Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Asynchronous Stochastic Proximal Optimization Algorithms with Variance Reduction

Qi Meng,[1]* Wei Chen,[2] Jingcheng Yu,[3]* Taifeng Wang,[2] Zhi-Ming Ma,[4] Tie-Yan Liu[2]
[1] School of Mathematical Sciences, Peking University, qimeng13@pku.edu.cn
[2]Microsoft Research, {wche, taifengw, tie-yan.liu}@microsoft.com
[3]Carnegie Mellon University, jingchey@cs.cmu.edu
[4]Academy of Mathematics and Systems Science, Chinese Academy of Sciences, mazm@amt.ac.cn

Abstract

Regularized empirical risk minimization (R-ERM) is an important branch of machine learning, since it constrains the capacity of the hypothesis space and guarantees the generalization ability of the learning algorithm. Two classic proximal optimization algorithms, i.e., proximal stochastic gradient descent (ProxSGD) and proximal stochastic coordinate descent (ProxSCD) have been widely used to solve the R-ERM problem. Recently, variance reduction technique was proposed to improve ProxSGD and ProxSCD, and the corresponding ProxSVRG and ProxSVRCD have better convergence rate. These proximal algorithms with variance reduction technique have also achieved great success in applications at small and moderate scales. However, in order to solve large-scale R-ERM problems and make more practical impacts, the parallel versions of these algorithms are sorely needed. In this paper, we propose asynchronous ProxSVRG (Async-ProxSVRG) and asynchronous ProxSVRCD (Async-ProxSVRCD) algorithms, and prove that Async-ProxSVRG can achieve near linear speedup when the training data is sparse, while Async-ProxSVRCD can achieve near linear speedup regardless of the sparse condition, as long as the number of block partitions are appropriately set. We have conducted experiments on a regularized logistic regression task. The results verified our theoretical findings and demonstrated the practical efficiency of the asynchronous stochastic proximal algorithms with variance reduction.

1 Introduction

In this paper, we focus on the regularized empirical risk minimization (R-ERM) problem, whose objective is a finite sum of smooth convex loss functions $f_i(x)$ plus a non-smooth regularization term $R(x)$, i.e.,

$$\min_{x\in\mathbb{R}^d} P(x) = F(x) + R(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) + R(x). \quad (1)$$

In particular, in the context of machine learning, $f_i(x)$ and $R(x)$ are defined as follows. Suppose we are given a collection of training data (a_1, b_1),...,(a_n, b_n), where each $a_i \in \mathbb{R}^d$ is an input feature vector and $b_i \in \mathbb{R}$ is the output variable. The loss function $f_i(x)$ measures the fitness of the model x on training data (a_i, b_i). Different learning tasks may use different loss functions, such as the least square loss $\frac{1}{2}(a_i^T x - b_i)^2$ for regression and the logistic loss $\log(1 + \exp(-b_i a_i^T x))$ for classification. The regularization term is used to constrain the capacity of the hypothesis space. For example, the non-smooth L_1 regularization term is widely used.

In order to solve the R-ERM problem, the proximal stochastic gradient descent method (ProxSGD) has been widely used, which exploits the additive nature of the empirical risk function and updates the model based on the gradient which is calculated at randomly sampled training data. However, the random sampling in ProxSGD introduces non-negligible variance, which makes that we need to use a decreasing step size to guarantee the algorithm's convergence, and the convergence rate is only sublinear (Langford, Li, and Zhang 2009; Rakhlin, Shamir, and Sridharan 2011). To tackle this problem, people have developed a set of new technologies. For example, in (Xiao and Zhang 2014), a variance reduction technique was introduced to improve ProxSGD and a new algorithm called ProxSVRG was proposed. It has been proven that even with a constant step size, ProxSVRG can achieve linear convergence rate.

