 1. For all algorithms, we use a block size of 200 (Remark 1.2), and we perform a hyperparameter grid search over step sizes to find the hyperparameters which yield the smallest distance $\hat{x} - x^*$ after 100 epochs. Note the vanilla implementation of ADSAGA is the same as the implementation of ASAGA; the difference is that data is distributed in our our experiments.

In Figure 3, we compare performance in terms of both iteration complexity and wallclock time. Figure 3(a) plots the accuracy $|\hat{x} - x^*|_2^2$ after 100 epochs, where $x^* := \min_{\hat{x}} |A\hat{x} - b|_2^2$. We observe that the algorithms that do not use variance reduction (asynchronous SGD and minibatch-SGD) do substantially worse. ADSAGA and IAG perform similarly, while (synchronous) minibatch-SAGA performs slightly better in terms of iteration complexity. In Figure 3(b), we plot the expected wallclock time to achieve to achieve $|\hat{x} - x^*|_2^2 \leq 10^{-10}$. We only include the variance-reduced algorithms in this plot, as SGD did not converge in a reasonable amount of time. Figure 3(b) demonstrates that while synchronous minibatch-SAGA may have better iteration complexity, due to the cost of waiting for all workers to synchronize at each iteration, the asynchronous algorithms (IAG and ADSAGA) perform better in terms of wallclock time. Both IAG and ADSAGA perform similarly. The advantage of asynchrony increases with the number of machines: while with 5 machines the asynchronous algorithms are only 20% faster, with 30 machines, they are 60% faster.

5 Conclusion and Open Questions

In this paper, we introduced and analyzed ADSAGA, a SAGA-like algorithm in an asynchronous, distributed-data setting. We showed that in a particular stochastic delay model, ADSAGA achieves convergence in $\tilde{O}\left(\left(n + \frac{L}{\mu} + \frac{\sqrt{mL_f L}}{\mu}\right)\log(1/\epsilon)\right)$ iterations. To the best of our knowledge, this is the first provable result for asynchronous algorithms in the distributed-data setting — under any delay model — that scales both logarithmically in $1/\epsilon$ and linearly in n.

This work leaves open several interesting questions:

1. For arbitrary but bounded delays in the distributed setting (studied in Gurbuzbalaban et al. (2017); Aytekin et al. (2016); Vanli et al. (2018)), is the dependence on n^2 in the iteration complexity optimal?

2. In the *decentralized* random gossip setting of Figure 1(c), what rates does a SAGA-like algorithm achieve?

3. How would the local-SGD or local minibatching approaches, which are popular in federated learning, perform in the presence of stochastic delays?

[4] We do not simulate the shared-data ASAGA because the full dataset is too large to fit in RAM, and loading the data from memory is very slow.

Acknowledgements

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

References

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

Zeyuan Allen-Zhu and Elad Hazan. Optimal black-box reductions between optimization objectives. In Advances in Neural Information Processing Systems, pages 1614–1622, 2016.

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.

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

Mahmoud S Assran and Michael G Rabbat. Asynchronous gradient push. IEEE Transactions on Automatic Control, 66(1):168–183, 2020.

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.

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

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

Sorathan Chaturapruek, John C Duchi, and Christopher Re. 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.

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

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.

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.

Roy Frostig, Rong Ge, Sham Kakade, and Aaron Sidford. Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization. In International Conference on Machine Learning, pages 2540–2548. PMLR, 2015.

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.

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

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.

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

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In Advances in neural information processing systems, pages 315–323, 2013.

Remi 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.

Jason D Lee, Qihang Lin, Tengyu Ma, and Tianbao Yang. Distributed stochastic variance reduced gradient methods by sampling extra data with replacement. The Journal of Machine Learning Research, 18(1):4404–4446, 2017a.

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, 2017b.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. arXiv preprint arXiv:1812.06127, 2018.

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.

Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In

Advances in neural information processing systems, pages 3384{3392, 2015.

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.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-e cient learning of deep networks from decentralized data. In Arti cial Intelligence and Statistics, pages 1273{1282. PMLR, 2017.

Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Re. 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.

Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and mitigating data stalls in dnn training. arXiv preprint arXiv:2007.06775, 2020.

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.

Yurii Nesterov et al. Lectures on convex optimization, volume 137. Springer, 2018.

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

Kenta Niwa, Guoqiang Zhang, W Bastiaan Kleijn, Noboru Harada, Hiroshi Sawada, and Akinori Fujino. Asynchronous decentralized optimization with implicit stochastic variance reduction. In International Conference on Machine Learning, pages 8195{8204. PMLR, 2021.

S Sundhar Ram, Angelia Nedic, 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.

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.

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.

Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence rate for nite training sets. In Advances in neural information processing systems, pages 2663{2671, 2012.

Sebastian U Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for sgd with delayed gradients and compressed updates. Journal of Machine Learning Research, 21:1{36, 2020.

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

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.

Blake Woodworth, Kumar Kshitij Patel, and Nathan Srebro. Minibatch vs local sgd for heterogeneous distributed learning. arXiv preprint arXiv:2006.04735, 2020.

Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. arXiv preprint arXiv:1903.03934, 2019.

Jiaqi Zhang and Keyou You. Asyspa: An exact asynchronous algorithm for convex optimization over digraphs. IEEE Transactions on Automatic Control, 65(6):2494{2509, 2019.

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.

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.

Jiacheng Zhuo, Qi Lei, Alex Dimakis, and Constantine Caramanis. Communication-e cient asynchronous stochastic frank-wolfe over nuclear-norm balls. In International Conference on Arti cial Intelligence and Statistics, pages 1464{1474. PMLR, 2020.

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.

A Extended Related Work

We survey the most related gradient-based asynchronous algorithms for strongly convex optimization, focusing on results for the setting of nite sums. For completeness, we state some results that apply to the more general setting of optimization over (possibly non- nite) data distributions. See Table 1 for a quantitative summary of the most relevant other works.

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 Zinkevich et al. (2010); Dekel et al. (2012). For nite sum minimization, minibatch SAGA is analyzed in Gazagnadou et al. (2019); Bibi et al. (2018); Gower et al. (2018); Nitanda et al. (2019), achieving a convergence rate of $O\left(n + \frac{L}{m} + \frac{mL_f}{m}\right)\log(1/\epsilon)$ for a minibatch size of m with distributed data. [5] Distributed SVRG is analyzed in Lee et al. (2017a). Katyusha (Allen-Zhu, 2017) presents an accelerated, variance reduced parallelizable algorithm for nite sums with convergence rate $O\left(n + \sqrt{\frac{nL}{m}} + m\sqrt{\frac{L_f}{m}}\right)\log(1/\epsilon)$; this rate is proved to be near-optimal in Nitanda et al. (2019). Woodworth et al. (2020) compares local mini-batching and local-SGD for the more general setting of non-i.i.d data.

Asynchronous Centralized Algorithms with Shared Data (Figure 1(a)) Centralized asynchronous algorithms often arise in shared-memory architectures or in compute systems with a central parameter server. The textbook (Bertsekas and Tsitsiklis, 1989) 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 , sublinear convergence rates of $O(\frac{1}{\epsilon})$ matching those of SGD were achieved for strongly convex stochastic optimization in Recht et al. (2011) (under sparsity assumptions) and Chaturapruek et al. (2015). For nite sum minimization, linear convergence is proved for asynchronous variance-reduced algorithms in Mania et al. (2015); Zhao and Li (2016); Reddi et al. (2015); Leblond et al. (2018); Zhou et al. (2018); Zhuo et al. (2020); the best known rate of $O\left(n + \frac{L}{}\right)\log(1/\epsilon)$ is achieved by ASAGA (Leblond et al., 2018) and MiG (Zhou et al., 2018), 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. Arjevani et al. (2020) and Stich and Karimireddy (2020) consider the setting where all delays are exactly equal to .

Asynchronous Centralized Algorithms with Distributed Data (Figure 1(b)) Several works (Gurbuzbalaban et al., 2017; Aytekin et al., 2016; Vanli et al., 2018) consider incremental aggregated gradient (IAG) algorithms, which use the update $U(i;x) = \nabla f_i(x) + \sum_{j\neq i} g_j$, 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 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. Xie et al. (2019) studies asynchronous federated optimization with arbitrary (bounded) delays; this work achieves a convergence rate that scales with $\frac{1}{}$.

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 (Ram et al., 2010; Jin et al., 2016; Lian et al., 2018) have considered the setting of randomized gossip, where each machine has an exponentially distributed clock and wakes up to communicate with its neighbors each time it ticks. In Lian et al. (2018), 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 model. Tian et al. (2020); Niwa et al. (2021); Assran and Rabbat (2020); Zhang and You (2019) study a decentralized setting with arbitrary but bounded delays. For strongly convex objectives f, Tian et al. (2020) achieves a linear rate of convergence gradient tracking techniques. However, the dependence on the number of nodes in this work is exponential.

We refer the reader to Assran et al. (2020) for a recent survey on asynchronous parallel optimization algorithms for a more complete discussion of compute architectures and asynchronous algorithms such as coordinate decent

[5] We prove this result in Proposition E.1 of the appendix, as the cited works (Gazagnadou et al., 2019; Bibi et al., 2018; Gower et al., 2018) prove slightly weaker bounds for minibatch SAGA using di erent condition numbers.

methods that are beyond the scope of this section.

B Restatement of Convergence Result and Detailed Proof Overview

In this section, we state and sketch the proof of our main result, which yields a convergence rate of $O\left(n + \frac{L}{\mu} + \frac{\sqrt{mL_f L}}{\mu}\right)\log(1/\epsilon)$ when $p_j = \frac{1}{m}$ for all j, that is, all machines perform updates with the same frequecy up to a constant factor.

Theorem B.1. Let $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$ be an L_f-smooth and μ-strongly convex function. Suppose that each f_i is L-smooth and convex. Let $r := \frac{8}{3}\left(76+168\left(\frac{p_{max}}{p_{min}}\right)^2\frac{m}{n}\right)$. For any partition of the n functions to m machines, after

$$k = mp_{min}\left(4n + 2r\frac{L}{\mu} + 2\sqrt{r}\sqrt{\frac{mL_f L}{\mu^2}}\right)\log\left(\frac{1 + \frac{1}{2m}}{\epsilon}\left(f(x^0) - f(x^*) + \frac{n\sigma^2}{2L}\right)\right)$$

iterations of Algorithm 3 with $\eta = \frac{1}{2rL + 2\sqrt{rmL_f L}}$, we have $\mathbb{E}\left[f(x^k) - f(x^*)\right] \le \epsilon$, where $\sigma^2 = \frac{1}{n}\sum_i \|\nabla f_i(x^*)\|_2^2$.

We prove this theorem in Section C. The key elements of our proof are the Unbiased Trajectory Lemma (Lemma B.3) and a novel potential function which captures progress both in the iterate x_k and in the stale gradients. Throughout, all expectations are over the choice of P and $i \sim \text{Uniform}(S_j)$ in each iteration of the logical algorithm.

