2016 IEEE 16th International Conference on Data Mining

Efficient Distributed SGD with Variance Reduction

Soham De and Tom Goldstein
Department of Computer Science, University of Maryland, College Park, USA 20742
Email: sohamde@cs.umd.edu, tomg@cs.umd.edu

***Abstract*—Stochastic Gradient Descent (SGD) has become one of the most popular optimization methods for training machine learning models on massive datasets. However, SGD suffers from two main drawbacks: (i) The noisy gradient updates have high variance, which slows down convergence as the iterates approach the optimum, and (ii) SGD scales poorly in distributed settings, typically experiencing rapidly decreasing marginal benefits as the number of workers increases. In this paper, we propose a highly parallel method, *CentralVR*, that uses error corrections to reduce the variance of SGD gradient updates, and scales linearly with the number of worker nodes. *CentralVR* enjoys low iteration complexity, provably *linear* convergence rates, and exhibits linear performance gains up to hundreds of cores for massive datasets. We compare *CentralVR* to state-of-the-art parallel stochastic optimization methods on a variety of models and datasets, and find that our proposed methods exhibit stronger scaling than other SGD variants.**

I. Introduction

Stochastic gradient descent (SGD) [1] is a general approach for solving minimization problems where the objective function decomposes into a sum over many terms. Such a function has the form

$$\min_x f(x), \quad f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x), \tag{1}$$

where each $f_i : \mathbb{R}^d \to \mathbb{R}$. This encompasses a wide range of problem types, including matrix completion and graph cuts [2]. However, the most popular use of SGD is for problems where the summation in (1) is over a large number of elements in a dataset. For example, each $f_i(x)$ could measure how well a certain model with parameter vector x explains or classifies the i-th entry in a large dataset.

For problems where each term in (1) corresponds to a single data observation, SGD selects a single data index i_k on each iteration k, approximates the gradient of the objective as $g^k = \nabla f_{i_k}(x^k) \approx \nabla f(x^k)$, and then performs the approximate gradient update

$$x^{k+1} = x^k - \eta g^k.$$

Typically, i_k is chosen uniformly at random from $\{1, 2, \cdots, n\}$ on each iteration k, thus making the gradient approximation unbiased. More precisely, we have $\mathbb{E}\big[\nabla f_{i_k}(x^k)\big] = \nabla f(x^k)$, where the expectation is with respect to the random index i_k of the sampled data. These approximate gradient updates are much cheaper than true gradient update steps, which is highly advantageous when x^k is far from the true solution.

A major drawback of SGD is that it is an inherently sequential algorithm. For truly large datasets, parallel or distributed algorithms are vital, driving interest in SGD variants that parallelize over massive distributed datasets. While there has been quite a bit of recent work in the area of parallel asynchronous SGD algorithms [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], these methods typically experience substantially reduced marginal benefit as the number of worker nodes increase over a certain limit. Thus, while some of these algorithms scale linearly when the number of worker nodes is small, they are less effective when the data is distributed over hundreds or thousands of nodes.

Moreover, most research in parallel or distributed SGD methods has been focused on the parameter server model of computation [2], [3], [5], [6], [8], where each update to the centrally stored parameter vector requires a communication phase between the local node and the central server. However, SGD methods tend to become unstable with infrequent communication, and there has been less work in the truly distributed setting where communication costs are high [10], [11], [12]. In this paper, we propose to boost the scalability of stochastic optimization algorithms using *variance reduction* techniques, yielding SGD methods that scale linearly over hundreds or thousands of nodes and can train models on massive datasets without the slowdown that existing stochastic methods experience.

A. Background

Variance reduction (VR) methods [13], [14], [15], [16], [17], [18], [19], [20], [21], [22] have recently gained popularity as an alternative to classical SGD. These methods reduce the variance in the stochastic gradient estimates, and are able to maintain a large constant step size to achieve fast convergence to high accuracy.

VR methods exploit the fact that gradient errors are highly correlated between different uses of the same function f_{i_k}. This is done by subtracting an error correction term from $\nabla f_{i_k}(x^k)$ that estimates the gradient error from the most recent use of f_{i_k}. Thus the stochastic gradients used by VR methods have the form

$$g^k = \underbrace{\nabla f_{i_k}(x^k)}_{\text{approximate gradient}} - \underbrace{\nabla f_{i_k}(y) + \widetilde{g}_y}_{\text{error correction term}}, \tag{2}$$

where y is an old iterate, and $\widetilde{g}_y$ is an approximation of the true gradient $\nabla f(y)$. Typically, $\widetilde{g}_y$ can be kept fixed over an epoch or can be updated cheaply on every iteration. As an example, the SVRG algorithm [13] has an update rule of the form

$$x^{k+1} = x^k - \eta\Big(\nabla f_{i_k}(x^k) - \nabla f_{i_k}(y) + \nabla f(y)\Big), \qquad (3)$$

where y is chosen to be a recent iterate from the algorithm history and is fixed over 1 or 2 epochs, and $\tilde{g}_y = \nabla f(y)$ is the true gradient of f at y, which needs to be computed once every 1 or 2 epochs. Another popular VR algorithm, SAGA [14], uses the following corrected gradient approximation

$$g^k = \nabla f_{i_k}(x^k) - \nabla f_{i_k}(\phi_{i_k}) + \frac{1}{n}\sum_{j=1}^{n} \nabla f_j(\phi_j), \qquad (4)$$

where each $\nabla f_j(\phi_j)$ denotes the most recent value of ∇f_j and ϕ_j denotes the iterate at which the most recent ∇f_j was evaluated. In this case $\widetilde{g}_y$ is the average of the $\nabla f_j(\phi_j)$ values for all $i \in \{1, 2, \cdots, n\}$. This error correction term reduces the variance in the stochastic gradients, and thus ensures fast convergence. Notice that for both the algorithms, SVRG and SAGA, if i_k is chosen uniformly at random from $\{1, 2, \cdots, n\}$, and conditioning on all x, we have $\mathbb{E}\big[g^k\big] = \nabla f(x^k)$. Thus, the error correction term has expected value 0 and the approximate gradient g^k is unbiased for both SVRG and SAGA.

Most work on VR methods has focused on studying their faster convergence rates and better stability properties when compared to classical SGD in the sequential setting. While there have been a few recent papers on parallelizing VR methods, these methods scale poorly in distributed settings and all prior work that we know of has focussed on small-scale parallel or shared memory settings, with the data distributed over 10 or 20 nodes [15], [23], [24]. These parallel algorithms use a parameter server model of computation, and are based on the assumption that communication costs are low, which may not be true in large-scale heterogenous distributed computing environments. The fact that the error correction term reduces the variance in the stochastic gradients, however, seems to indicate that distributed VR methods could be helpful in distributed settings. In particular, the variance-reduced gradients would help in dealing with the problems of instability and slower convergence faced by regular stochastic methods when the frequency of communication between the server and the local nodes is increased.

B. Contributions

In this work, we use variance reduction to dramatically boost the performance of SGD in the highly distributed setting. We do this by exploiting the dependence of VR methods on the gradient correction term $\widetilde{g}_y$. We allow many local worker nodes to run simultaneously, while communicating with the central server only through the exchange of this central error correction term and the locally stored iterates. The proposed schemes allow many asynchronous processes to work towards a central solution with minimal communication, while simultaneously benefitting from the fast convergence provided by VR.