Proximal stochastic coordinate descent (ProxSCD) is another method which is used to solve the R-ERM problem (Shalev-Shwartz and Tewari 2011). Since the variance introduced by the coordinate sampling asymptotically goes to zero, the ProxSCD attains linear convergence rate when the objective function $P(x)$ is strongly convex (Wright 2015). However, ProxSCD still requires that all component functions in the empirical risk are accessible in each iteration, which is time consuming. In (Zhao et al. 2014), a new algorithm called ProxSVRCD (also known as MRBCD) was proposed to improve ProxSCD. This algorithm, in addition to randomly samples a block of coordinates, also randomly samples training data in each iteration and uses the variance reduction technique. It has been proven that ProxSVRCD can achieve linear convergence rate and outperform ProxSCD by a lower iteration complexity.

While the aforementioned new algorithms (i.e., ProxSVRG and ProxSVRCD) have both good theoretical properties and empirical performances, the investigations on them were mainly conducted in the sequential (single-machine) setting. In this big data era, we usually need to deal with

*This work was done when the author was visiting Microsoft Research Asia.

very large scale R-ERM problem. In this case, sequential algorithms usually cost too much time. To tackle the challenge, parallelization of these algorithms is sorely needed. Recently literature researches in parallel methods tend to use asynchronous parallelization due to its high efficient in system (Dean et al. 2012; Recht et al. 2011). We are interested in asynchronous parallel implementations of the aforementioned stochastic proximal algorithms with variance reduction, which are, however, not well studied in the literature, to the best of our knowledge.

For asynchronous ProxSVRG (Async-ProxSVRG), we consider the *consistent read* setting, in which we ensure the atomic pull and push of the whole parameter for the local workers. For asynchronous ProxSVRCD (Async-ProxSVRCD), since the updates are performed over coordinate blocks, we only ensure the atomic pull and push of a coordinate block of the parameter for local workers for the sake of system efficiency. Comparing with Async-ProxSVRG setting, we name it as *inconsistent read* setting. We conduct theoretical analysis for the asynchronous algorithms. According to our results: (1) Async-ProxSVRG can achieve near linear speedup with respect to the number of local workers, when the input feature vectors are sparse; (2) If the data are non-sparse, ProxSVRCD can still achieve near linear speedup, when the block size is small comparing to the input dimension. The intuition of the linear speedup of the asynchronous proximal algorithms with variance reduction can be explained as follows. Asynchronous implementation updates the master parameter based on the delayed gradients. If the data are sparse for asynchronous ProxSVRG or the coordinate block size is small comparing to the input dimension for ProxSVRCD, the influence of the delayed gradients can be bounded, and the asynchronous implementations are roughly equivalent to the sequential version.

In addition to the theoretical analysis, we have also conducted experiments on benchmark datasets to test the performances of the asynchronous stochastic proximal algorithms with variance reduction. According to the experimental results, we have the following observations: (1) Async-ProxSVRG have good speedup, especially for sparse data; (2) Async-ProxSVRCD also have good speedup, and is more efficient than Async-ProxSVRG when the input feature vectors are relatively dense or the coordinate block size is small. (3) Async-ProxSVRG and Async-ProxSVRCD can converge faster than other asynchronous algorithms reported in literature such as Async-ProxSGD (Lian et al. 2015) and Async-ProxSCD (Liu and Wright 2015). The results are consistent across different datasets, indicating that our observations are general and the two asynchronous proximal algorithms are highly efficient and scalable for practical use.

This paper is organized as follows: in Section 2, we briefly introduce the stochastic proximal algorithms with variance reduction including ProxSVRG and ProxSVRCD, and then related works; in Section 3, we describe the asynchronous parallelization of these algorithms; in Section 4, we prove the convergence rates for Async-ProxSVRG and Async-ProxSVRCD; in Section 5, we report the experimental results and make discussions; finally, in the last section, we conclude the paper and present future research directions.

2 Background

In this section, we will briefly introduce proximal algorithms with variance reduction, and then review the existing convergence analysis for asynchronous parallel algorithms.