We begin by introducing some notation which will be used in defining the potential function. Let H, G, and U be the matrices whose jth columns contain the vectors $h_j = g_{j-1}$, g_j, and u_j respectively. We use the superscript k to denote the value of any variable from Algorithm 3 at the beginning of iteration k. When the iteration k is clear from context, we will eliminate the superscript k. To further simplify, we will use the following definitions: $\nabla_i := \nabla f_i(x)$, $\alpha_j := \alpha_{j-1} - \eta\nabla f_{i_j}(x)$, and $g_j := g_{j-1} - \nabla f_{i_j}(x)$, where i_j is the index of the function used by machine j to compute g_j and α_j, as indicated in Algorithm 3.

We will analyze the expectation of the following potential function $\Phi(x, G, H, U, \alpha, \gamma)$:

$$\Phi(x, G, H, U, \alpha, \gamma) := \sum_{\ell=1}^{5}\Phi_\ell(x, G, H, U, \alpha, \gamma),$$

where

$$\Phi_1(x, G, H, U, \alpha, \gamma) := 4m\left(f(x) - f(x^*)\right),$$

$$\Phi_2(x, G, H, U, \alpha, \gamma) := \begin{pmatrix} x - x^* \\ (U1 + m\eta) \end{pmatrix}^T \begin{pmatrix} I_d & I_d \\ I_d & 2I_d \end{pmatrix} \begin{pmatrix} x - x^* \\ (U1 + m\eta) \end{pmatrix},$$

$$\Phi_3(x, G, H, U, \alpha, \gamma) := \gamma^2 c_3 \sum_j \frac{p_{min}}{p_j}\|g_j\|_2^2,$$

$$\Phi_4(x, G, H, U, \alpha, \gamma) := \gamma^2 c_4\left(2\sum_i \frac{p_{min}}{p_j}\|\nabla_i\|_2^2 - \sum_j \frac{p_{min}}{p_j}\|\nabla_j\|_2^2\right),$$

$$\Phi_5(x, G, H, U, \alpha, \gamma) := \gamma^2 c_5 \sum_j \|u_j\|_2^2,$$

and

$$c_5 := \frac{16}{3}(mL_f + 1),$$

$$c_3, c_4 = \left(1 + \frac{m}{n}\left(\frac{p_{max}}{p_{min}}\right)^2\right)c_5.$$

The exact values of c_3 and c_4 are given in the Appendix.

It is easy to check that the potential function is non-negative. In particular, $_1$ and $_3$, $_5$ are clearly non-negative, and $_2$ is non-negative because $^1_1$ $_2^1$ is positive de nite. Finally, $_4$ is non-negative because the terms in the sum over j are a subset of the terms in the sum over i.

This potential function captures not only progress in $f(x^k)$ $f(x)$ and jx^k $x\ j_2^2$, but also the extent to which the expected stale update, $\frac{1}{m}U1 + \bar{}$, 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 $\frac{1}{m}U1 + \bar{}$. The exact coe cients in the potential function are chosen to cancel extraneous quantities that arise when evaluating the expected di erence in potential between steps. In the rest of the text, we abbreviate the potential $(x^k; G^k; H^k; U^k; \ ^k; \ ^k)$, given by the variables at the start of the kth iteration, by (k). All expectations below are over the random choices of j P and i $\mathrm{Uniform}(S_j)$ in the kth iteration of Algorithm 3, and we implicitly condition on the history $f x^k; G^k; H^k; U^k; \ ^k; \ ^k g$ in such expectations.

The following proposition is our main technical proposition.

Proposition B.2. In Algorithm 3, for any step size $\frac{}{2rL + 2^p\overline{rmL_f L}}$,

$$E_{i;j}[\ (k+1)] \quad (1 \quad)\ (k)$$

where $= mp_{min}\ \min\ \frac{1}{4n}; \quad$, and r is a constant de ned in Theorem B.1 dependent only on $\frac{p_{max}}{p_{min}}$.

We sketch the proof of this proposition. For ease of presentation, we assume in this section that $p_j = \frac{1}{m}$ for all j. The formal proof, which contains precise constants and the dependence on p_{min}, is given in Section C. We begin by stating the Unbiased Trajectory lemma, which shows that the expected update to x moves in expectation towards the gradient $r\ f(x)$.

Lemma B.3 (Unbiased Trajectory). At any iteration k, we have

$$E_{i;j}[x^{k+1}] = x^k \quad p_{min}\ U^k 1 + m^{-k};$$

and

$$E_{i;j}\ U^{k+1}1 + m^{-k+1} = p_{min}\ 1 \quad \frac{1}{n} \quad U^k 1 + m^{-k} + mp_{min}\ r\ f(x^k):$$

Using this condition, we can control the expected change in $_1 + \ _2$, yielding the following lemma, stated with precise constants in Section C. Let $q := 1 + \ mL_f$.

Lemma B.4. (Informal)

$$\begin{aligned} E_{i;j}[\ _1(k+1) + \ _2(k+1)] \quad & _1(k) \quad _2(k) \\ & 2p_{min} \quad \begin{matrix} x & x \\ 0 & \end{matrix}^T \begin{matrix} 0 & 0 \\ 0 & I_d \end{matrix} \begin{matrix} x & x \\ (U1 + m^{-}) & \end{matrix} \quad 2mp_{min} \quad \begin{matrix} (x & x\ )^T r\ f(x) \\ 1 & \end{matrix} \\ & + \quad ^2\ @\frac{q}{n}\sum_i j\ _i j_2^2 + \frac{q}{n}\sum_j ju_j j_2^2 + \frac{1}{n}\sum_j jg_j j_2^2 + j\ _j j_2^2\ A \\ & + O(\ ^2)\frac{1}{n}\sum_i jr\ f_i(x) \quad r \quad f_i(x\ )j_2^2: \end{aligned}$$

The first two terms of this lemma yield a signi cant decrease in the potential. However, the potential may increase from the remaining second order terms, which come from the variance of the update. We can cancel the second order term involving the $j\ _i j^2$, $ju_j j^2$, $jg_j j^2$, and $j\ _j j^2$ terms by considering the expected change in potential in $_3$, $_4$ and $_5$, captured in the next lemma, formally stated in Section C. While we use big-O notation here, as one can see in the formal lemma in Section C, the exact constants in $_3$, $_4$ and $_5$ are chosen to cancel the second order terms in Lemma B.4.

Lemma B.5. (Informal)

$$E[\ _3(k+1) + \ _4(k+1) + \ _5(k+1)] \quad (\ _3(k) + \ _4(k) + \ _5(k)) \qquad \left(1 - \frac{mp_{min}}{4n}\right)(\ _3(k) + \ _4(k) + \ _5(k))$$
$$^2\left(\frac{q}{n}\sum_i \ j\ _i\ j_2^2 + \frac{q}{n}\sum_j \ ju_j\ j_2^2 + \frac{1}{n}\sum_j \ jg_j\ j_2^2 + j\ _j\ j_2^2\right)$$
$$+ O\left(^2 q\ \frac{1}{n}\sum_i \ jr\ f_i(x) \quad r\ \ f_i(x\ )j_2^2\right):$$

Intuitively, the $1 - \frac{mp_{min}}{n}$ contraction in this lemma is possible because in each iteration, with probality at least $\frac{mp_{min}}{n}$, any one of the n variables $_{i_j}$ is replaced by the variable g_j, which is in turn replaced by a fresh gradient $r\ f_i(x)$.

Finally, we use the smoothness of each f_i to bound $\frac{1}{n}\sum_i jr\ f_i(x) \quad r\ \ f_i(x\ )j_2^2$ by $L(x \quad x\ )^T r\ f(x)$ (Lemma C.9). This allows us to cancel all of the $\frac{1}{n}\sum_i jr\ f_i(x) \quad r\ \ f_i(x\ )j_2^2$ terms with the negative $(x \quad x\ )r\ f(x)$ term in Lemma B.4. We show that if $\frac{p}{L + \sqrt{mL_f L}}$, the negative $2mp_{min}\ (x \quad x\ )^T r\ f(x)$ term in Lemma B.4 dominates the positive $O\left(^2 q\ \frac{1}{n}\sum_i jr\ f_i(x) \quad r\ \ f_i(x\ )j_2^2\right)$ term (Claim C.10). Using the $$-strong convexity of f, we show that the remaining fraction of the negative $2mp_{min}\ (x \quad x\ )^T r\ f(x)$ term leads to a negative $2mp_{min}\ jx \quad x\ j_2^2$ and $\frac{m^2 p_{min}}{n}(f(x) \quad f(x\ ))$ term.

Combining Lemma B.4 and Lemma B.5 with the observations above, we obtain the following lemma:

Lemma B.6. For $\frac{p}{L + \sqrt{mL_f L}}$,

$$E[\ (k+1)] \qquad (k) \qquad 2p_{min} \quad \begin{pmatrix} y \\ (U1 + m^{-}) \end{pmatrix}^T \begin{pmatrix} \frac{m}{2} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y \\ (U1 + m^{-}) \end{pmatrix}$$
$$\frac{mp_{min}}{4n}(\ _1(k) + \ _3(k) + \ _4(k) + \ _5(k))$$

Some linear-algebraic manipulations (Lemma C.12) yield Proposition B.2.

C Proof of ADSAGA Convergence

In this section we prove Theorem B.1, restated as Theorem A.1.

Theorem C.1. Let $f(x) = \frac{1}{n}\sum_{i=1}^n f_i(x)$ be an L_f-smooth and $$-strongly convex function. Suppose that each f_i is L-smooth and convex. Let $r := \frac{8\ 76+168\ \left(\frac{p_{max}}{p_{min}}\right)^2 \frac{m}{n}}{3}$. For any partition of the n functions to m machines, after

$$k = mp_{min}\quad 4n + 2r\frac{L}{} + 2^p r\frac{\sqrt{mL_f L}}{}\ !\ \log\left(\frac{1 + \frac{1}{2m}\quad f(x^0) \quad f(x\ ) + \frac{n}{2L}^2}{}\right)$$

iterations of Algorithm 3 with $= \frac{p}{2rL + 2\sqrt{rmL_f L}}$, we have $E\ f(x^k) \quad f(x\ )\ \ ;$ where $^2 = \frac{1}{n}\sum_i jr\ f_i(x\ )j_2^2$.

Remark C.2. Due to the strong convexity, we have $jx \quad x\ j_2^2 \quad \frac{2}{}(f(x) \quad f(x\ ))$, so this theorem also implies that $jx^k \quad x\ j_2^2$ after $O\left(n + \frac{L}{} + \frac{\sqrt{mL_f L}}{}\right)\log(1/\ )$ iterations.

We begin by establishing notation and reviewing the update performed at each step of Algorithm 3.