This work has four main contributions:

- First, we present a new VR algorithm *CentralVR*, built on SAGA, that is robust to noise and variance in the dataset. We propose synchronous (*CentralVR-Sync*) and asynchronous (*CentralVR-Async*) variations of *CentralVR* which can linearly scale up over massive datasets using hundreds of cores.
- Second, we theoretically study the convergence of *CentralVR* when $\widetilde{g}_y$ is only updated periodically (as in the distributed setting), and prove linear convergence of the method with constant stepsizes.
- Third, we propose distributed versions of the existing popular VR algorithms, SVRG and SAGA, that are robust to high communication latency between the worker nodes and the central server, and can scale over large distributed settings ranging over hundreds of nodes. Table I summarizes the distributed algorithms proposed in the paper and their storage and computation requirements.
- Finally, we present empirical results over different models and datasets that show that these distributed algorithms can be trained on massive highly distributed datasets in far less time than existing state-of-the-art stochastic optimization methods. Performance of all these distributed methods scales linearly up to hundreds of workers with low communication frequency. We show empirically that the proposed methods converge much faster than competing options.

TABLE I: Distributed Algorithms Proposed

Proposed Algorithm	Asynchronous?	Storage (No. of gradients)	Gradients/Iteration
CentralVR-Sync	No	n	1
CentralVR-Async	Yes	n	1
Distributed SVRG	No	2	2.5
Distributed SAGA	Yes	n	1

II. CentralVR Algorithm: single-worker case

We begin by proposing our new VR scheme, *CentralVR*, in the single-worker case. As we will see later, the proposed method has a natural generalization to the distributed setting that has low communication requirements.

A. Algorithm Overview

Our proposed VR scheme is divided into epochs, with n updates taking place in each epoch. Let the iterates generated in the m-th epoch be written as $\{x_m^j\}_{j=1}^n$. Also let $\tilde{x}_m^l$ denote the iterate at which the l-th data index was most recently before the $m+1$-th epoch (i.e., on or before the m-th epoch). Then, the update rule for *CentralVR* is given by:

$$x_{m+1}^{k+1} = x_{m+1}^k - \eta v_{m+1}^k, \qquad (5)$$

where v_{m+1}^k is defined as

$$v_{m+1}^k := \nabla f_{i_k}(x_{m+1}^k) - \nabla f_{i_k}(\tilde{x}_m^{i_k}) + \overline{g}_m, \qquad (6)$$

$$\text{and} \quad \overline{g}_m := \frac{1}{n}\sum_{j=1}^{n} \nabla f_j(\tilde{x}_m^j).$$

Thus, $\overline{g}_m$ is the average of the gradients of all component functions $\{\nabla f_j\}_{j=1}^n$, each evaluated at the most recent iterate $\{\tilde{x}_m^j\}_{j=1}^n$ at which the corresponding function was used on or before the m-th epoch. These gradients are stored in a table, and the average gradient $\overline{g}_m$ is updated at the end of each epoch, i.e., after every n parameter updates.

Note that if i_k is chosen uniformly at random from the set $\{1, 2, \cdots, n\}$ on each iteration k, then, conditioning on all history (all x), we have

$$\mathbb{E}\big[\nabla f_{i_k}(\tilde{x}_m^{i_k})\big] = \frac{1}{n}\sum_{j=1}^{n} \nabla f_j(\tilde{x}_m^j) = \overline{g}_m.$$

Thus, the error correction term has expected value 0, and $\mathbb{E}\big[v_{m+1}^k\big] = \nabla f(x_{m+1}^k)$, i.e., the approximate gradient v_{m+1}^k is unbiased.

B. Permutation Sampling

In practical implementations, it is natural to consider a random permutation of the data indices on every epoch, rather than uniformly choosing a random index on every iteration. Thus, on each epoch, a random permutation of the data indices is chosen and a pass is made over the entire dataset, resulting in n updates, one per data sample. Permutation sampling often outperforms uniform random sampling empirically [25], [26], although theoretical justification for this is still limited (see [27], [28] for some recent results).

As an alternative to uniform random sampling, *CentralVR* can leverage random permutations over the data indices. Let π_m denote a random permutation of the data indices $\{1, 2, \cdots, n\}$ for the m-th epoch, with π_m^j denoting the data index chosen in the j-th iteration in the m-th epoch. Thus, now $\tilde{x}_m^l$ denotes the iterate corresponding to the point when the l-th data index was chosen in the m-th epoch. The update rule with the random permutation is given by (5) and (6), with $i_k = \pi_{m+1}^k$.

Summing (5) over all $k = 0, 1, \cdots, n-1$, we get

$$\begin{aligned}\sum_{k=0}^{n-1} v_{m+1}^k &= \sum_{k=0}^{n-1}\Big(\nabla f_{\pi_{m+1}^k}(x_{m+1}^k) - \nabla f_{\pi_{m+1}^k}(\tilde{x}_m^{i_k}) + \overline{g}_m\Big)\\ &= \sum_{k=0}^{n-1}\Big(\nabla f_k(\tilde{x}_{m+1}^k) - \nabla f_k(\tilde{x}_m^k) + \overline{g}_m\Big)\\ &= \sum_{k=0}^{n-1} \nabla f_k(\tilde{x}_{m+1}^k).\end{aligned}$$

Thus, summing (5) over all $k = 0, 1, \cdots, n-1$, using the telescoping sum in x_{m+1}^k, and using the convention that $x_{m+1}^0 = x_m^n$, we get

$$x_{m+2}^0 = x_{m+1}^0 - \eta \sum_{j=1}^{n} \nabla f_j\big(\tilde{x}_{m+1}^j\big). \tag{7}$$

Equation (7) shows the update rule in terms of the iterates at the ends of the epochs. Thus, over an epoch, the average gradient accumulated by *CentralVR* is unbiased and thus is a good estimate of the true gradient. This average gradient term can be accumulated cheaply during an epoch, without any noticeable overhead.

C. Algorithm Details for CentralVR

The detailed steps of *CentralVR* are listed in Algorithm 1. Note, the stored gradients and the average gradient term $\widetilde{g}_y$ are initialized using a single epoch of "vanilla" SGD with no VR correction.

Algorithm 1 CentralVR Algorithm: single worker case

```
1: parameters learning rate $\eta$
2: initialize $x$, $\{\nabla f_j(\tilde{x}^j)\}_j$, and $\overline{g}$ using plain SGD
3: while not converged do
4:     $\widetilde{g} \leftarrow 0$
5:     set $\pi$: random permutation of indices $1, 2, \cdots, n$
6:     for $k$ in $\{1, \ldots, n\}$ do
7:         set: $x^{k+1} \leftarrow x^k - \eta\big(\nabla f_{\pi_k}(x^k) - \nabla f_{\pi_k}(\tilde{x}^{\pi_k}) + \overline{g}\big)$
8:         accumulate average: $\widetilde{g} \leftarrow \widetilde{g} + \nabla f_{\pi_k}(x^k)/n$
9:         store gradient: $\nabla f_{\pi_k}(\tilde{x}^{\pi_k}) \leftarrow \nabla f_{\pi_k}(x^k)$
10:    end for
11:    set average gradient for next epoch: $\overline{g} \leftarrow \widetilde{g}$
12: end while
```

CentralVR builds on the SAGA method. SAGA relies on the update rule (4), which requires an average over a large number of iterates ($\widetilde{g}_y = \frac{1}{n}\sum_i \nabla f_i(\phi_i)$) to be continuously updated on every iteration. In the distributed setting, where the vector $\widetilde{g}_y$ must be shared across nodes, maintaining an up-to-date average requires large amounts of communication. This makes SAGA less stable in distributed implementations when the communication frequency is decreased. Updating $\widetilde{g}_y$ only occasionally (as we do in the distributed variants of *CentralVR* below) translates into significant communication savings in the distributed setting.