2.1 ProxSGD and ProxSCD

At first, let us briefly introduce the standard stochastic proximal gradient algorithms,i.e., ProxSGD and ProxSCD. With ProxSGD, at iteration k, the update rule to solve the R-ERM problem (i.e., Eqn (1)) is as follows:

$$x_{k+1} = prox_{\eta_k R}\left\{x_k - \eta_k \nabla f_{\mathcal{B}_k}(x_k))\right\}, \tag{2}$$

where η_k is the step size, $\mathcal{B}_k$ is a mini-batch of randomly selected training data, $\nabla f_{\mathcal{B}_k}(x_k) = \frac{1}{|\mathcal{B}_k|}\sum_{i\in\mathcal{B}_k}\nabla f_i(x_k)$ and the proximal mapping is defined as $prox_R(y) = argmin_{x\in\mathbb{R}^d}\left\{\frac{1}{2}\|x-y\|_2^2 + R(x)\right\}$.

ProxSCD exploits the block separability of the regularization term R in the R-ERM problem, i.e.,$R(x) = \sum_{j=1}^m R_j(x_{C_j})$, where x_{C_j} is the j-th coordinate block of x. For example, for the $L1$-norm regularizer, $\{C_j; j = 1,\cdots,m\}$ is a partition of $\{1,\cdots,d\}$ with $m = \frac{d}{block\ size}$, and $R_j(x_{C_j}) = \sum_{l\in C_j}|x_l|$, where l is the coordinate index. ProxSCD randomly selects a coordinate block and update the coordinates in that block based on their gradients while keeps the value of the other coordinates unchanged, i,e.,

$$x_{k+1,C_{j_k}} = prox_{\eta R_{j_k}}\left\{x_{k,C_{j_k}} - \eta\nabla_{C_{j_k}}F(x_{k-1}))\right\}, \tag{3}$$

where C_{j_k} is the coordinate block sampled at iteration k, and $\nabla_{C_j}F(x) = [\nabla F(x)]_{C_j}$.

2.2 Proximal Algorithms with Variance Reduction

For ProxSGD, the step size η_k has to be decreasing in order to mitigate the variance introduced by random sampling, which usually leads to slow convergence. To tackle this problem, one of the most popular variance reduction techniques was proposed by Johnson and Zhang (Johnson and Zhang 2013). Xiao and Zhang applied this variance reduction technique to improve ProxSGD, and a new algorithm called ProxSVRG was proposed (Xiao and Zhang 2014).

The ProxSVRG algorithm divides the optimization process into multiple stages. At the beginning of stage s, ProxSVRG calculates the full gradient at the current solution $\tilde{x}_{s-1}$, i.e., $\nabla F(\tilde{x}_{s-1})$. Then, at iteration k inside stage s, the solution is updated as follows:

$$v_k = \nabla f_{\mathcal{B}_k}(x_k) - \nabla f_{\mathcal{B}_k}(\tilde{x}_{s-1}) + \nabla F(\tilde{x}_{s-1}), \tag{4}$$

$$x_{k+1} = prox_{\eta_k R}\left\{x_k - \eta_k v_k\right\}, \tag{5}$$

where $-\nabla f_{\mathcal{B}_k}(\tilde{x}_{s-1}) + \nabla F(\tilde{x}_{s-1})$ is the variance reduction regularization term.

For ProxSCD, since the variance introduced by the block selection asymptotically goes to zero, it attains linear convergence rate. However, it still requires that all component functions are accessible within every iteration. Zhao $et.al.$ used variance reduction technique to improve ProxSCD with random training data sampling and a new algorithm called ProxSVRCD was proposed (Zhao et al. 2014). [1]

[1] In (Zhao et al. 2014), this algorithm was named MRBCD. In this paper, we call it ProxSVRCD to ease our reference.