Recall that x is the value held at the PS, and let y denote $x \quad x\ $. Let G and H be the matrices whose jth column contains the vector g_j and h_j respectively. For all the variables in Algorithm 3 and discussed above, we use a superscript k to denote their value at the beginning of iteration k. When the iteration k is clear from context, we will eliminate the superscripts k. To further simplify, we will use the following definitions: $_i := \ _i \quad r\ f_i(x\ )$,

$_j := _j \; r \; f_{i_j}(x\ )$, and $g_j := g_j \; r \; f_{i_j}(x\ )$, where i_j is the index of the function used by machine j to compute g_j and $_j$, as indicated in Algorithm 3.

Recall that we analyze the expectation of the following potential function $(x; G; H; U; \ ; \ \ )$:

$$(x; G; H; U; \ ; \ \ ) := \sum_{`=1}^{5} \ `(x; G; H; U; \ ; \ \ );$$

where

$$\begin{aligned}
_1(x; G; H; U; \ ; \ \ ) &:= c_1(f(x) \quad f(x\ ));\\
_2(x; G; H; U; \ ; \ \ ) &:= \begin{matrix} x & x \end{matrix}^T \begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix} \begin{matrix} x & x \end{matrix} \ ;\\
_3(x; G; H; U; \ ; \ \ ) &:= \ ^2c_3 \sum_{0^j} \frac{p_{min}}{p_j} j g_j \ j_2^2;\\
_4(x; G; H; U; \ ; \ \ ) &:= \ ^2c_4 \ @2 \sum_i \frac{p_{min}}{p_j} j \ _i j_2^2 \quad \sum_j \frac{p_{min}}{p_j} j \ _j j_2^2 A \ ;\\
_5(x; G; H; U; \ ; \ \ ) &:= \ ^2c_5 \sum_j j u_j j_2^2:
\end{aligned}$$

Above, we abbreviate the $2d \quad 2d$ matrix $\begin{matrix} I_d & I_d \\ I_d & 2I_d \end{matrix}$ as $\begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix}$. Later, we will choose the $c_\cdot$ as follows:

$$\begin{aligned}
c_1 &= 4m \ ;\\
c_3 &= \ 64 + 168\frac{m}{n} \ \left(\frac{p_{max}}{p_{min}}\right)^{2!} \ c_5\\
c_4 &= \ 22 + \frac{76m}{n} \ \left(\frac{p_{max}}{p_{min}}\right)^{2!} \ c_5;\\
c_5 &= \frac{4}{3}(4mL_f \ + 4) :
\end{aligned}$$

The following proposition is our main technical proposition.

Proposition B.2. In Algorithm 3, for any step size $\frac{p^1}{2rL + 2\sqrt{rmL_f L}}$,

$$E_{i;j}[\ (k+1)] \quad (1 \quad \ ) \ (k)$$

where $= mp_{min} \min\left\{\frac{1}{4n}; \ \ \right\}$, and r is a constant defined in Theorem B.1 dependent only on $\frac{p_{max}}{p_{min}}$.

Before we prove this proposition, we prove Theorem B.1, which follows from Proposition B.2.

Proof. (Theorem B.1) Upon initialization, the expected potential $E[\ (0)]$ (over the random choices of i_j) equals

$$\begin{aligned}
E[\ (0)] &= 4m \ (f(x^0) \quad f(x\ )) + \ jx^0 \quad x\ j_2^2 + \ ^2(m(c_3 \quad c_4) + 2nc_4)\frac{1}{n}\sum_i \ jr \ f_i(x\ )j_2^2\\
&\quad 4m \ + \frac{2}{} \quad f(x^0) \quad f(x\ ) \ + \ ^2(c_3 + c_4)\sum_i \ jr \ f_i(x\ )j_2^2\\
&\quad 4m \ + \frac{2}{} \quad f(x^0) \quad f(x\ ) \ + \ ^2n(c_3 + c_4) \ ^2;
\end{aligned}$$

where the second line uses -strong convexity, the fact that $c_3 > c_4$ and that $m \quad n$. For $= \frac{1}{2rL + 2\sqrt{rmL_f L}}$, we use the fact that $\frac{1}{2rL}$ since $\frac{c_3 + c_4}{2r} \quad mL_f + 1 \quad m$; thus

$$E[\ (0)] \quad 4m\ +\ \frac{2}{-}\quad f(x^0) \quad f(x\ )\ +\ \frac{n\,(c_3 + c_4)}{2rL} \quad ^2$$
$$4m\ +\ \frac{2}{-}\quad f(x^0) \quad f(x\ )\ +\ \frac{2\,mn}{L} \quad ^2;$$

where we have plugged in the de nition of r.

With $= mp_{min} \min \frac{1}{4n}$; as in Proposition B.2 and and k as in Theorem B.1, we have

$$k \quad \log\left(\frac{1}{1 + \frac{1}{2m} \quad f(x^0) \quad f(x\ )\ +\ \frac{n}{2L}^2}\right);$$

and so

$$(1 \quad)^k \quad \exp(\ k\ ) \quad \exp\left(\ \log\left(\frac{1}{1 + \frac{1}{2m} \quad f(x^0) \quad f(x\ )\ +\ \frac{n}{2L}^2}\right)\right)$$
$$\frac{4m}{4m\ +\ \frac{2}{-}\ (f(x^0) \quad f(x\ ))\ +\ \frac{2\,mn}{L}^2} \quad \frac{4m}{E[\ (0)]}.$$

It follows from Proposition B.2 that

$$E[\ (k)] \quad 4m$$

Since $(k) \quad 4m\ (f(x^k) \quad f(x\ ))$, we have $E[f(x^k) \quad f(x\ )] \quad$ as desired. □

Proof. (Proposition B.2) To abbreviate, let $M = \begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix}$. We also abbreviate $r_i := r\ f_i(x^k)$, and let r be the matrix whose ith column is r_i. All expectations are over the random choice of $j \quad P$ and $i \quad \text{Uniform}(S_j)$ in Algorithm 3. Because, each function i only belongs to a single machine $j(i)$, we will sometime abbreviate the choice of $i; j$ in each iteration as just a choice of i; when we do so, any otherwise unspeci ed use of j should be interpreted as the machine $j(i)$ where f_i is stored.

Recall that in each iteration, given the random choice of j and i in that iteration, the following updates to the variables in Algorithm 3 are made, with $_j = \frac{p_{min}}{p_j}$:

$$\begin{aligned}
x^{k+1} \quad & x^k \quad _j(u_j^k + -^k); \\
-^{k+1} \quad & -^k + \frac{1}{n}h_j^k; \\
{}^{k+1}_{i_j} \quad & g_j^k; \\
g_j^{k+1} \quad & r\ f_i(x^k); \\
{}^{k+1}_{j} \quad & \begin{cases} {}^k_i & i \in i_j; \\ g_j^k & i = i_j; \end{cases} \\
h_j^{k+1} \quad & g_j^{k+1} \quad {}^{k+1}_j = \begin{cases} r\ f_i(x^k) \quad {}^k_i & i \in i_j; \\ r\ f_i(x^k) \quad g_j^k & i = i_j: \end{cases} \\
u_j^{k+1} \quad & u_j^k \quad 1 \quad \frac{p_{min}}{p_j} + \frac{p_{min}}{p_j}h_j^{k+1} \quad \frac{m}{n} \quad 1 \quad \frac{p_{min}}{p_j} \quad h_j^k \\
i_j^{k+1} \quad & i:
\end{aligned} \tag{?}$$

Note that the update to u_j contains a reference to both h_j^{k+1} and h_j^k. At the end of each iteration, we maintain the invariant $\sum_{i_j}^{k+1} = \sum_j^{k+1}$, and $\bar{m}^{k+1} = \frac{1}{n}\sum_i \sum_i^{k+1}$ because

$$h_j^k = g_j^k \quad \sum_j^k = \sum_{i_j}^{k+1} \quad \sum_{i_j}^k :$$

Dropping the superscripts of k, let

$$\Delta_i := \frac{x^{k+1}}{(U^{k+1}1 + m\bar{m}^{k+1})} \frac{x}{(U1 + m\bar{m})}$$

be the change to the vector $\frac{x}{U1 + m\bar{m}}$ if function i is chosen at iteration k. We begin by computing Δ_i.

Claim C.3.

$$\Delta_i = \sum_j \quad u_j + r\Delta_i \quad \sum_i + \frac{u_j}{} \bar{} \; I(i = i_j + \tfrac{m}{n}) \; h_j :$$

Proof. If $i \neq i_j$, then we have

$$U^{k+1}1 = U1 + \frac{p_{min}}{p_j}(u_j + r f_i(x) \quad \sum_i) \quad \frac{m}{n} \quad 1 \quad \frac{p_{min}}{p_j} \quad h_j :$$

Otherwise if $i = i_j$, then

$$\begin{aligned} U^{k+1}1 &= U1 + \frac{p_{min}}{p_j}(u_j + r f_i(x) \quad g_j) \quad \frac{m}{n} \quad 1 \quad \frac{p_{min}}{p_j} \quad h_j \\ &= U1 + \frac{p_{min}}{p_j}(u_j + r f_i(x) \quad \sum_i \quad g_j + \sum_j) \quad \frac{m}{n} \quad 1 \quad \frac{p_{min}}{p_j} \quad h_j \\ &= U1 + \frac{p_{min}}{p_j}(u_j + r f_i(x) \quad \sum_i \quad h_j) \quad \frac{m}{n} \quad 1 \quad \frac{p_{min}}{p_j} \quad h_j \end{aligned}$$

where the second line follows because $\sum_i = \sum_j$ in this case. In both cases, we have

$$m\bar{m}^{k+1} = m\bar{m} + \frac{m}{n} h_j :$$

Putting this all together with the update to x and the fact that $\sum_j = \frac{p_{min}}{p_j}$ yields the claim:

$$\begin{aligned} \Delta_i &= \left(\frac{p_{min}}{p_j}(u_j + r\Delta_i \quad \sum_i) + \frac{m}{n} \quad 1 \quad \frac{p_{min}}{p_j} \quad \frac{p_{min}}{p_j} I(i = i_j) + \frac{m}{n} \quad h_j \right)^{\sum_j (u_j + \bar{})} \\ &= \frac{p_{min}}{p_j} \quad u_j + r\Delta_i \quad \sum_i + I(i = i_j) + \frac{m}{n} \quad h_j \Big)^{\sum_j (u_j + \bar{})} \\ &= \sum_j \quad u_j + r\Delta_i \quad \sum_i + I(i = i_j) + \frac{m}{n} \quad h_j \Big)^{\sum_j (u_j + \bar{})} \end{aligned}$$

□

Computing the expectation of Δ_i and plugging in $\sum_j = \frac{p_{min}}{p_j}$ yields the Unbiased Trajectory lemma:

Lemma B.3 (Unbiased Trajectory).

$$E_{i;j}[\Delta_i] = p_{min} \quad \frac{U1 + m\bar{}}{U1 + m\bar{}} \quad \frac{m}{n} r \; 1 :$$