CentralVR has the same time and space complexities as SAGA. Namely, on every iteration, 1 gradient computation is required, similar to SGD, and the n gradients $\{\nabla f_k(\tilde{x}_m^k)\}_{k=1}^n$ also need to be stored. Note that this is not a significant storage requirement, since for models like logistic regression and ridge regression only a single number is required to be stored corresponding to each gradient.

III. Convergence Analysis

We now present convergence bounds for Algorithm 1. We make the following standard assumptions about the function when studying convergence properties. First, each f_i is strongly convex with strong convexity constant μ:

$$f_i(x) \geq f_i(y) + \nabla f_i(y)^T(x-y) + \frac{\mu}{2}\|x-y\|^2. \tag{8}$$

Second, each f_i has Lipschitz continuous gradients with Lipschitz constant L so that

$$f_i(x) \leq f_i(y) + \nabla f_i(y)^T(x-y) + \frac{L}{2}\|x-y\|^2. \tag{9}$$

We now present our main result. The proof is presented in the appendix.

Theorem 1. *Consider* CentralVR *with data index i_k drawn uniformly at random (with replacement) on each iteration k. Define α as*

$$\alpha := \max\Big(1 - \eta\mu, \frac{2L^2\eta}{\mu(1-2L\eta)}\Big).$$

If the step size η is small enough such that $0 < \alpha < 1$, then we have the following bound:

$$\begin{aligned}&\left\|x^0_{m+2} - x^\star\right\|^2 + c\big(\overline{f(x_{m+1})} - f(x^\star)\big)\\ &\le \alpha\Big(\left\|x^0_{m+1} - x^\star\right\|^2 + c\big(\overline{f(\tilde{x}_m)} - f(x^\star)\big)\Big),\end{aligned}$$

where $c = 2n\eta(1 - 2L\eta)$ and we define $\overline{f(x_m)} := \frac{1}{n}\sum_{k=0}^{n-1} f(x^k_m)$. In other words, the method converges linearly.

Remark on Step Size Restrictions: From Theorem 1, notice that *CentralVR* converges linearly when the step size η is small enough such that

$$\eta < \min\Big(\frac{1}{\mu}, \frac{1}{2L}, \frac{\mu}{2L(L+\mu)}\Big).$$

If we observe that $L \ge \mu$, then we see that this condition is satisfied whenever $\eta < \frac{\mu}{2L(L+\mu)}$.

IV. Distributed Algorithms

We now consider the distributed setting, with a single central server and p local clients, each of which contains a portion of the data set. In this setting, the data is decomposed into disjoint subsets $\{\Omega_s\}$, where s denotes a particular local client, and $\sum_s |\Omega_s| = n$. We denote the i-th function stored on client s as f^s_i. Our goal is to minimize the global objective function of the form

$$f(x) = \frac{1}{n}\sum_{s=1}^{p}\sum_{j=1}^{|\Omega_s|} f^s_j(x).$$

We consider a centralized setting, where the clients can only communicate with the central server, and our goal is to derive stochastic algorithms that scale linearly to high p, while remaining stable even under low communication frequencies between local and central nodes.

A. Synchronous Version

CentralVR naturally extends to the distributed synchronous setting, and is presented in Algorithm 2. To distinguish the algorithm from the single worker case, we call it *CentralVR-Sync*. On each epoch, the local nodes first retrieve a copy of the central iterate x, and also $\tilde{g}_y$, which represents the averaged gradient over all data. The *CentralVR* method is then performed on each node, and the most recent gradient for each data point $\nabla f^s_{i_k}(\tilde{x}^{i_k})$ is stored. By sharing $\tilde{g}_y$ across nodes, we ensure that the local gradient updates utilize global gradient information from remote nodes. This prevents the local node from drifting far away from the global solution, even if each local node runs for one whole epoch before communicating back with the central server.

Algorithm 2 CentralVR-Sync Algorithm

```
1: parameters learning rate $\eta$
2: initialize $x$, $\{\nabla f_j(\tilde{x}^j)\}_j$, $\overline{g}$
3: while not converged do
4:   for each local node $s$ do
5:     $\tilde{g} \leftarrow 0$
6:     set $\pi$: random permutation of indices $1, 2, \cdots, |\Omega_s|$
7:     for $k$ in $\{1, \dots, |\Omega_s|\}$ do
8:       $x^{k+1} \leftarrow x^k - \eta\big(\nabla f^s_{\pi_k}(x^k) - \nabla f^s_{\pi_k}(\tilde{x}^{\pi_k}) + \overline{g}\big)$
9:       accumulate average: $\tilde{g} \leftarrow \tilde{g} + \nabla f^s_{\pi_k}(x^k)/|\Omega_s|$
10:      store gradient: $\nabla f^s_{\pi_k}(\tilde{x}^{\pi_k}) \leftarrow \nabla f^s_{\pi_k}(x^k)$
11:    end for
12:    set average gradient to send to server: $\overline{g} \leftarrow \tilde{g}$
13:    send $x$, $\overline{g}$ to central node
14:    receive updated $x$, $\overline{g}$ from central node
15:  end for
16:  central node:
17:    average $x$, $\overline{g}$ received from workers
18:    broadcast averaged $x$, $\overline{g}$ to local workers
19: end while
```

In *CentralVR-Sync*, each local node performs local updates for one epoch, or $|\Omega_s|$ iterations, before communicating with the server. This is a rather low communication frequency compared to a parameter server model of computation in which updates are continuously streamed to the central node. This makes a significant difference in runtimes when the number of local nodes is large, as shown in later sections.

B. Asynchronous Version

The synchronous algorithm can be extended very easily to the asynchronous case, *CentralVR-Async*, as shown in Algorithm 3. In *CentralVR-Async*, the central server keeps a copy of the current iterate x and average gradient $\overline{g}$. The key idea for *CentralVR-Async* is that, once a local node completes an epoch, it sends the *change* in the local averages, given by Δx_s and $\Delta\overline{g}_s$, over the last epoch to the central server. This change is added to the global x and $\overline{g}$ to update the parameters stored on the central server. Thus, when the central server receives parameters from a local node s, the updates it performs have the form

$$x = x + \frac{1}{p}\Delta x_s \quad \text{and} \quad \overline{g} = \overline{g} + \frac{1}{p}\Delta\overline{g}_s,$$

where Δx_s and $\Delta\overline{g}_s$ are given by

$$\begin{aligned}&\Delta x_s = \{x^n_{m+1} - x^n_m\}_s \qquad \text{and}\\ &\Delta\overline{g}_s = \Bigg\{\frac{1}{|\Omega_s|}\sum_{j\in\Omega_s}\nabla f^s_j(\tilde{x}^j_{m+1}) - \frac{1}{|\Omega_s|}\sum_{j\in\Omega_s}\nabla f^s_j(\tilde{x}^j_m)\Bigg\}_s.\end{aligned}$$

Sending the change in the local parameter values rather than the local parameters themselves ensures that, when updating the central parameter, the previous contribution to the average from that local worker is just replaced by the new contribution. Thus, a fast working local node does not bias the global average solution toward its local solution with an excessive number of updates. This makes the algorithm more robust to heterogenous computing environments where nodes work at disparate speeds.