ProxSVRCD is similar to ProxSVRG, the update formula for iteration k inside stage s takes the following form:

$$v_k = \nabla f_{\mathcal{B}_k}(x_k) - \nabla f_{\mathcal{B}_k}(\tilde{x}_{s-1}) + \nabla F(\tilde{x}_{s-1}), \quad (6)$$

$$x_{k+1,C_{j_k}} = prox_{\eta_k R_{j_k}}\left\{x_{k,C_{j_k}} - \eta_k v_{k,C_{j_k}}\right\}, \quad (7)$$

$$x_{k+1,\backslash C_{j_k}} \leftarrow x_{k,\backslash C_{j_k}}. \quad (8)$$

where $\backslash C_{j_k} = \{l : l \in \bigcup_{j \neq j_k} C_j\}$ and $-\nabla f_{\mathcal{B}_k}(\tilde{x}_{s-1}) + \nabla F(\tilde{x}_{s-1})$ is the variance reduction regularization term.

2.3 Existing Convergence Analysis of Asynchronous Parallel Algorithms

The asynchronous parallel methods have been successfully applied to accelerate many optimization algorithms including stochastic gradient descent (SGD)(Agarwal and Duchi 2011; Feyzmahdavian, Aytekin, and Johansson 2015; Recht et al. 2011; Mania et al. 2015), stochastic coordinate descent (SCD) (Liu et al. 2013; Liu and Wright 2015), stochastic dual coordinate ascent (SDCA) (Tran et al. 2015) and randomized Kaczmarz algorithm (Liu, Wright, and Sridhar 2014). However, to the best of our knowledge, the asynchronous parallel versions of ProxSVRG and ProxSVRCD are not well studied, as well as their theoretical properties.

We briefly review the works which are closely related to ours as follows. Reddi $et.al.$ studied asynchronous SVRG and proved that, asynchronous SVRG can achieve near linear speedup under some sparse condition (Reddi et al. 2015). Liu and Wright analyzed the asynchronous ProxSCD. They proved that the asynchronous ProxSCD can achieve near linear speedup if the delay is bounded by $\mathcal{O}(d^{\frac{1}{4}})$, where d is the input dimension (Liu and Wright 2015).

However, to the best of our knowledge, there is no study on the asynchronous parallel versions of proximal algorithms with variance reduction.

3 Asynchronous Proximal Algorithms with Variance Reduction

In this section, we describe our Async-ProxSVRG and Async-ProxSVRCD algorithms under the following asynchronous parallel architecture. Suppose there are P local workers and one master. For local workers, each of them has full access to the training data and stores a non-overlapping partition N_p $(p = 1, ..., P)$ of the training data. Each local worker independently communicates with the master to *pull* the global parameters from the master, and it computes the stochastic gradients locally and then *push* the gradients to the master. For the master, it maintains the global model. It updates the model parameters with the gradient pushed by local workers and sends the model parameters to local workers when it receives the pull request. Master can control the access conflict based on different granularity. In Async-ProxSVRG, the local worker will access the entire model in every update. Therefore, we let master only respond to one local worker's request at one time, which means the global model is atomic for all workers. In Async-ProxSVRCD, the local worker will only access a coordinate block in every update and different workers might work on different blocks without interfering with others. In this case, master will respond to multiple local workers simultaneously if only they are not accessing the same coordinate block, which means the global model is atomic at coordinate block level.

With variance reduction technique, the optimization process is divided into multiple stages (i.e., outer loop: $s = 1, \cdots, S$). In each stage, there are two phases: full gradient computation and solution updates (i.e., inner loop: $k = 1, \cdots, K$).

Full gradient computation: the workers collectively compute the full gradient in parallel based on the entire training data. Specifically, each worker pulls the master parameter from the master, computes the gradients over one part of the training data, and pushes the sum of the gradients to the master. Then the master aggregates the gradients from the workers to obtain the full gradient, and broadcasts it to the workers.

Solution updates: the workers compute the VR-regularized stochastic gradient in an asynchronous way and the master makes updates according to the proximal algorithms. To be specific, at iteration k, one local worker (who just finished its local computation) pulls the master parameters from the master, computes the VR-regularized stochastic gradient according to Eqn (4) for ProxSVRG or Eqn(6) for ProxSVRCD, and then pushes it to the master without any synchronization with the other workers. After the master receives the VR-regularized gradient from this worker, it updates the master parameter according to Eqn (5) for ProxSVRG or Eqn (7)(8) for Prox SVRCD. Then the global clock becomes $k+1$, and the next iteration begins. Corresponding details can be found in Algorithm 1.