Proof. First note that $I(i = i_j)$ is independent from j, and occurs with probability $\frac{1}{|S_j|} = \frac{m}{n}$. Hence, from Claim C.3, we have

$$\begin{aligned} E_{i,j}[\ _i] &= E_{i,j}\ _j \quad u_j + r_i \quad _i + \quad I(i = i_j + \tfrac{m}{n})\ h_j \\ &= \sum_{j,i \in S_j} \frac{m}{n} p_j\ _j \quad u_j + r_i \quad _i + \quad I(i = i_j + \tfrac{m}{n})\ h_j \\ &= p_{min} \quad \begin{matrix} U1 & m^- \\ U1 + \frac{m}{n} r & 1 \quad m^- \end{matrix} : \end{aligned}$$

□

Now

$$_2(k+1) = \quad \begin{matrix} y \\ (U1 + m^-) \end{matrix} + \ _i^T \quad \begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix} \quad \begin{matrix} y \\ (U1 + m^-) \end{matrix} + \ _i ;$$

so we can compute the difference

$$\begin{aligned} E[\ _2(k+1)] \quad _2(k) &= 2 \quad \begin{matrix} y \\ (U1 + m^-) \end{matrix}^T \quad \begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix} \ E_{i,j}[\ _i] + E_{i,j} \quad _i^T M \ _i \\ &= 2\, p_{min} \quad \begin{matrix} y \\ (U1 + m^-) \end{matrix}^T \quad \begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix} \quad \begin{matrix} U1 + m^- \\ U1 + m^- \ \frac{m}{n} r\ 1 \end{matrix} + E_{i,j} \quad _i^T M \ _i \\ &= 2p_{min} \quad \begin{matrix} y \\ (U1 + m^-) \end{matrix}^T \quad \begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix} \quad \begin{matrix} y \\ (U1 + m^-) \end{matrix} + E_{i,j} \quad _i^T M \ _i \\ &\quad \frac{2\, m p_{min}}{n} y^T r\ 1 + \frac{4\ ^2 m p_{min}}{n} (U1 + m^-)^T r\ 1: \end{aligned} \tag{4}$$

We bound the quadratic term in the difference $E[\ _2(k)] \quad _2(k)$ in (4) in the following claim.

Claim C.4.

$$E_{i,j} \quad _i^T M \ _i \quad \frac{4 m p_{min}}{n}^2 \Big(4 \sum_j |g_j|_2^2 + 4 \sum_j |\ _j|_2^2 + \frac{n}{m} \sum_j |u_j|_2^2 + 2 \sum_i |\ _i|_2^2 + 2 \sum_i |r_i \quad r\ _i(x\ )|_2^2 \Big) :$$

Proof.

$$
\begin{aligned}
&\frac{1}{2}E_{i;j}\left[\nabla_i^T M \nabla_i\right] \\
&= E_{i;j}\left[\frac{\|j\|_2^2}{2}\left(u_j - r_i + \eta_i + \frac{m}{n} + \mathbb{I}(i = i_j)\right)h_j\right]^T M \left(u_j - r_i + \eta_i + \frac{m}{n} + \mathbb{I}(i = i_j)\right)h_j\Bigg] \\
&\quad \frac{mp_{\min}}{n}\sum_i \left[\left(u_j - r_i + \eta_i + \frac{m}{n} + \mathbb{I}(i = i_j)\right)h_j\right]^T M \left(u_j - r_i + \eta_i + \frac{m}{n} + \mathbb{I}(i = i_j)\right)h_j\Bigg] \\
&\quad \frac{4mp_{\min}}{n}\left(1 - \frac{m}{n}\right)\sum_i \left[\left(0 - \frac{m}{n}h_j\right)^T M \left(0 - \frac{m}{n}h_j\right)\right] + \frac{4mp_{\min}}{n}\frac{m}{n}\sum_i \left[\left(0 - \left(1 - \frac{m}{n}\right)h_j\right)^T M \left(0 - \left(1 - \frac{m}{n}\right)h_j\right)\right] \\
&\quad + \frac{4mp_{\min}}{n}\sum_i \left[\left(u_j - u_j\right)^T M \left(u_j - u_j\right)\right] \\
&\quad + \frac{4mp_{\min}}{n}\sum_i \left[\left(-\eta_i r_i(x) - \eta_i r_i(x)\right)^T M \left(-\eta_i r_i(x) - \eta_i r_i(x)\right)\right] \\
&\quad + \frac{4mp_{\min}}{n}\sum_i \left[\left(0 - r_i r_i(x)\right)^T M \left(0 - r_i r_i(x)\right)\right].
\end{aligned}
\tag{5}
$$

Here the first inequality is by the fact the definition $_j = \frac{p_{\min}}{p_j}$, and the second is by Jensen's inequality and the fact that the distribution of j conditioned on the indicator $\mathbb{I}(i = i_j)$ is equivalent to the distribution of j when i is chosen uniformly at random.

We can bound each of these terms by plugging in $M = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$.

For the first two terms involving h_j, we have,

$$
\begin{aligned}
&\left(1 - \frac{m}{n}\right)\sum_i \left[\left(0 - \frac{m}{n}h_j\right)^T M \left(0 - \frac{m}{n}h_j\right)\right] + \frac{m}{n}\sum_i \left[\left(1 - \frac{m}{n}\right)h_j\right]^T M \left(1 - \frac{m}{n}\right)h_j\Bigg] \\
&= 2\left(\frac{m}{n} - \frac{m^2}{n^2}\right)\sum_i \|h_j\|_2^2 \\
&= 2\left(1 - \frac{m}{n}\right)\sum_j \|h_j\|_2^2 \\
&\quad 4\sum_j \|g_j\|_2^2 + 4\sum_j \|_j\|_2^2.
\end{aligned}
\tag{6}
$$

Similarly,

$$
\sum_i \left[\left(u_j - u_j\right)^T M \left(u_j - u_j\right)\right] = \sum_i \|u_j\|_2^2 = \frac{n}{m}\sum_j \|u_j\|_2^2. \tag{7}
$$

For the final two terms, we have

$$
\begin{aligned}
&\sum_i \left[\left(-\eta_i r_i(x) - \eta_i r_i(x)\right)^T M \left(-\eta_i r_i(x) - \eta_i r_i(x)\right)\right] + \sum_i \left[\left(0 - r_i r_i(x)\right)^T M \left(0 - r_i r_i(x)\right)\right] \\
&\quad \sum_i \left[\left(-\eta_i r_i(x) - \eta_i r_i(x)\right)^T M \left(-\eta_i r_i(x) - \eta_i r_i(x)\right)\right] + \sum_i \left[\left(0 - r_i r_i(x)\right)^T M \left(0 - r_i r_i(x)\right)\right] \\
&= 2\sum_i \|_i\|_2^2 \sum_i \|-\|_2^2 + 2\sum_i \|r_i r_i(x)\|_2^2 \\
&\quad \frac{n}{m}\sum_j \|u_j\|_2^2 + 2\sum_i \|_i\|_2^2 + 2\sum_i \|r_i r_i(x)\|_2^2.
\end{aligned}
\tag{8}
$$

Plugging these three equations into Equation 5 yields the claim.

□

Next we bound the expected change in $_1$.

Claim C.5.

$$E[\ _1(k+1)] \quad _1(k) \quad \frac{c_1\ p_{min}}{n}(U1 + m^{-})^T\ r\ 1 + c_1 L_f\ ^2 m p_{min}\ @\frac{1}{m}\sum_j ju_j j_2^2 + \frac{1}{n}\sum_i j\ _i j_2^2 A$$

Proof. Using convexity and L_f-smoothness,

$$\begin{aligned} E[\ _1(k+1)] \quad (k) &= c_1 E_{i,j}\ [f(x \quad _j(u_j + ^{-})] \quad c_1 f(x) \\ &\quad c_1 E[\ \ _j(u_j + ^{-})^T]\frac{r\ 1}{n} + \frac{c_1 L_f}{2} E[j\ _j(u_j + ^{-})j_2^2] \\ &= \frac{c_1\ p_{min}}{n}(U1 + m^{-})^T\ r\ 1 + \frac{c_1 L_f}{2}\sum_j p_j\ _j^2 ju_j + ^{-}j_2^2 \\ &\quad \frac{c_1\ p_{min}}{n}(U1 + m^{-})^T\ r\ 1 + c_1 L_f \sum_j p_j\ _j^2 ju_j j_2^2 + c_1 L_f\ ^2 m p_{min} j^{-}j^2 \\ &\quad \frac{c_1\ p_{min}}{n}(U1 + m^{-})^T\ r\ 1 + c_1 L_f\ ^2 p_{min}\sum_j ju_j j_2^2 + \frac{c_1 L_f\ ^2 m p_{min}}{n}\sum_i j\ _i j^2: \end{aligned}$$

Here we used Jensen's in the second inequality, and the fact that $^{-} = \frac{1}{n}\sum_i\ _i[\ _i + r\ _i(x\ )]$, so because $\frac{1}{n}\sum_i r\ _i(x\ ) = 0$, we have $j^{-}j^2 \quad \frac{1}{n}\sum_i j\ _i j_2^2$ in the third inequality. □

We now combine (4), Claim C.4, and Claim C.5, plugging in our choice of $c_1 = 4m$, which was chosen so that the $(U1 + m^{-})^T r\ 1$ terms cancel. This yields the following lemma.

Lemma B.4. (Formal)

$$\begin{aligned} E[\ _1(k+1) + \ _2(k+1)] \quad (\ _1(k) + \ _2(k)) \\ 2p_{min}\ \begin{pmatrix} y \\ (U1 + m^{-}) \end{pmatrix}^T \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y \\ (U1 + m^{-}) \end{pmatrix} \quad \frac{2\ m p_{min}}{n}\ y^T r\ 1 \\ + \sum_j ju_j j_2^2\ ^2 p_{min}\ (4mL_f\ + 4) \\ + \sum_j jg_j j_2^2\ \ \frac{16 m p_{min}\ ^2}{n} \\ + \sum_j j\ _j j_2^2\ \ \frac{16 m p_{min}\ ^2}{n} \\ + \sum_i j\ _i j_2^2 \frac{^2 m p_{min}}{n}(4m L_f\ + 16) \\ + \sum_i jr\ _i\ \ r\ _i(x\ )j_2^2\ \ \frac{8 m p_{min}\ ^2}{n}\ : \end{aligned}$$

We proceed in the following claims to bound the differences in expectation of $_3$ and $_4$.