Algorithm 3 CentralVR-Async Algorithm
```
1:  parameters learning rate $\eta$
2:  initialize $x$, $\{\nabla f_j(\tilde{x}^j)\}_j, \overline{g}, \alpha = 1/p, x_{\text{old}} = \overline{g}_{\text{old}} = 0$
3:  while not converged do
4:    for each local node do
5:      $\widetilde{g} \leftarrow 0$
6:      set $\pi$: random permutation of indices $1, 2, \cdots, |\Omega_s|$
7:      for $k$ in $\{1, \ldots, |\Omega_s|\}$ do
8:        $x^{k+1} \leftarrow x^k - \eta\big(\nabla f^s_{\pi_k}(x^k) - \nabla f^s_{\pi_k}(\tilde{x}^{\pi_k}) + \overline{g}\big)$
9:        accumulate average: $\widetilde{g} \leftarrow \widetilde{g} + \nabla f^s_{\pi_k}(x^k)/|\Omega_s|$
10:       store gradient: $\nabla f^s_{\pi_k}(\tilde{x}^{\pi_k}) \leftarrow \nabla f^s_{\pi_k}(x^k)$
11:     end for
12:     set average gradient: $\overline{g} \leftarrow \widetilde{g}$
13:     compute change: $\Delta x \leftarrow x - x_{\text{old}}$, $\Delta\overline{g} \leftarrow \overline{g} - \overline{g}_{\text{old}}$
14:     set: $x_{\text{old}} \leftarrow x$, $\overline{g}_{\text{old}} \leftarrow \overline{g}$
15:     send $\Delta x$, $\Delta\overline{g}$ to central node
16:     receive updated $x$, $\overline{g}$ from central node
17:   end for
18:   central node:
19:     receive $\Delta x$, $\Delta\overline{g}$ from a local worker
20:     update: $x \leftarrow x + \alpha\Delta x$, $\overline{g} \leftarrow \overline{g} + \alpha\Delta\overline{g}$
21:     send new $x$, $\overline{g}$ back to local worker
22: end while
```

The proposed *CentralVR* scheme has several advantages. It does not require a full gradient computation as in SVRG, and thus can be made fully asynchronous. Moreover, since the average gradient $\widetilde{g_y}$ in the error correction term is updated only at the end of an epoch, communication periods can be increased between the central server and the local nodes, while still maintaining fast and stable convergence.

V. Distributed Variants of Other VR Methods

In this section, we propose distributed variants of popular variance reduction methods: SVRG and SAGA. The properties of these variants are overviewed in Table I.

A. Distributed SVRG

In this section, we present a distributed version of SVRG appropriate for distributed scenarios with high communication delays. Recently, in [15], the authors presented an asynchronous distributed version of SVRG using a parameter server model of computation. In SVRG, the average gradient term is $\widetilde{g_y} = \nabla f(y)$ as shown in (3). This correction term is very accurate because it uses the entire dataset. This would indicate that the algorithm would be robust to high communication periods between the local nodes and the server.

However, a truly asynchronous method is not possible with SVRG since a synchronization step is unavoidable when computing the full gradient. Thus, in this section, we present a synchronous variant of SVRG in Algorithm 4. We define an additional parameter τ to denote the communication period, i.e., the number of updates to run on each local node before communicating with the central server.

The true gradient $\overline{g}$ is maintained across all nodes throughout the whole communication period τ, thus ensuring that the local workers stay close to the desired solution, even when τ is large. After τ updates, the current iterate x_s on each local node s is averaged on the central server to get x. The true gradient is evaluated at x, i.e., $\overline{g} = \nabla f(x)$, and $\overline{x} = x$ is used on each local node during the next epoch.

Algorithm 4 Synchronous SVRG
```
1:  parameters step size $\eta$, communication period $\tau$
2:  initialize $x$
3:  while not converged do
4:    set: $\overline{x} \leftarrow x$
5:    set: $\overline{g} \leftarrow \nabla f(\overline{x})$ via synchronization step
6:    for each local node $s$ do
7:      for $k$ in $\{1, \ldots, \tau\}$ do
8:        sample $i_k \in \{1, \ldots, |\Omega_s|\}$ with replacement
9:        $x^{k+1} \leftarrow x^k - \eta\big(\nabla f^s_{i_k}(x^k) - \nabla f^s_{i_k}(\overline{x}) + \overline{g}\big)$
10:     end for
11:     send $x$ to central node
12:     receive updated $x$ from central node
13:   end for
14:   central node:
15:     average $x$ received from workers
16:     broadcast averaged $x$ to local workers
17: end while
```

B. Distributed SAGA

The update rule for SAGA is given in (4). Since there is no synchronization step required as in SVRG, there is a very natural asynchronous version of the algorithm under the parameter server model of computation. A linear convergence proof has been presented for the parameter server model of SAGA (see Theorem 3 in [15]). However, this work does not contain any empirical studies of the method. The parameter server framework is a very natural generalization of SAGA, however it has very high bandwidth requirements for large numbers of nodes.

Algorithm 5 presents an asynchronous version of SAGA with lower communication frequency. Like SVRG, we define a communication period parameter τ which determines the number of iterations to run on each machine before central communication.

In the SAGA algorithm, the average gradient term $\overline{g}$ is updated on each iteration. Thus, as local iterations progress, the average gradient evolves differently on each local node. This makes the algorithm less robust to higher communication periods τ. As the communication period increases, the local nodes drift farther apart from each other and the global solution. Thus, the learning rate needs to shrink as τ increases over a certain limit. This in turn slows down convergence. For this reason, distributed SAGA is less tolerant to long communication periods than the Algorithms in Sections IV and

Algorithm 5 Asynchronous SAGA

```
1: parameters step size $\eta$, communication period $\tau$
2: initialize $x$, $\{\nabla f_j(\tilde{x}^j)\}_j, \alpha = 1/p, x_{\text{old}} = \overline{g}_{\text{old}} = 0$
3: set average gradient: $\overline{g} \leftarrow \frac{1}{n}\sum_j \nabla f_j(\tilde{x}^j)$
4: while not converged do
5:   for each local node do
6:     for $k$ in $\{1, \dots, \tau\}$ do
7:       sample $i_k \in \{1, \dots, n\}$ with replacement
8:       $x^{k+1} \leftarrow x^k - \eta\big(\nabla f^s_{i_k}(x^k) - \nabla f^s_{i_k}(\tilde{x}^{i_k}) + \overline{g}\big)$
9:       update: $\overline{g} \leftarrow \overline{g} + \frac{1}{n}\big(\nabla f^s_{i_k}(x^k) - \nabla f^s_{i_k}(\tilde{x}^{i_k})\big)$
10:      store gradient: $\nabla f^s_{i_k}(\tilde{x}^{i_k}) \leftarrow \nabla f^s_{i_k}(x^k)$
11:    end for
12:    compute change: $\Delta x \leftarrow x - x_{\text{old}}$, $\Delta\overline{g} \leftarrow \overline{g} - \overline{g}_{\text{old}}$
13:    set: $x_{\text{old}} \leftarrow x$, $\overline{g}_{\text{old}} \leftarrow \overline{g}$
14:    send $\Delta x$, $\Delta\overline{g}$ to central node
15:    receive updated $x$, $\overline{g}$ from central node
16:  end for
17:  central node:
18:    receive $\Delta x$, $\Delta\overline{g}$ from a local worker
19:    update: $x \leftarrow x + \alpha\Delta x$, $\overline{g} \leftarrow \overline{g} + \alpha\Delta\overline{g}$
20:    send new $x$, $\overline{g}$ back to local worker
21: end while
```

V-A. However, it still has fast convergence for much higher communication periods than existing stochastic schemes.