Please note that, the gradient pushed by a local worker to the master could be delayed. The reason is, when the worker is working on its own local computation, other workers might finish their computations and push their gradients to the master, and the master updates the master parameter accordingly. As aforementioned, for Async-ProxSVRG, the whole model is atomic to each workers access. When the worker 0 is working on its own local computation, worker 1 and worker 2 might finish their computations, pushed their gradients to the master, and the master updates the master parameter accordingly. Thus, when worker 0 finish its computation and push it to the master, the global clock has already plus 2. Thus, the local gradients have delay=2 for the current master parameter. We use a random variable τ_k to denote the *delay* of local gradients received by the master at global clock k. The delay equals to the number of updates that other workers have committed to the master between one particular worker pulls the parameter from the master and pushes gradients to the master. For asynchronous ProxSVRCD, multiple workers may access the master parameter simultaneously, updating different coordinate blocks. Then different coordinate blocks in the model could be inconsistent regarding to the global update clock. To be precise, at global clock k, the master makes update based on the gradients computed by a local worker, who read the first coordinate block of the master parameter at global clock $k - \tau_k$. We denote the finally pulled parameter as $\hat{x}_k$, which can be represented as below:

$$\hat{x}_k = x_{k-\tau_k} + \sum_{h \in J(k)} (x_{h+1} - x_h), \quad (9)$$

where $J(k) \subset \{k-\tau_k,,k-1\}$. The k-th update can be described as $x_{k+1,C_{j_k}} = prox_{\eta_k R_{j_k}}\left\{x_{k,C_{j_k}} - \eta_k u_{k,C_{j_k}}\right\}$, where $u_k = \nabla f_{\mathcal{B}_k}(\hat{x}_k) - \nabla f_{\mathcal{B}_k}(\tilde{x}) + \nabla F(\tilde{x})$. The delay τ_k equals to the difference between the clock at which a local worker pulls the first coordinate block from the master and the clock at which the local worker pushes the gradients to the master.

We conduct theoretical analysis for Async-ProxSVRG and Async-ProxSVRCD based on the above setting in the next section. Like other asynchronous parallel algorithms, the delay also plays an important role in the convergence rate of asynchronous proximal algorithms with variance reduction.

Algorithm 1 Async-ProxSVRG and Async-ProxSVRCD

Require: initial vector $\tilde{x}_0$, step size η, number of inner loops K, size of mini-batch B, number of coordinate blocks m.
Ensure: $\tilde{x}_S$
  for $s = 1, 2, ..., S$ **do**
    $\tilde{x} = \tilde{x}_{s-1}, x_0 = \tilde{x}$
    For local worker p: calculate $\nabla F_p(\tilde{x}) = \sum_{i \in N_p} \nabla f_i(\tilde{x})$ and send it to the master.
    For master: calculate $\nabla F(\tilde{x}) = \frac{1}{n}\sum_{p=1}^{P} \nabla F_p(\tilde{x})$ and send it to each local worker.
    for $k = 1, ..., K$ **do**
      1. Async-ProxSVRG: consistent read
      For local worker p: randomly select a mini-batch $\mathcal{B}_k$ with $|\mathcal{B}_k| = B$.
      Pull current state $x_{k-\tau_k}$ from the master.
      Compute $u_k = \nabla f_{\mathcal{B}_k}(x_{k-\tau_k}) - \nabla f_{\mathcal{B}_k}(\tilde{x}) + \nabla F(\tilde{x})$.
      Push u_k to the master.
      For master:
      Update $x_{k+1} = prox_{\eta R}(x_k - \eta u_k)$.
      2. Async-ProxSVRCD: inconsistent read
      For local worker p: randomly select $\mathcal{B}_k$ with $|\mathcal{B}_k| = B$, and randomly select $j_k \in [m]$.
      Pull current state $\hat{x}_k$ from the master.
      Compute $u_k = \nabla f_{\mathcal{B}_k}(\hat{x}_k) - \nabla f_{\mathcal{B}_k}(\tilde{x}) + \nabla F(\tilde{x})$.
      Push u_k to the master.
      For master:
      Update $x_{k+1,C_{j_k}} = prox_{\eta R_{j_k}}\left\{x_{k,C_{j_k}} - \eta u_{k,C_{j_k}}\right\}$; $x_{k+1,\backslash C_{j_k}} \leftarrow x_{k,\backslash C_{j_k}}$
    end for
    $\tilde{x}_s = \frac{1}{K}\sum_{k=1}^{K} x_k$
  end for