Claim C.6.

$$E[\ _3(k+1)] \quad _3(k) \quad ^2 c_3 p_{min}\sum_j jg_j j_2^2\ + \ ^2 c_3 p_{min}\frac{m}{n}\sum_i jr\ _i\ \ r\ _i(x\ )j_2^2:$$

Proof. If function i is chosen in iteration k, then g_j becomes r_i, so

$$\Psi_3(k+1) = {}^2c_3 \Bigg(\sum_{j \neq i} \frac{p_{min}}{p_j} \| g_j - \nabla_j \|_2^2 + \frac{p_{min}}{p_i} \| r_i - \nabla_i(x) \|_2^2 \Bigg).$$

Taking the expectation over i yields the claim. □

Claim C.7.

$$\begin{aligned} E_{i,j}[\Psi_4(k+1)] - \Psi_4(k) \leq & -\frac{p_{min}\, m}{4n} \Psi_4(k) - {}^2\frac{m p_{min}\, c_4}{2n} \sum_i \| \nabla_i - x_i \|_2^2 \\ & - {}^2\frac{p_{min}\, c_4}{4} \sum_j \| \nabla_j - x_j \|_2^2 \\ & + {}^2 2 p_{min}\, c_4 \sum_j \| g_j - x_j \|_2^2. \end{aligned}$$

Proof. If function i is chosen in the kth iteration, then

$$\begin{aligned} \Psi_4(k+1) - \Psi_4(k) = & -{}^2c_4 \frac{p_{min}}{p_i} \| g_i - x_i \|_2^2 - \| \nabla_i - x_i \|_2^2 \\ & + {}^2c_4 \frac{p_{min}}{p_i} \| \nabla_i - x_i \|_2^2 - \| \nabla_i - x_i \|_2^2 + \mathbb{I}(i = i_j)(\| \nabla_i - x_i \|_2^2 - \| g_i - x_i \|_2^2) \\ = & -{}^2c_4 \frac{p_{min}}{p_i} \| \nabla_i - x_i \|_2^2 + \| \nabla_i - x_i \|_2^2 - {}^2c_4 \frac{p_{min}}{p_i} \| g_i - x_i \|_2^2 \\ & + {}^2c_4 \frac{p_{min}}{p_i} \mathbb{I}(i = i_j) \| \nabla_i - x_i \|_2^2 - \| g_i - x_i \|_2^2 ; \end{aligned}$$

where we have plugged in $\nabla_{i_j} = \nabla_j$, and in the case when $i = i_j$, the equality $\nabla_i = \nabla_j$. Taking the expectation

over i yields

$$
\begin{aligned}
E_{i;j}[\ _4(k+1)] \quad _4(k) &= {}^2c_4 p_{\min} \left(\frac{m}{n}\sum_i j\ _i j_2^2 + \sum_j j\ _j j_2^2\right) \\
&\quad + {}^2 2c_4 p_{\min} \sum_j j g_j j_2^2 \\
&\quad + {}^2c_4 p_{\min} \frac{m}{n}\left(\sum_j j\ _j j_2^2 \sum_j j g_j j_2^2\right) \\
&= {}^2c_4 p_{\min} \frac{m}{n}\sum_i j\ _i j_2^2 \\
&\quad {}^2c_4 p_{\min}\ 1\ \frac{m}{n}\sum_j j\ _j j_2^2 \\
&\quad + {}^2c_4 p_{\min}\ 2\ \frac{m}{n}\sum_j j g_j j_2^2 \\
&= \frac{{}^2c_4 p_{\min} m}{4n}\left(2\sum_i j\ _i j_2^2 \sum_j j\ _j j_2^2\right) \\
&\quad {}^2c_4 \frac{m p_{\min}}{n}\ 1\ \frac{2}{4}\sum_i j\ _i j_2^2 \\
&\quad {}^2c_4\ 1\ \frac{3m}{4n}\sum_j j\ _j j_2^2 \\
&\quad + {}^2c_4 p_{\min}\ 2\ \frac{m}{n}\sum_j j g_j j_2^2 \\
&\quad \frac{p_{\min} m}{4n}\ _4(k) \\
&\quad {}^2c_4 \frac{m p_{\min}}{2n}\sum_i j\ _i j_2^2 \\
&\quad {}^2c_4 p_{\min}\ 1\ \frac{3m}{4n}\sum_j j\ _j j_2^2 \\
&\quad + {}^2c_4 p_{\min}\ 2\ \frac{m}{n}\sum_j j g_j j_2^2 \\
&\quad \frac{p_{\min} m}{4n}\ _4(k) \quad {}^2c_4 \frac{m p_{\min}}{2n}\sum_i j\ _i j_2^2 \\
&\quad {}^2\frac{c_4 p_{\min}}{4}\sum_j j\ _j j_2^2 \\
&\quad + {}^2 2p_{\min} c_4 \sum_j j g_j j_2^2;
\end{aligned}
$$

where the first inequality follows from the fact that $\frac{p_{\min}}{p_j} \quad p_j$ for all j, and the second inequality follows from bounding $0 \quad \frac{m}{n} \quad 1$. □

Claim C.8.

$$
E[\ _5(k+1)] \quad _5(k) \quad c_5\ {}^2 4\ p_{\min} \sum_j j u_j j^2 + \frac{4m p_{\min}}{n}\sum_i j r_i \quad r\ _i(x\ )j_2^2 + j\ _i j_2^2 \ + \frac{16m}{n p_{\min}}\sum_j p_j^2\ j g_j j_2^2 + j\ _j j_2^2\ 5 :
$$

Proof. If function i is chosen in iteration k, then u_j becomes

$$u_j \quad 1 \quad \frac{p_{min}}{p_j} \quad + \frac{p_{min}}{p_j}(r_{\ i} \quad _i) \quad \frac{m}{n} + \frac{p_{min}}{p_j} \quad I(i = i_j) \quad \frac{m}{n} \quad h_j :$$

Hence in this case, by applying Jensen's inequality, we have

$$\begin{aligned}
{}_5(k+1) \quad {}_5(k) &= c_5^{\ 2} u_j \quad 1 \quad \frac{p_{min}}{p_j} \quad + \frac{p_{min}}{p_j}(r_{\ i} \quad _i) \quad \frac{m}{n} + \frac{p_{min}}{p_j} \quad I(i = i_j) \quad \frac{m}{n} \quad h_j \quad ^2 \quad c_5^{\ 2} j u_j j^2 \\
&\quad \# \\
&\quad c_5^{\ 2} \frac{1}{1 \quad \frac{p_{min}}{p_j}} u_j \quad 1 \quad \frac{p_{min}}{p_j} \quad ^2 + \frac{1}{\frac{p_{min}}{p_j}} \frac{p_{min}}{p_j}(r_{\ i} \quad _i) \quad \frac{m}{n} + \frac{p_{min}}{p_j} \quad I(i = i_j) \quad \frac{m}{n} \quad h_j \quad ^2 \quad j \ u_j j^2 \\
&\quad \# \\
&= c_5^{\ 2} \quad \frac{p_{min}}{p_j} j u_j j^2 + \frac{p_j}{p_{min}} \frac{p_{min}}{p_j}(r_{\ i} \quad _i) \quad \frac{m}{n} + \frac{p_{min}}{p_j} \quad I(i = i_j) \quad \frac{m}{n} \quad h_j \quad ^2 :
\end{aligned}$$

Now we use Jensen's inequality and the fact that $\frac{p_{min}}{p_j} \quad 1$ to break up the second squared term:

$$\begin{aligned}
&\frac{p_{min}}{p_j}(r_{\ i} \quad _i) \quad \frac{m}{n} + \frac{p_{min}}{p_j} \quad I(i = i_j) \quad \frac{m}{n} \quad h_j \quad ^2 \\
&\quad 4\frac{p_{min}^2}{p_j^2} \quad j r_{\ i} \quad r_{\ i}(x \ )j_2^2 + j \ _i j_2^2 \quad + 2 \quad \frac{m}{n} + \frac{p_{min}}{p_j} \quad I(i = i_j) \quad \frac{m}{n} \quad ^2 j h_j j^2 \\
&\quad 4\frac{p_{min}^2}{p_j^2} \quad j r_{\ i} \quad r_{\ i}(x \ )j_2^2 + j \ _i j_2^2 \quad + 2 \quad \frac{m}{n} + \frac{p_{min}}{p_j} I(i = i_j) \quad ^2 j h_j j^2 \\
&\quad 4\frac{p_{min}^2}{p_j^2} \quad j r_{\ i} \quad r_{\ i}(x \ )j_2^2 + j \ _i j_2^2 \quad + 4 \quad \frac{m}{n} + \frac{p_{min}}{p_j} I(i = i_j) \quad ^2 j g_j j_2^2 + 4 \quad \frac{m}{n} + \frac{p_{min}}{p_j} I(i = i_j) \quad ^2 j \ _j j_2^2 :
\end{aligned} \tag{9}$$

Observe that for a xed j,

$$\sum_{i:j(i)=j} \quad \frac{m}{n} + \frac{p_{min}}{p_j} I(i = i_j) \quad ^2 \quad \sum_{i:j(i)=j} \quad \frac{m}{n} + I(i = i_j) \quad ^2 = \quad \frac{m}{n} + 1 \quad ^2 + \quad \frac{n}{m} \quad 1 \quad \frac{m}{n} \quad ^2 \quad 4:$$

Finally, taking the expectation over i, we have

$$\begin{aligned}
E[\ {}_5(k+1)] \quad {}_5(k) &\\
&c_5^{\ 2} 4 \quad p_{min} \sum_j j u_j j^2 + \frac{4mp_{min}}{n} \sum_i j r_{\ i} \quad r_{\ i}(x \ )j_2^2 + j \ _i j_2^2 + \sum_j \frac{16mp_j^2}{np_{min}} \quad j g_j j_2^2 + j \ _j j_2^2 \quad 5
\end{aligned} \tag{10}$$

□

Combining Claim C.6, Claim C.7, and Claim C.8 yields the following lemma.