The asynchronous SAGA method (Algorithm 5) is built on the same idea as the proposed asynchronous algorithm: running averages are kept on each local node, and at the end of an epoch the *change* in the parameter values are sent to the central server. This makes the algorithm more robust when local nodes work at heterogenous speeds.

In our distributed SAGA algorithm, care has to be taken while updating the average gradient $\overline{g}$. Note that $\overline{g}$ is averaged over the whole dataset. Thus, when replacing the gradient value at the current index i_k, the update is scaled down by a factor of n (the total number of global samples, as opposed to $|\Omega_s|$, the number of local samples). At the end of a local epoch, the average of the stored gradients on each local node is sent back to the central server, along with the current estimate x. This ensures that the average gradient term on the central server $\hat{g}$ is built from the most recent gradient computations at each index.

VI. Empirical Results

In this section, we present the empirical performance of the proposed methods, both in sequential and distributed settings. We benchmark the methods for two test problems: first, a binary classification problem with ℓ_2-regularized logistic regression where each f_i is of the form

$$f_i(x) = \log\big(1 + \exp(b_i a_i^T x)\big) + \lambda\|x\|^2,$$

where feature vector $a_i \in \mathbb{R}^d$ has label $b_i \in \mathbb{R}$. We also consider a ridge regression problem of the form

$$f_i(x) = (a_i^T x - b_i)^2 + \lambda\|x\|^2.$$

We present all our results with the ℓ_2 regularization parameter set at $\lambda = 10^{-4}$, though we found that our results were not sensitive to this choice of parameter.

A. Single Worker Results

We first test our algorithms in the sequential, non-distributed setting. It is well known that VR beats vanilla SGD by a wide margin in many applications. However, the different VR methods vary widely in their empirical behavior. We compare the single worker *CentralVR* algorithm to the two most popular VR methods, SVRG [13] and SAGA [14].

We test the methods on two synthetic "toy" datasets, in addition to two real-world datasets. Synthetic classification data was generated by sampling two normal distributions with unit variance and means separated by one unit. For the least-squares prediction problem, we generate a random normal matrix A and random labels of the form $b = Ax + \epsilon$, where ϵ is standard Gaussian noise. For each case, we kept the size of the dataset at $n = 5000$ with $d = 20$ features. For the binary classification problem, we kept equal numbers of data samples for each class. We also tested performance of our algorithms on two standard real world datasets: IJCNN1 [29] for binary classification and the MILLIONSONG [30] dataset for least squares prediction. IJCNN1 contains 35,000 training data samples of 22 dimensions, while MILLIONSONG contains 463,715 training samples of 90 dimensions. For all our experiments, we maintain a constant learning rate, and choose the learning rate that yields fastest convergence.

Results appear in Figure 1. We compare convergence rates of the algorithms in terms of number of gradient computations for each method. This provides a level playing field since different VR methods require different numbers of gradient computations per iteration, and gradient computations dominate the computing time. The proposed *CentralVR* algorithm widely out-performs SAGA and SVRG in all cases, requiring less than one-third of the gradient computations of the other methods.

B. Distributed Results

We now present results of our algorithms in highly distributed settings. We implement the algorithms using a Python binding to MPI, and all experiments were run on an Intel Xeon E5 cluster with 24 cores per node. All our asynchronous implementations are "locked", where at a given time only one local node can update the parameters on the central server. However, all proposed asynchronous algorithms can be easily implemented in a lock-free setting, leading to further speedups.

We compare the distributed versions of *CentralVR*, *CentralVR-Async* [CVR-Async in Figures 2 and 3] and *CentralVR-Sync* [CVR-Sync in Figures 2 and 3], proposed in Section IV with the following algorithms:

- Distributed SVRG (Section V-A) [D-SVRG in Figures 2 and 3]. We set the communication period $\tau = 2n$ as recommended in [13]. We found the performance of the algorithm to be very robust to τ.

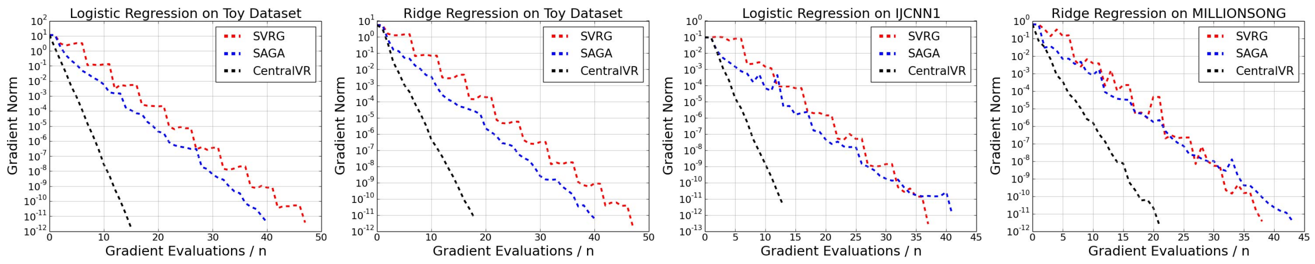

Fig. 1: Single Worker Results. Logistic regression on toy dataset; Ridge regression on toy data; Logistic regression on IJCNN1 dataset; Ridge regression on MILLIONSONG dataset; In each case *CentralVR* converges much faster than SVRG and SAGA.

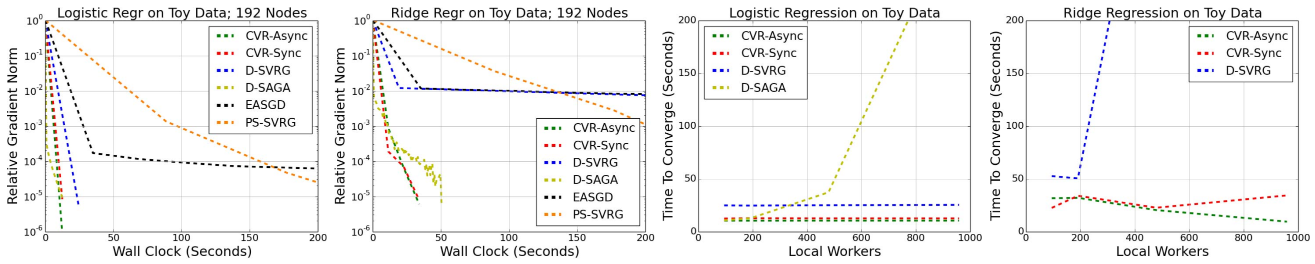

Fig. 2: Distributed Results on toy datasets for *CentralVR-Sync* and *CentralVR-Async*, compared to Distributed SVRG (Section V-A), Distributed SAGA (Section V-B), Parameter Server SVRG and EASGD. Left two plots: Convergence curve for Logistic and ridge regression on synthetic data over 192 nodes. Right two plots: Time required for convergence as number of local workers is increased (data on each local worker is *constant* – i.e., total data scales *linearly* with the number of local workers) for logistic and ridge regression.