4 Convergence Analysis

In this section, we prove the convergence rates of the asynchronous parallel proximal algorithms with variance reduction introduced in the previous section.

4.1 Async-ProxSVRG

At first, we introduce the following assumptions, which are very common in the theoretical analysis for asynchronous parallel algorithms (Recht et al. 2011; Reddi et al. 2015).

Assumption 1: (Convexity) $F(x)$ and $R(x)$ are convex and $R(x)$ is block sparable. The objective function $P(x)$ is μ-strongly convex, i.e., $\forall x, y \in \mathbb{R}^d$, we have,

$$P(y) \geq P(x) + \xi^T(y-x) + \frac{\mu}{2}\|y-x\|^2, \forall \xi \in \partial P(x).^2$$

Assumption 2: (Smoothness) The components $\{f_i(x); i \in [n]\}$ of $F(x)$ are differentiable and have Lipschitz continuous partial gradients, i.e., $\exists T, L > 0$, such that $\forall x, y \in \mathbb{R}^d$ with $x_j \neq y_j$, we have

$$\|\nabla_j f_i(x) - \nabla_j f_i(y)\| \leq T\|x_j - y_j\|, \forall i \in [n], j \in [d].$$
$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x-y\|, \forall i.$$

Assumption 3: (Bounded and Independent Delay) The random delay variables $\tau_1, \tau_2, ...$ in consistent read setting are independent of each other and independent of $\mathcal{B}_k$, and their expectations are upper bounded by τ, i.e., $\mathbb{E}\tau_k \leq \tau$.

Assumption 4: (Data Sparsity) The maximal frequency of a feature appearing in the dataset is upper bounded by Δ.

Based on these assumptions, we prove that Async-ProxSVRG has linear convergence rate.

Theorem 4.1 *Suppose Assumptions 1-4 hold. If the step size* $\eta < \min\left\{\frac{1}{4L\Delta\tau^2}, \frac{1}{L\tau\sqrt{8B\Delta}}, \frac{B}{16L}\right\}$, *and the inner loop size* K *is sufficiently large so that*

$$\rho = \frac{B}{\eta\mu K(B-8\eta L)} + \frac{8\eta L}{(B-8\eta L)} < 1,$$

then Async-ProxSVRG has linear convergence rate in expectation:

$$\mathbb{E}P(\tilde{x}_s) - P(x^*) \leq \rho^s[P(\tilde{x}_0) - P(x^*)],$$

where $x^* = argmin_x P(x)$.

Due to space limitation, we only provide the proof sketch and put the proof details into supplementary materials.

Proof Sketch of Theorem 4.1:

Firstly we introduce some notations. Let $x_{k+1} - x_k = -\eta g_k$, $v_k = \nabla f_{\mathcal{B}_k}(x_k) - \nabla f_{\mathcal{B}_k}(\tilde{x}) + \nabla F(\tilde{x})$, and $u_k = \nabla f_{\mathcal{B}_k}(x_{k-\tau_k}) - \nabla f_{\mathcal{B}_k}(\tilde{x}) + \nabla F(\tilde{x})$.