Lemma B.5. (Formal)

$$\begin{aligned}
E[\ {}_3(k+1) + \quad {}_4(k+1) + \quad {}_5(k+1)] \quad & {}_3(k) \quad {}_4(k) \quad {}_5(k) \\
& \frac{p_{min} m}{4n}(\ {}_3(k) + \quad {}_4(k) + \quad {}_5(k)) + \quad ^2 \frac{mp_{min}}{n}(c_3 + 4c_5) \sum_i j r_{\ i} \quad r_{\ i}(x \ )j_2^2 \\
& \quad ^2 \frac{m}{n} \sum_i (p_{min}(4mL_f \quad + 16)) j \ _i j_2^2 \quad ^2 \sum_j (4mL_f \quad + 4) j u_j j_2^2 \\
& \quad ^2 \sum_j \quad \frac{16mp_{min}}{n} \quad j g_j j_2^2 + j \ _j j_2^2 :
\end{aligned}$$

Proof.

$$
\begin{aligned}
&E[\ _3(k+1) + \ _4(k+1) + \ _5(k+1)] \quad \ _3(k+1) \quad \ _4(k+1) \quad \ _5(k+1) \\
&\quad {}^2c_3 p_{\min} \sum_j |g_j|_2^2 + {}^2c_3 \frac{mp_{\min}}{n} \sum_i |r_i - r_i(x)|_2^2 \\
&\quad \frac{p_{\min} m}{4n} \ _4(k) \quad {}^2\frac{mp_{\min} c_4}{2n} \sum_i |_i|_2^2 \quad {}^2\frac{p_{\min} c_4}{4} \sum_j |_j|_2^2 + {}^2 2p_{\min} c_4 \sum_j |g_j|_2^2 \\
&\quad + c_5 \ {}^2 4 \ p_{\min} \sum_j |u_j|^2 + \frac{4mp_{\min} c_5}{n} \sum_i |r_i - r_i(x)|_2^2 + |_i|_2^2 + \sum_j \frac{16mp_j^2}{np_{\min}} |g_j|_2^2 + |_j|_2^2 \ 5 \\
= \ & \frac{p_{\min} m}{4n} \ _4(k) \\
&\quad + {}^2\frac{m}{n} \sum_i \quad \frac{p_{\min} c_4}{2} + 4 p_{\min} c_5 \quad |_i|_2^2 \\
&\quad + {}^2 \sum_j \quad p_{\min} c_3 + 2 p_{\min} c_4 + \frac{16mp_j^2 c_5}{np_{\min}} \quad |g_j|_2^2 \\
&\quad + {}^2 \sum_j \quad \frac{p_{\min} c_4}{4} + \frac{16mp_j^2 c_5}{np_{\min}} \quad |_j|_2^2 \\
&\quad + {}^2 \sum_j \quad (\ c_5 p_{\min}) |u_j|_2^2 \\
&\quad + {}^2 \quad \frac{mp_{\min}}{n} c_3 + \frac{4mp_{\min}}{n} c_5 \quad \sum_i |r_i - r_i(x)|_2^2 \\
= \ & \frac{p_{\min} m}{4n} (\ _3(k) + \ _4(k) + \ _5(k)) \\
&\quad + {}^2\frac{m}{n} \sum_i \quad \frac{p_{\min} c_4}{2} + 4 p_{\min} c_5 \quad |_i|_2^2 \\
&\quad + {}^2 \sum_j \quad p_{\min} c_3 + 2 p_{\min} c_4 + \frac{16mp_j^2 c_5}{np_{\min}} + \frac{p_{\min}^2 mc_3}{4np_j} \quad |g_j|_2^2 \\
&\quad + {}^2 \sum_j \quad \frac{p_{\min} c_4}{4} + \frac{16mp_j^2 c_5}{np_{\min}} \quad |_j|_2^2 \\
&\quad + {}^2 \sum_j \quad c_5 p_{\min} + \frac{p_{\min} mc_5}{4n} \quad |u_j|_2^2 \\
&\quad + {}^2 \quad \frac{mp_{\min}}{n} c_3 + \frac{4mp_{\min}}{n} c_5 \quad \sum_i |r_i - r_i(x)|_2^2 \\
&\frac{p_{\min} m}{4n} (\ _3(k) + \ _4(k) + \ _5(k)) \\
&\quad {}^2\frac{m}{n} \sum_i \quad (p_{\min} (4mL_f \quad + 16)) |_i|_2^2 \\
&\quad {}^2 \sum_j \quad \frac{16mp_{\min}}{n} \quad |g_j|_2^2 \\
&\quad {}^2 \sum_j \quad \frac{16mp_{\min}}{n} \quad |_j|_2^2 \\
&\quad {}^2 \sum_j \quad mp_{\min} \ (4mL_f \quad + 4) \ |u_j|_2^2 \\
&\quad + {}^2\frac{mp_{\min}}{n} (c_3 + 4 c_5) \sum_i |r_i - r_i(x)|_2^2 :
\end{aligned}
$$

Here the first inequality follows from Claims C.6, C.7, C.8, and the last inequality follows from plugging in our choice of constants: $c_5 = \frac{4}{3}(4mL_f + 4)$, $c_4 = 22 + \frac{76m}{n}\left(\frac{p_{max}}{p_{min}}\right)^2 c_5$, and $c_3 = 64 + 168\frac{m}{n}\left(\frac{p_{max}}{p_{min}}\right)^2 c_5$. □

We use the following lemma to bound the $\sum_i \|r_i - r_i(x^*)\|_2^2$ terms, which appear in Lemmas B.4 and B.5.

Lemma C.9.
$$\sum_i \|r_i - r_i(x^*)\|_2^2 \le Ly^T r - 1.$$

Proof. By the convexity of each f_i and their L-smoothness,
$$\sum_i \|r_i - r_i(x^*)\|_2^2 \le \sum_i Ly^T(r_i - r_i(x^*)) = Ly^T r - 1.$$
Above, we used the fact that $\sum_i r_i(x^*) = 0$. □

We now combine Lemma B.4 and Lemma B.5 to find the total expected difference in potential.

$$\begin{aligned}
E[\Phi(k+1)] - \Phi(k) &= (E[\Gamma_1(k+1) + \Gamma_2(k+1)] - (\Gamma_1(k) + \Gamma_2(k))) \\
&\quad + (E[\Gamma_3(k+1) + \Gamma_4(k+1) + \Gamma_5(k+1)] - (\Gamma_3(k) + \Gamma_4(k) + \Gamma_5(k))) \\
&\le -2p_{min}\left(\frac{y}{(U1 + m^-)}\right)^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \left(\frac{y}{(U1 + m^-)}\right) - \frac{2mp_{min}}{n} y^T r - 1 \\
&\quad + \frac{8mp_{min}^2}{n} \sum_i \|r_i - r_i(x^*)\|_2^2 \\
&\quad - \frac{mp_{min}}{4n}(\Gamma_3(k) + \Gamma_4(k) + \Gamma_5(k)) + \frac{2mp_{min}}{n}(c_3 + 4c_5) \sum_i \|r_i - r_i(x^*)\|_2^2 \\
&\le -2p_{min}\left(\frac{y}{(U1 + m^-)}\right)^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \left(\frac{y}{(U1 + m^-)}\right) \\
&\quad - \frac{mp_{min}}{4n}(\Gamma_3(k) + \Gamma_4(k) + \Gamma_5(k)) - \frac{mp_{min}\, Cy^T r - 1}{n},
\end{aligned}$$

where
$$C = 2^2 L (8 + c_3 + 4c_5);$$
and in the last inequality, we have used Lemma C.9.

Now because f is convex, $f(x) - f(x^*) \le \frac{y^T r - 1}{n}$, and so rearranging terms and plugging in the value of c_1, we have

$$\begin{aligned}
E[\Phi(k+1)] - \Phi(k) &\le -2p_{min}\left(\frac{y}{(U1 + m^-)}\right)^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \left(\frac{y}{(U1 + m^-)}\right) - \frac{mp_{min}}{4n}(\Gamma_3(k) + \Gamma_4(k) + \Gamma_5(k)) \\
&\quad - \frac{c_1 mp_{min}}{4n}(f(x) - f(x^*)) - mp_{min}\left(C - \frac{c_1}{4n}\right)\frac{y^T r - 1}{n} \\
&= -2p_{min}\left(\frac{y}{(U1 + m^-)}\right)^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \left(\frac{y}{(U1 + m^-)}\right) - \frac{mp_{min}}{4n}(\Gamma_1(k) + \Gamma_3(k) + \Gamma_4(k) + \Gamma_5(k)) \\
&\quad - mp_{min}\left(C - \frac{c_1}{4n}\right)\frac{y^T r - 1}{n}.
\end{aligned} \tag{11}$$

The next claim shows that for small enough γ, the final term in this equation is large.

Claim C.10. For $\gamma \le \frac{p_{min}}{2rL + 2^2 mL_f L}$,
$$C - \frac{c_1}{4n} \ge \frac{1}{2};$$

where $r = \frac{8 \quad 76+168 \quad \frac{p_{max}}{p_{min}} \quad ^2 \frac{m}{n}}{3}$ as in Theorem B.1.

Proof. First observe that

$$r \quad \frac{8 + \quad 68 + 168 \quad \frac{p_{max}}{p_{min}} \quad ^2 \frac{m}{n} \quad \frac{4}{3}(4mL_f \quad + 4)}{2(mL_f \quad + 1)} = \frac{L(8 + c_3 + 4c_5)}{2L(mL_f \quad + 1)}:$$

Thus

$$C \quad \frac{c_1}{4n} \quad 2 \quad 2 \quad ^2 r(mL_f \quad + 1)$$
$$= 2 \quad (1 \quad rL \quad ^2 rmL_f L)$$
$$:$$

for $\frac{p^1}{2rL + 2\sqrt{rmL_f L}}$. Here in the last line we have used the fact that $(1 \quad rL \quad ^2 rmL_f L)$ is increasing in , and for any $a; b > 0$, $1 \quad \frac{a}{2(a+b)} \quad \frac{b^2}{(2(a+b))^2} \quad \frac{1}{2}$ (we plugged in $a = rL$ and $b^2 = rmL_f L$).

□

Using the strong convexity of f, we have $y^T r \quad 1 \quad n \quad jyj_2^2$: Hence plugging Claim C.10 into (11) yields the following lemma.

Lemma B.6. For $< \frac{p^1}{2rL + 2\sqrt{rmL_f L}}$,

$$E[\ (k+1)] \quad (k) \quad 2p_{min} \quad \begin{matrix} y \\ (U1 + m^-) \end{matrix}^T \begin{matrix} \frac{m}{2} & 0 \\ 0 & 1 \end{matrix} \begin{matrix} y \\ (U1 + m^-) \end{matrix}$$
$$\frac{mp_{min}}{4n}(\ _1(k) + \ _3(k) + \ _4(k) + \ _5(k))$$

Recall that our goal is to nd some $\frac{mp_{min}}{4n}$ such that

$$E[\ (k+1)] \quad (1 \quad) \quad (k):$$

We will do this by nding some that satis es for all $y; U$ and $^-$:

$$\frac{\begin{matrix} y \\ (U1 + m^-) \end{matrix}^T \begin{matrix} \frac{m}{2} & 0 \\ 0 & 1 \end{matrix} \begin{matrix} y \\ (U1 + m^-) \end{matrix}}{\begin{matrix} y \\ (U1 + m^-) \end{matrix}^T \begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix} \begin{matrix} y \\ (U1 + m^-) \end{matrix}} \quad \frac{}{2p_{min}}; \tag{12}$$

or equivalently

$$Q \quad \frac{}{p_{min}} I;$$

where

$$Q := \begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix}^{1=2} \begin{matrix} m & 0 \\ 0 & 2 \end{matrix} \begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix}^{1=2}: \tag{13}$$

Indeed, establishing (12) will imply that $E[\ (k+1)] \quad (k) \quad _2(k) \quad \frac{mp_{min}}{4}(\ _1(k) + \ _3(k) + \ _4(k) + \ _5(k))$; so if $\frac{mp_{min}}{4n}$, then $E[\ (k)] \quad (1 \quad) \quad (k)$.

We bound the smallest eigenvalue of Q by evaluating the trace and determinant of the product of $2 \quad 2$ matrices that underlie the block matrices above in the matrix product forming Q.

Lemma C.11. For any symmetric $2 \quad 2$ matrix A, $_{min}(A) \quad \frac{Det(A)}{Tr(A)}$.