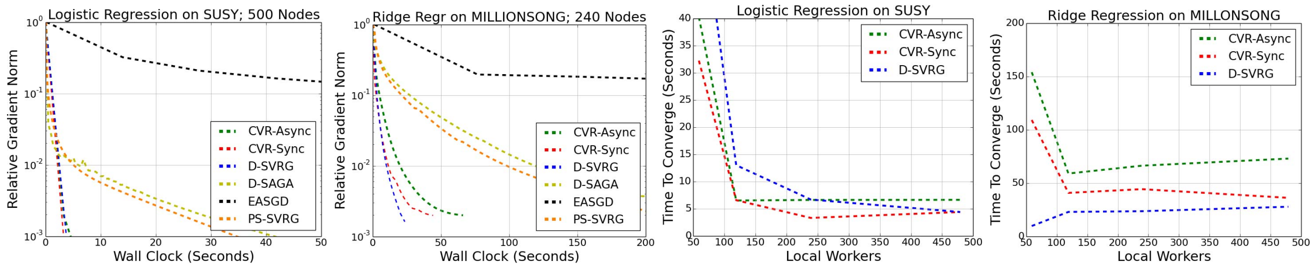

Fig. 3: Distributed Results on SUSY and MILLIONSONG for *CentralVR-Sync* and *CentralVR-Async*, compared to Distributed SVRG (Section V-A), Distributed SAGA (Section V-B), Parameter Server SVRG (Param Server SVRG) and EASGD. (Left two plots) Convergence curve for Logistic regression and ridge regression on SUSY over 500 nodes and on MILLIONSONG over 240 nodes. (Right two plots) Time required for convergence as number of local workers is increased.

- Distributed SAGA (Section V-B) [D-SAGA in Figures 2 and 3]. We vary the communication period $\tau = \{10, 100, 1000, 10000\}$ and present results for the τ yielding best results. The algorithm remains relatively stable for $\tau = \{10, 100, 1000\}$ but convergence speeds start slowing down significantly at $\tau = 10000$.
- Elastic Averaging SGD (EASGD): This is a recently proposed asynchronous SGD method [11] that has been shown to efficiently accelerate training times of deep neural networks. As in [11], we tested the algorithm for communication periods $\tau = \{4, 16, 64\}$, and found results to be nearly insensitive to τ (τ updates occur before communication). We also found the regular EASGD algorithm to outperform the momentum version (M-EASGD). We test performance both for a constant step size as well as a decaying step size (using a local clock on each machine) as given by $\eta_0/(1+\gamma k)^{0.5}$ (as in [11]), where η_0 is the initial step size, k is the local iteration number, and γ is the decay parameter. EASGD has been shown to outperform the related popular asynchronous SGD method Downpour [3], on both convex and non-convex settings.
- Asynchronous "Parameter Server" SVRG [PS-SVRG in Figures 2 and 3]: an asynchronous version of SVRG on a parameter server model of computation [15]. This method outperforms a popular asynchronous SGD method, Hogwild! [2], which also uses a parameter server model. We set the epoch size to $2n$, as recommended in [15].

For the variance reduction methods, we performed experiments using a constant step size, as well as the simple learning rate decay rule $\eta_l = \eta_0 \gamma^l$ (here, l is the number of *epochs*, instead of iterations). Decaying the step size does not yield consistent performance gains, and constant step sizes work very well in practice.

We compared the algorithms on a binary classification problem and a least-squares prediction problem using both toy datasets and real world datasets. The toy datasets were created on each local worker exactly the same way as for the sequential experiments. The toy datasets had $d = 1000$ features and $|\Omega_s| = 5000$ samples for *each* core s, i.e., the total size of the dataset was $p \times 5000$, where p denotes the number of local nodes. We also used the real world datasets MILLIONSONG [30] (containing close to 500,000 data samples) for ridge regression and SUSY [31] (5,000,000 data samples) for logistic regression.

Figure 2 shows results of our distributed experiments on toy datasets. The left two plots compare the rates of convergence of our algorithms scaled over 192 cores for logistic regression and ridge regression. The x-axis displays wall clock time in seconds and the y-axis displays the relative norm of the gradient, i.e., the ratio between the current gradient norm and the initial gradient norm. In almost all cases the proposed algorithms, in particular *CentralVR*, have substantially superior rates of convergence over established schemes. The right two plots in Figure 2 demonstrate the scalability of our algorithms. On the y-axis, we plot the wall clock time (in seconds) required for convergence, and on the x-axis, we vary the number of nodes as 96, 192, 480 and 960. Each local worker has $|\Omega_s| = 5000$ data points in each case, i.e., the amount of data scales linearly with the number of nodes. Notice that *CentralVR-Sync* and *CentralVR-Async* exhibit nearly perfect linear scaling, even when the number of workers is almost 1000. The dataset size in this regime is close to 5 million data points, and the proposed *CentralVR* methods train both our logistic and ridge regression models to five digits of precision in less than 15 seconds.

Figure 3 shows results of our distributed experiments on the large datasets SUSY and MILLIONSONG. The left two plots show convergence results for our algorithms over 500 nodes for SUSY and 240 nodes for MILLIONSONG. In both cases, we see that our proposed algorithms outperform or remain competitive with previously proposed schemes. The right two plots show the scaling of our algorithms as we increase the number of local workers for training SUSY and MILLIONSONG. We see that for MILLIONSONG, increasing the number of local workers initially decreases convergence time, but speed levels out for large numbers of workers, likely due to the smaller size of the local dataset fragments. On the larger SUSY problem, we find a consistent decrease in the convergence times as we increase the number of workers. We train on this 5,000,000 sample dataset in less than 5 seconds using 750 local workers.

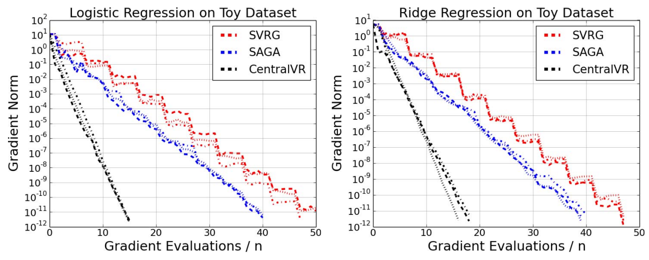

Fig. 4: Single Worker Results with Added Noise. Logistic regression on toy dataset; Ridge regression on toy data;

C. Robustness to Noise and Variance

In this section, we compare the robustness of *CentralVR* in the single worker case on added noise and increased variance in the dataset. We consider the toy datasets used in section VI-A, and add noise while generating the datasets. This is done for the ridge regression data by controlling the ϵ parameter when generating the labels $b = Ax + \epsilon$. For the logistic regression data, we add noise by increasing the variance of the two normal distributions used to generate data for the two classes. This results in increased overlap between the two classes, which increases the variance of the gradients.

Figure 4 presents the results. We see that the error correction term for all three VR algorithms effectively deals with increased noise and variance in the dataset, with convergence times remaining virtually unaffected. We see that over all noise settings, the convergence rate of *CentralVR* remains much faster than both SVRG and SAGA.

VII. Conclusion

This manuscript introduces a new variance reduction scheme, *CentralVR*, that has lower communication requirements than conventional schemes, allowing it to perform better in highly parallel cloud or cluster computing platforms. In addition, distributed versions of well-known variance reduction stochastic gradient descent (SGD) methods are presented that also perform well in highly distributed settings. We show that by leveraging variance reduction, we can combat the diminishing returns that plague classical SGD methods when scaled across many workers, achieving linear performance scaling to over 1000 cores. This represents a significant increase in scalability over previous stochastic gradient methods.

VIII. Acknowledgements

This work was supported by the National Science Foundation (#1535902), and the Office of Naval Research (#N00014-15-1-2676).

IX. Proofs of Technical Results

A. Lemmas

We first start with two lemmas that will be useful in the proof for Theorem 1.

Lemma 1. *For any f defined as $f := \frac{1}{n}\sum_{i=1}^{n} f_i$, where each f_i satisfies* (8) *and* (9)*, and on conditioning on any x, we have*

$$\mathbb{E}\big\|\nabla f_j(x) - \nabla f_j(x^\star)\big\|^2 \le 2L(f(x) - f(x^\star)),$$

where j is sampled uniformly at random from $\{1, 2, \ldots, n\}$ and $x^\star$ denotes the minimizer of f.