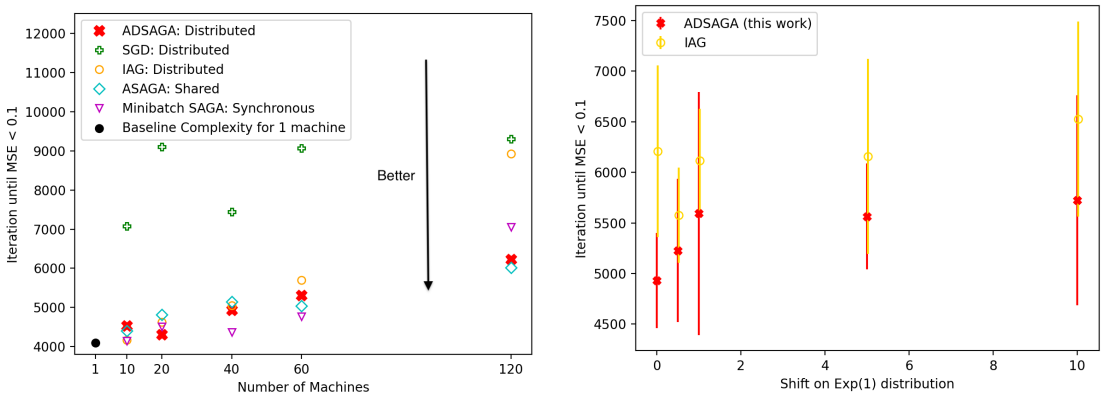


Figure 4: Comparison of ADSAGA (this work), ASAGA (Leblond et al., 2018), minibatch SAGA (Gazagnadou et al., 2019), SGD (Lian et al., 2018), and IAG (Gurbuzbalaban et al., 2017): Iteration complexity to achieve $|x_k - x^*|_2^2 \leq 0.1$, averaged over 8 runs. (b) Comparison of IAG and ADSAGA with shifted-exponential delays for 60 machines.

Proof. Let $d := \mathrm{Det}(A)$, and $t := \mathrm{Tr}(A)$. By the characteristic equation, putting

$$\lambda_{min}(A) = \frac{t - \sqrt{t^2 - 4d}}{2} = \frac{t}{2}\left(1 - \sqrt{1 - \frac{4d}{t^2}}\right) \geq \frac{t}{2}\left(1 - \left(1 - \frac{2d}{t^2}\right)\right) = \frac{d}{t},$$

where the inequality follows from the fact that $\sqrt{1+x} \leq 1 + \frac{x}{2}$ for $x \geq -1$. □

Claim C.12. *For* $\eta \leq \frac{1}{2rL + 2\sqrt{rmL_f L}}$,

$$\lambda_{min}(Q) \geq \min\left(1, \mu m \eta\right).$$

Proof. We compute the determinant and trace of Q. Note that $\det\left(\begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}\right) = 1$.

$$D := \mathrm{Det}(Q) = 2\mu m \eta$$

Using the circular law of trace, and computing the inverse $\begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}^{-1} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$, we have

$$\mathrm{Tr}(Q) = 2\mu n \eta + 2 = D + 2.$$

Now by Lemma C.11

$$\lambda_{min}(Q) \geq \frac{D}{D+2} \geq \min\left(1, \frac{D}{2}\right) = \min\left(1, \mu m \eta\right),$$

where the second inequality holds for any $D > 0$. □

Recalling our bound that $\gamma \leq \frac{m p_{\min}}{4n}$, this claim shows that we can choose $\gamma = m p_{\min} \min\left(\frac{1}{4n}, \mu\eta\right)$ as desired. □

D Extended Experiments

We conduct experiments to compare the convergence rates of ADSAGA to other state-of-the-art algorithms: SGD, IAG, ASAGA, and minibatch SAGA. In our first set of experiments, we simulate the stochastic delay model of Section 1.1. In our second set, we implement these algorithms in a distributed compute cluster. The main takeaways of our experiments are the following:

(a) (b)

Figure 5: (a) Convergence accuracy after 100 epochs. (b) Wallclock time to achieve $|x_k - x^*|_2^2 \leq 10^{-10}$, averaged over 8 runs.

1. In experiments on a cluster, ADSAGA is comparable to IAG and outperforms all other algorithms in wall-clock time.
2. In experiments with simulated (shifted) exponential delays, ADSAGA outperforms IAG in iteration complexity.
3. ADSAGA performs well even without knowledge of the update rates $\{p_j\}$, even with significant machine heterogeneity.

Data. For all experiments, we simulate these algorithms on a randomly-generated least squares problem $\min_{\hat{x}} |A\hat{x} - b|_2^2$. Here $A \in \mathsf{R}^{n \times d}$ 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 $Z \sim \mathcal{N}(0, \sigma^2 I_n)$. In the first set of experiments with simulated delays, we choose $n = 120$, $d = 60$, and $\sigma = 1$. For the the second set of experiments on the distributed cluster, we choose a larger 10GB least squares problem with $n = 600000$ and $d = 200000$, and $\sigma = 100$.

Simulated Delays. In our first set of experiments, we empirically validate our theoretical results by comparing the iteration complexity of these algorithms in the stochastic delay model described in Section 1.1, and in a more general delay model. First we simulate ADSAGA, SGD, IAG in the distributed data setting in the delay model of Section 1.1 with all p_i equal; we simulate ASAGA in this model in the shared-data setting; and we also compare to minibatch SAGA assuming a *synchronous* implementation with a minibatch size m equal to the number of machines. We 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, 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. Second, we simulate ADSAGA and SGD in a more general delay model where the update time is a *shifted* exponential random variable. In this model, each machine takes T time to compute its update, where T is a random variable distributed according to an exponential distribution with some shift s:

$$T \sim s + \text{Exp}(1).$$

See Remark 1.1 for why this model reduces to the delay model of Section 1.1 when the shift is zero. Shifted exponential or heavy-tailed work-time distributions are observed in practice (Dean and Barroso (2013); Lee et al. (2017b)).

In Figure 4, we plot the expected iteration complexity to achieve $|\hat{x} - x^*|_2^2 \leq 0.1$, where $x^* := \min_{\hat{x}} |A\hat{x} - b|_2^2$ is the empirical risk minimizer. Figure 4 demonstrates that, in the model in Section 1.1, ADSAGA outperforms the two alternative state-of-the-art algorithms for the distributed setting: SGD and IAG, especially as the number of machines m grows. SGD converges significantly more slowly, even for a small number of machines, due to the variance in gradient steps. In a more practical delay setting with shifted-exponential update times (no longer memoryless, but still heavy-tailed), with 60 machines, ADSAGA still outperforms IAG (Figure 4(b)).

Distributed Experiments. In our second set of experiments, we run the five data-distributed algorithms (ADSAGA, SGD, IAG, minibatch-SAGA, and minibatch-SGD) in the distributed compute cluster and compare their wallclock times to convergence. We do not simulate the shared-data algorithm ASAGA because the full dataset is too large to fit in RAM, and loading the data from memory is very slow. The nodes used contained any of the following four processors: Intel E5-2640v4, Intel 5118, AMD 7502, or AMD 7742. We implement the algorithms in Python using MPI4py, an open-source MPI implementation. For ADSAGA, IAG, asynchronous SGD, and minibatch SAGA, each node stores its partition of the data in RAM. In each of the three asynchronous algorithms, at each iteration, the PS waits to receive a gradient update from any node (using MPI.ANY_SOURCE). The PS then sends the current parameter back to that node and performs the parameter update specified by the algorithm. In the synchronous minibatch algorithm, the PS waits until updates have been received by all nodes before performing an update and sending the new parameter to all nodes. To avoid bottlenecks at the PS, the PS node checks the convergence criterion and logs progress only once per epoch in all algorithms.

We run ADSAGA, SGD, IAG, and minibatch-SAGA on $\hat{x}$ with one PS and m worker nodes for $m \in \{5, 10, 15, 20, 30\}$. We implement the vanilla version of ADSAGA, which does not require the variables u_j designed for heterogeneous update rates. In practice, while we measured substantial heterogeneity in the update rates of each machine — with some machines making up to twice as many updates as others — we observed that the vanilla ADSAGA implementation worked just as well as the full implementation with the u_j variables. For all algorithms, we use a block size of 200 (as per Remark 1.2), and we perform a hyperparameter grid search over step sizes to find the hyperparameters which yield the smallest distance $\hat{x} - x^*$ after 100 epochs.

In Figure 5, we compare the performance of these algorithms in terms of iteration complexity and wallclock time. In Figure 5(a), we plot the accuracy $|\hat{x} - x^*|_2^2$ of each algorithm after 100 epochs, where $x^* := \min_{\hat{x}} |A\hat{x} - b|_2^2$ is the empirical risk minimizer. We observe that the algorithms that do not use variance reduction (asynchronous SGD and minibatch-SGD) are not able to converge nearly as well as the variance-reduced algorithms. ADSAGA and IAG perform similarly, while minibatch-SAGA performs slightly better in terms of iteration complexity. In Figure 5(b), we plot the expected wallclock time to achieve to achieve $|\hat{x} - x^*|_2^2 \leq 10^{-10}$. We only include the variance-reduced algorithms in this plot, as we were not able to get SGD to converge to this accuracy in a reasonable amount of time. Figure 5(b) demonstrates that while the synchronous minibatch-SAGA algorithm may be advantageous in terms of iteration complexity alone, due to the cost of waiting for all workers to synchronize at each iteration, the asynchronous algorithms (IAG and ADSAGA) converge in less wallclock time. Both IAG and ADSAGA perform similarly. This advantage of asynchrony increases as the number of machines increases: while with 5 machines the asynchronous algorithms are only 20% faster, with 30 machines, they are 60% faster. Our experiments confirm that ADSAGA, the natural adaptation of SAGA to the distributed setting, is both amenable to theoretical analysis and performs well practically.

E Minibatch Rates

We prove minibatch rates for SAGA in this appendix for both the shared and distributed data setting. In the shared data setting, minibatch SAGA is an instantiation Algorithm 4 with $S_j = [n]$ for all $j \in [m]$. In the distributed data setting, S_j is the set of functions held at machine j.

Proposition E.1. *Let $f(x) = \frac{1}{n}\sum_i f_i(x)$ be μ-strongly convex and L_f-smooth. Suppose each f_i is convex and L-smooth. Let $\sigma^2 := \mathsf{E}_{i,j}|\nabla f_i(x^*)|_2^2$.*

Consider the minibatch SAGA algorithm (Algorithm 4) with a minibatch size of m for either the shared data or distributed data setting with m machines. Then with a step size of $\eta = \frac{1}{2mL_f+3L}$, after

$$\left(\frac{3n}{m} + \frac{12L}{m\mu} + \frac{4L_f}{\mu}\right)\log\left(\frac{|x^0 - x^*|_2^2 + \frac{4n\sigma^2}{(2mL_f+3L)^2}}{\epsilon}\right)$$

iterations, ie. $\tilde{O}\left(n + \frac{L}{\mu} + \frac{mL_f}{\mu}\log(1/\epsilon)\right)$ total gradient computations, we have

$$|x^k - x^*| \leq \epsilon.$$

Proof. For convenience, we define $y^k := x^k - x^*$, and $\nabla_i := \nabla f_i(x)$. When the iteration k is clear from context, we omit the superscript k.