Proof. A standard result used frequently in the convex optimization literature is as follows:

$$\|\nabla f_j(x) - \nabla f_j(x^\star)\|^2 \le 2L(f_j(x) - f_j(x^\star) - \langle \nabla f_j(x^\star), x - x^\star\rangle),$$

where f_j is L-Lipschitz smooth. A proof for this inequality can be found in [32] (Theorem 2.1.5 on page 56).

Since j is sampled uniformly at random from $\{1, 2, \ldots, n\}$, we can write

$$\begin{aligned}&\mathbb{E}(f_j(x) - f_j(x^\star) - \langle \nabla f_j(x^\star), x - x^\star\rangle)\\ &\quad = f(x) - f(x^\star) - \langle \nabla f(x^\star), x - x^\star\rangle\\ &\quad = f(x) - f(x^\star),\end{aligned}$$

where we use the property that $\nabla f(x^\star) = 0$.

Thus, we get the desired result:

$$\mathbb{E}\big\|\nabla f_j(x) - \nabla f_j(x^\star)\big\|^2 \le 2L(f(x) - f(x^\star)).$$

□

Lemma 2. *For any f defined as $f := \frac{1}{n}\sum_{i=1}^{n} f_i$, where each f_i satisfies* (8) *and* (9)*, and for any x and i we have*

$$\big\|\nabla f_i(x) - \nabla f_i(x^\star)\big\|^2 \le \frac{2L^2}{\mu}\big(f(x) - f(x^\star)\big),$$

where $x^\star$ denotes the minimizer of f.

Proof. A standard result used frequently in the convex optimization literature is as follows:

$$\|\nabla f_i(x) - \nabla f_i(x^\star)\|^2 \le L^2\|x - x^\star\|^2,$$

where f_i is L-Lipschitz smooth. A proof for this inequality can be found in [32] (Theorem 2.1.5 on page 56).

From (8), we get:

$$\begin{aligned}\|x - x^\star\|^2 &\le \frac{2}{\mu}\big(f(x) - f(x^\star) - \langle x - x^\star, \nabla f(x^\star)\rangle\big)\\ &= \frac{2}{\mu}\big(f(x) - f(x^\star)\big),\end{aligned} \tag{10}$$

using the property that $\nabla f(x^\star) = 0$. The desired result follows immediately. □

B. Proof of Theorem 1

Proof. Let the update rule for *CentralVR* be denoted as

$$x_{m+1}^{k+1} = x_{m+1}^k - \eta v_{m+1}^k,$$

where we define:

$$v_{m+1}^k := \Big[\nabla f_{i_k}(x_{m+1}^k) - \nabla f_{i_k}(\tilde{x}_m^{i_k}) + \frac{1}{n}\sum_j \nabla f_j(\tilde{x}_m^j)\Big].$$

In this proof, we assume that the data indices are accessed randomly with replacement. Thus, $\tilde{x}_m^{i_k}$ denotes the last iterate when the i_k-th data index was chosen in or before the m-th epoch. Thus, conditioning on all x, v_{m+1}^k is an unbiased estimator of the true gradient at x_{m+1}^k, i.e., we get:

$$\begin{aligned}\mathbb{E}\Big[v_{m+1}^k\Big] &= \nabla f(x_{m+1}^k) - \frac{1}{n}\sum_j \nabla f_j(\tilde{x}_m^j) + \frac{1}{n}\sum_j \nabla f_j(\tilde{x}_m^j)\\ &= \nabla f(x_{m+1}^k).\end{aligned} \tag{11}$$

Conditioned on all history (all x), we first begin with the standard identity:

$$\begin{aligned}&\mathbb{E}\big[\|x_{m+1}^{k+1} - x^\star\|^2\big]\\ &= \mathbb{E}\Big[\|x_{m+1}^k - \eta v_{m+1}^k - x^\star\|^2\Big]\\ &= \|x_{m+1}^k - x^\star\|^2 - 2\eta(x_{m+1}^k - x^\star)^T\mathbb{E}[v_{m+1}^k] + \eta^2\mathbb{E}\|v_{m+1}^k\|^2\\ &= \|x_{m+1}^k - x^\star\|^2 - 2\eta(x_{m+1}^k - x^\star)^T\nabla f(x_{m+1}^k)\\ &\quad + \eta^2\mathbb{E}\|v_{m+1}^k\|^2,\end{aligned} \tag{12}$$

where we use (11).

We now bound (12). Using the definition of strong convexity in (8), we can simplify the inner product term in (12) as

$$\begin{aligned}&(x^\star - x_{m+1}^k)^T\nabla f(x_{m+1}^k)\\ &\quad \le -\big(f(x_{m+1}^k) - f(x^\star)\big) - \frac{\mu}{2}\|x^\star - x_{m+1}^k\|^2.\end{aligned} \tag{13}$$

We now bound the magnitude of the gradient term in (12):

$$\begin{aligned}&\mathbb{E}\|v_{m+1}^k\|^2\\ =&\mathbb{E}\Big\|\nabla f_{i_k}(x_{m+1}^k) - \nabla f_{i_k}(\tilde{x}_m^{i_k}) + \frac{1}{n}\sum_j \nabla f_j(\tilde{x}_m^j)\Big\|^2\\ =&\mathbb{E}\Big\|\nabla f_{i_k}(x_{m+1}^k) - \nabla f_{i_k}(x^\star) + \nabla f_{i_k}(x^\star)\\ &\quad - \nabla f_{i_k}(\tilde{x}_m^{i_k}) + \frac{1}{n}\sum_j \nabla f_j(\tilde{x}_m^j)\Big\|^2\\ \le&2\mathbb{E}\big\|\nabla f_{i_k}(x_{m+1}^k) - \nabla f_{i_k}(x^\star)\big\|^2 + 2\mathbb{E}\Big\|\nabla f_{i_k}(\tilde{x}_m^{i_k})\\ &\quad - \nabla f_{i_k}(x^\star) - \Big(\frac{1}{n}\sum_j \nabla f_j(\tilde{x}_m^j) - \frac{1}{n}\sum_j \nabla f_j(x^\star)\Big)\Big\|^2\\ =&2\mathbb{E}\big\|\nabla f_{i_k}(x_{m+1}^k) - \nabla f_{i_k}(x^\star)\big\|^2 + 2\mathbb{E}\Big\|\nabla f_{i_k}(\tilde{x}_m^{i_k})\\ &\quad - \nabla f_{i_k}(x^\star) - \mathbb{E}\Big[\nabla f_{i_k}(\tilde{x}_m^{i_k}) - \nabla f_{i_k}(x^\star)\Big]\Big\|^2\\ \le&2\mathbb{E}\big\|\nabla f_{i_k}(x_{m+1}^k) - \nabla f_{i_k}(x^\star)\big\|^2\\ &\quad + 2\mathbb{E}\big\|\nabla f_{i_k}(\tilde{x}_m^{i_k}) - \nabla f_{i_k}(x^\star)\big\|^2\\ \le&4L\big(f(x_{m+1}^k) - f(x^\star)\big) + \frac{4L^2}{\mu}\mathbb{E}\big(f(\tilde{x}_m^{i_k}) - f(x^\star)\big).\end{aligned} \tag{14}$$

The second equality uses the property that $\nabla f(x^\star) = 0$. The first inequality uses the property that $\|a+b\|^2 \le 2\|a\|^2 + 2\|b\|^2$. The second inequality uses $\mathbb{E}\|\phi - \mathbb{E}\phi\|^2 = \mathbb{E}\|\phi\|^2 - \|\mathbb{E}\phi\|^2 \le \mathbb{E}\|\phi\|^2$, for any random vector ϕ. The third inequality follows from Lemma 1 and Lemma 2.