Algorithm 4 Minibatch SAGA

MinibatchSAGA $x, \eta, \{f_i\}, \{S_j\}, t$
$\alpha_i = 0$ for $i \in [n]$. Initialize last gradients to 0 at each machine
$\overline{\alpha} = 0$. Initialize last gradient averages at PS
for $k = 0$ **to** t **do**
  for $j = 1$ **to** m **do**
    $i_j \sim \text{Uniform}(S_j)$. Randomly choose a function
    $g \leftarrow \sum_j \nabla f_{i_j}(x)$. Compute the minibatch gradient
    $\beta \leftarrow \sum_j \alpha_{i_j}(x)$. Compute the variance reduction term
  end for
  for $j = 1$ **to** m **do**
    $\alpha_{i_j} \leftarrow \nabla f_{i_j}(x)$. Update the relevant $_i$ variables
    $x \leftarrow x - \eta(g - \beta + m\overline{\alpha})$. Take a gradient step
    $\overline{\alpha} \leftarrow \mathsf{E}_i[\alpha_i]$. Update $^-$
  end for
end for
Return x

Consider the following potential function

$$\phi(x, \alpha) := \phi_1(x, \alpha) + \phi_2(x, \alpha),$$

where

$$\phi_1(x, \alpha) := |x - x^*|_2^2;$$
$$\phi_2(x, \alpha) := 4n\eta^2 \mathsf{E}_{i,j}\left[|\alpha_i - \nabla_i(x^*)|_2^2\right].$$

For convenience we denote $\phi(k) := \phi(x^k, \alpha^k)$.

Let $B := \{i_j\}_{j\in[m]}$ be the minibatch chosen at iteration k. Let $\mathbf{1}_B \in \mathsf{N}^n$ be the indicator vector for the multi-set B. Let $S(B)$ be the set containing all elements of B and $\mathbf{1}_{S(B)}$ the corresponding indicator vector. Let $\mathbf{1}_{S_j} \in \{0,1\}^n$ be the indicator vector of S_j, the data points at machine j.

Lemma E.2.

$$\mathsf{E}_B[\phi_1(k+1)] - \phi_1(k) \le -2m\eta y^T \nabla f(x) + \eta^2 m^2 |\nabla f(x)|_2^2 + 2\eta^2 mLy^T \nabla f(x) + 2\eta^2 m \mathsf{E}_{i,j}\left[|\alpha_i - \nabla_i(x^*)|_2^2\right].$$

Proof. Let α denote the matrix whose ith column is α_i. We abuse notation by using $\overline{\alpha}$ to also mean the matrix $\mathbf{1}_n\overline{\alpha}^T$. We have

$$\phi_1(k+1) - \phi_1(k) = -2m\eta y^T(\nabla - \alpha + \overline{\alpha})\mathbf{1}_B + \eta^2 \mathbf{1}_B^T(\nabla - \alpha + \overline{\alpha})^T(\nabla - \alpha + \overline{\alpha})\mathbf{1}_B,$$

so

$$\begin{aligned}\mathsf{E}_B[\phi_1(k+1)] - \phi_1(k) &= -\frac{2m}{n}\eta y^T \nabla f(x) + \eta^2 \mathsf{E}_B\left[\mathbf{1}_B^T(\nabla - \alpha + \overline{\alpha})^T(\nabla - \alpha + \overline{\alpha})\mathbf{1}_B\right] \\ &= -\frac{2m}{n}\eta y^T \nabla f(x) + \eta^2 \operatorname{Tr}\left(\mathsf{E}_B\left[\mathbf{1}_B\mathbf{1}_B^T\right](\nabla - \alpha + \overline{\alpha})^T(\nabla - \alpha + \overline{\alpha})\right),\end{aligned} \tag{14}$$

where the second line follows from the circular law of trace. □

Consider first the shared data case. Here we have

$$\mathsf{E}_B\left[\mathbf{1}_B\mathbf{1}_B^T\right] = \frac{m(m-1)}{n(n-1)}\mathbf{1}\mathbf{1}^T + \left(\frac{m}{n} - \frac{m(m-1)}{n(n-1)}\right)I_n.$$

In the distributed data case, we have

$$\mathsf{E}_B\left[\mathbf{1}_B\mathbf{1}_B^T\right] = \frac{m^2}{n^2}\mathbf{1}\mathbf{1}^T - \frac{m^2}{n^2}\sum_j \mathbf{1}_{S_j}\mathbf{1}_{S_j}^T + \frac{m}{n}I_n \preceq \frac{m^2}{n^2}\mathbf{1}\mathbf{1}^T + \frac{m}{n}I_n.$$

Hence by linearity of the trace operator, in both cases,

$$\begin{aligned}
&\operatorname{Tr}\left(\mathsf{E}_B\left[\mathbf{1}_B\mathbf{1}_B^T\right](\nabla-\alpha+\overline{\alpha})^T(\nabla-\alpha+\overline{\alpha})\right)\\
&\leq \frac{m^2}{n^2}\operatorname{Tr}\left(\mathbf{1}\mathbf{1}^T(\nabla-\alpha+\overline{\alpha})^T(\nabla-\alpha+\overline{\alpha})\right)+\frac{m}{n}\operatorname{Tr}\left((\nabla-\alpha+\overline{\alpha})^T(\nabla-\alpha+\overline{\alpha})\right)\\
&= \frac{m^2}{n^2}\mathbf{1}^T(\nabla-\alpha+\overline{\alpha})^T(\nabla-\alpha+\overline{\alpha})\mathbf{1}+\frac{m}{n}\sum_i|\nabla_i-\alpha_i+\overline{\alpha}|_2^2\\
&\leq m^2|\nabla f(x)|_2^2+2m\mathsf{E}_{i,j}\left[\nabla_i-\nabla_i(x^*)\right]+2m\mathsf{E}_{i,j}\left[\alpha_i-\nabla_i(x^*)+\overline{\alpha}\right]\\
&\leq m^2|\nabla f(x)|_2^2+2m\mathsf{E}_{i,j}\left[\nabla_i-\nabla_i(x^*)\right]+2m\mathsf{E}_{i,j}\left[\alpha_i-\nabla_i(x^*)\right],
\end{aligned}$$

where the third line follows from the circular law of trace, the fourth line from Jensen's inequality, and the fifth line from the positivity of variance, that is $\mathsf{E}_{i,j}\left[\alpha_i-\nabla_i(x^*)+\overline{\alpha}\right]=\mathsf{E}_{i,j}\left[\alpha_i-\nabla_i(x^*)\right]+|\overline{\alpha}|_2^2$.

Plugging in Lemma C.9 and combining with (14) yields the lemma.

Lemma E.3.

$$\mathsf{E}_B[\phi_2(k+1)]-\phi_2(k)\leq-\left(1-\exp\left(-\frac{m}{n}\right)\right)\phi_2(k)+4m\eta^2Ly^T\nabla f(x)$$

Proof.

$$\phi_3(k+1)-\phi_3(k)=4n\eta^2\left(\sum_{i\in S(B)}\left(|\nabla_i-\nabla_i(x^*)|_2^2-|\alpha_i-\nabla_i(x^*)|_2^2\right)\right)\tag{15}$$

In the shared data setting, $\mathsf{E}_B[\mathbf{1}_{S(B)}]=\left(1-\left(1-\frac{1}{n}\right)^m\right)\mathbf{1}$. In the distributed data setting, $\mathsf{E}_B[\mathbf{1}_{S(B)}]=\frac{m}{n}\mathbf{1}$.

Now

$$\left(1-\exp\left(-\frac{m}{n}\right)\right)\leq\left(1-\left(1-\frac{1}{n}\right)^m\right)\leq\frac{m}{n}.$$

Plugging these bounds into (15) with Lemma C.9 directly yields the lemma. □

Combining these two lemmas, we get

$$\mathbf{E}[\phi(k+1)]-\phi(k)\leq(-2\eta m+6m\eta^2L)y^T\nabla f(x)+\eta^2m^2|\nabla f(x)|_2^2-\left(1-\frac{m}{2n}-\exp\left(-\frac{m}{n}\right)\right)\phi_2(k).$$

Now for $\eta<\frac{1}{6L}$, we have

$$(-2m\eta+6m\eta^2L)y^T\nabla f(x)+\eta^2m^2|\nabla f(x)|_2^2\leq-\eta my^T\nabla f(x)+\eta^2m^2|\nabla f(x)|_2^2.$$

Further, if $\eta<\frac{1}{2mL_f}$, by the L_f-smoothness of f, we have

$$-\eta my^T\nabla f(x)+\eta^2m^2|\nabla f(x)|_2^2\leq-\frac{\eta m}{2}y^T\nabla f(x).$$

By the μ-strong convexity of f, we have

$$-\frac{\eta m}{2}y^T\nabla f(x)\leq-\frac{\mu\eta m}{2}|y|_2^2=\frac{\mu\eta m}{2}\phi_1(k).$$

Further, $\left(1-\frac{m}{2n}-\exp\left(-\frac{m}{n}\right)\right)\geq\frac{m}{3n}$. It follows that for $\eta\leq\frac{1}{2mL_f+6L}$,

$$\mathbf{E}[\phi(k+1)]-\phi(k)\leq\frac{\eta\mu m}{2}\phi_1(k)-\frac{m}{3n}\phi_2(k).\tag{16}$$

Choosing $\eta=\frac{1}{2mL_f+6L}$, it follows that

$$\mathsf{E}_B[\phi(k+1)|\phi(k)]\leq(1-\gamma)\,\phi(k),$$

where $\gamma = \min\left(\frac{\mu m}{4mL_f + 12L}, \frac{m}{3n}\right)$.

Hence after $k = \frac{1}{\gamma}\log\left(\frac{\phi(0)}{\epsilon}\right)$ iterations, we have

$$\mathsf{E}[\phi(k)] \leq (1-\gamma)^k \phi(0) \leq \exp(-\gamma k)\phi(0) = \epsilon.$$

Now $\phi(0) = |x^0 - x^*|_2^2 + \frac{4n\sigma^2}{(2m^2L_f + 3mL)^2}$, and $\phi(k) \geq |x^k - x^*|_2^2$,

so after $k = \left(\frac{3n}{m} + \frac{12L}{m\mu} + \frac{4L_f}{\mu}\right)\log\left(\frac{|x^0 - x^*|_2^2 + \frac{4n\sigma^2}{\left(2mL_f + 3L\right)^2}}{\epsilon}\right)$ iterations, we have

$$\mathsf{E}[|x^k - x^*|] \leq \epsilon.$$

□