We now plug (13) and (14) into (12) and rearrange to get

$$\begin{aligned}&\mathbb{E}\Big[\|x_{m+1}^{k+1} - x^\star\|^2\Big] + 2\eta(1 - 2L\eta)\big(f(x_{m+1}^k) - f(x^\star)\big)\\ &\le \big\|x_{m+1}^k - x^\star\big\|^2 - \eta\mu\big\|x_{m+1}^k - x^\star\big\|^2\end{aligned}$$

$$+\frac{4L^2\eta^2}{\mu}\mathbb{E}\big(f(\tilde{x}_m^{i_k}) - f(x^\star)\big). \tag{15}$$

Taking expectation on all x and summing (15) over all $k = 0, 1, \ldots, n-1$, we get a telescoping sum in $\|x_{m+1}^k - x^\star\|^2$ that yields:

$$\begin{aligned}
&\mathbb{E}\|x_{m+2}^0 - x^\star\|^2 + 2n\eta(1-2L\eta)\mathbb{E}\big(\overline{f(x_{m+1})} - f(x^\star)\big)\\
\leq&\mathbb{E}\|x_{m+1}^0 - x^\star\|^2 - \eta\mu\sum_{k=0}^{n-1}\mathbb{E}\|x_{m+1}^k - x^\star\|^2\\
&+\frac{4nL^2\eta^2}{\mu}\mathbb{E}\big(\overline{f(\tilde{x}_m)} - f(x^\star)\big),
\end{aligned} \tag{16}$$

where we use the convention $x_m^n = x_{m+1}^0$, and define $\overline{f(x_m)}$ as $\overline{f(x_m)} := \frac{1}{n}\sum_{k=0}^{n-1} f(x_m^k)$.

We now observe that

$$\mathbb{E}\|x_{m+1}^0 - x^\star\|^2 \leq \sum_{k=0}^{n-1}\mathbb{E}\|x_{m+1}^k - x^\star\|^2.$$

Thus we can rewrite

$$-\eta\mu\sum_{k=0}^{n-1}\mathbb{E}\|x_{m+1}^k - x^\star\|^2 \leq -\eta\mu\mathbb{E}\|x_{m+1}^0 - x^\star\|^2.$$

Substituting this in (16), we get:

$$\begin{aligned}
&\mathbb{E}\|x_{m+2}^0 - x^\star\|^2 + 2n\eta(1-2L\eta)\mathbb{E}\big(\overline{f(x_{m+1})} - f(x^\star)\big)\\
\leq&(1-\eta\mu)\mathbb{E}\|x_{m+1}^0 - x^\star\|^2 + \frac{4nL^2\eta^2}{\mu}\mathbb{E}\big(\overline{f(\tilde{x}_m)} - f(x^\star)\big).
\end{aligned}$$

We can rewrite this to get:

$$\begin{aligned}
&\mathbb{E}\|x_{m+2}^0 - x^\star\|^2 + 2n\eta(1-2L\eta)\mathbb{E}\big(\overline{f(x_{m+1})} - f(x^\star)\big)\\
\leq&\alpha\Big(\mathbb{E}\|x_{m+1}^0 - x^\star\|^2 + 2n\eta(1-2L\eta)\mathbb{E}\big(\overline{f(\tilde{x}_m)} - f(x^\star)\big)\Big),
\end{aligned}$$

where we define α as:

$$\alpha := \max\Big(1-\eta\mu, \frac{4nL^2\eta^2}{2n\mu\eta(1-2L\eta)}\Big).$$

The result immediately follows.

$\square$

References

[1] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

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

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

[4] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 2719–2727.

[5] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 873–881.

[6] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, 2014, pp. 19–27.

[7] O. Shamir and N. Srebro, "Distributed stochastic optimization and learning," in *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*. IEEE, 2014, pp. 850–857.

[8] M. Zinkevich, J. Langford, and A. J. Smola, "Slow learners are fast," in *Advances in Neural Information Processing Systems*, 2009, pp. 2331–2339.

[9] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., 1989.

[10] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[11] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Advances in Neural Information Processing Systems*, 2015, pp. 685–693.

[12] A. Mokhtari and A. Ribeiro, "Dsa: Decentralized double stochastic averaging gradient algorithm," *Journal of Machine Learning Research*, vol. 17, no. 61, pp. 1–35, 2016.

[13] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.

[14] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems*, 2014, pp. 1646–1654.

[15] S. J. Reddi, A. Hefny, S. Sra, B. Póczos, and A. J. Smola, "On variance reduction in stochastic gradient descent and its asynchronous variants," in *Advances in Neural Information Processing Systems*, 2015, pp. 2629–2637.

[16] N. L. Roux, M. Schmidt, and F. R. Bach, "A stochastic gradient method with an exponential convergence _rate for finite training sets," in *Advances in Neural Information Processing Systems*, 2012, pp. 2663–2671.

[17] A. Defazio, J. Domke *et al.*, "Finito: A faster, permutable incremental gradient method for big data problems," in *Proceedings of The 31st International Conference on Machine Learning*, 2014, pp. 1125–1133.

[18] J. Konečný, J. Liu, P. Richtárik, and M. Takáč, "ms2gd: Mini-batch semi-stochastic gradient descent in the proximal setting," *arXiv preprint arXiv:1410.4744*, 2014.

[19] J. Konečný and P. Richtárik, "Semi-stochastic gradient descent methods," *arXiv preprint arXiv:1312.1666*, 2013.

[20] L. Xiao and T. Zhang, "A proximal stochastic gradient method with progressive variance reduction," *SIAM Journal on Optimization*, vol. 24, no. 4, pp. 2057–2075, 2014.

[21] C. Wang, X. Chen, A. J. Smola, and E. P. Xing, "Variance reduction for stochastic gradient optimization," in *Advances in Neural Information Processing Systems*, 2013, pp. 181–189.

[22] R. Harikandeh, M. O. Ahmed, A. Virani, M. Schmidt, J. Konečný, and S. Sallinen, "Stop wasting my gradients: Practical svrg," in *Advances in Neural Information Processing Systems*, 2015, pp. 2242–2250.

[23] H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan, "Perturbed iterate analysis for asynchronous stochastic optimization," *arXiv preprint arXiv:1507.06970*, 2015.

[24] X. Pan, M. Lam, S. Tu, D. Papailiopoulos, C. Zhang, M. I. Jordan, K. Ramchandran, C. Re, and B. Recht, "Cyclades: Conflict-free asynchronous machine learning," *arXiv preprint arXiv:1605.09721*, 2016.

[25] L. Bottou, "Curiously fast convergence of some stochastic gradient descent algorithms," in *Proceedings of the symposium on learning and data science, Paris*, 2009.

[26] ——, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 421–436.

[27] M. Gürbüzbalaban, A. Ozdaglar, and P. Parrilo, "Why random reshuffling beats stochastic gradient descent," *arXiv preprint arXiv:1510.08560*, 2015.

[28] O. Shamir, "Without-replacement sampling for stochastic gradient methods: Convergence results and application to distributed optimization," *arXiv preprint arXiv:1603.00570*, 2016.

[29] D. Prokhorov, "Ijcnn 2001 neural network competition," *Slide presentation in IJCNN*, vol. 1, 2001.

[30] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[31] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, 2014.

[32] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.