Step 1: By the sparsity assumption, we have

$$F(x) \geq F(y) - \nabla F(x)(y-x) - \frac{L\Delta}{2}\|x-y\|^2. \quad (10)$$

Step 2: By using the convexity assumption and Ineq.(10), we have:

$$P(x^*) \geq P(x_{k+1}) + (u_k - \nabla F(x_{k-\tau_k}))^T(x_{k+1} - x^*) +$$
$$\eta\|g_k\|^2 - \frac{L\eta^2\Delta\tau_k}{2}\sum_{h=k-\tau_k}^{k}\|g_h\|^2 + g_k^T(x^* - x_{k+1}). \quad (11)$$

Step 3: We use Lemma 3 in (Xiao and Zhang 2014) to bound the term $\mathbb{E}_{\mathcal{B}_k}(u_k - \nabla F(x_{k-\tau_k}))^T(x_{k+1} - x^*)$ in Ineq.(11).

Step 4: By following the proof of ProxSVRG, and set η that satisfies $\eta < \min\{\frac{1}{4L\Delta\tau^2}, \frac{1}{L\tau\sqrt{8B\Delta}}, \frac{B}{16L}\}$, we can get the results.

[2] In this paper, if there is no specification, $\|\cdot\|$ is the L_2-norm.

Remark: Theorem 4.1 actually shows that, Async-ProxSVRG can achieve linear speedup when Δ is small and $\tau \leq \sqrt{8/B^2\Delta}$. $L/\mu = \sqrt{n}$(Shamir, Srebro, and Zhang 2014). For sequential ProxSVRG, with step size $\eta = 0.1B/L$, the inner loop size K should be in the same order of $\mathcal{O}(L/B\mu)$ to make $\rho < 1$. The computation complexity (number of gradients need to calculate) for the inner loop is in the same order of $\mathcal{O}(L/\mu)$. For the Async-ProxSVRG, with $\eta = \min\{\frac{1}{4L\Delta\tau^2}, \frac{1}{L\tau\sqrt{8B\Delta}}, \frac{0.05B}{L}\}$, the inner loop size K should be in the same order of $\mathcal{O}(\Delta\tau^2 L/\mu + \tau\sqrt{8B\Delta}L/\mu + L/B\mu)$ to make $\rho < 1$.

For the case $\tau \leq \min\{\sqrt{50/B^3\Delta}, \sqrt{5/B\Delta}\}$ (i.e., $\frac{0.05B}{L}$ is smaller than the other two), by setting $\eta = \frac{0.05B}{L}$, the order of inner loop size K is $\mathcal{O}(L/B\mu)$ and the corresponding computation complexity is $\mathcal{O}(L/\mu)$, which is the same as the sequential ProxSVRG. Therefore, Async-ProxSVRG can achieve nearly the same performance as the sequential version, but τ times faster since we are running the algorithm asynchronously, and thus we achieve "linear speedup".

For the case $\tau \geq \min\{\sqrt{50/B^3\Delta}, \sqrt{5/B\Delta}\}$, the inner loop size K should be in the same order of $\mathcal{O}((B\Delta\tau + B\sqrt{8B\Delta})\tau L/B\mu)$. Compared with the sequential ProxSVRG, Async-ProxSVRG can not obtain linear speedup but still have a theoretical speedup of $1/(B\Delta\tau + B\sqrt{8B\Delta})$ if $B\Delta\tau + B\sqrt{8B\Delta} < 1$.

According to Theorem 4.1 and the above discussions, we provide the following corollary for a setup of the parameters in Async-ProxSVRG which can achieve near linear speedup.

Corollary 4.2 *Suppose Assumptions 1-4 hold. If we set* $B = \left(\frac{1}{\Delta}\right)^{\frac{1}{6}}$, $\tau \leq \sqrt{5/\Delta^{\frac{1}{2}}}$, $\eta = \frac{0.05\Delta^{\frac{1}{6}}}{L}$ *and* $K = \frac{200L\Delta^{\frac{1}{6}}}{\mu}$, *then Async-ProxSVRG has the following linear convergence rate:*

$$\mathbb{E}P(\tilde{x}_s) - P(x^*) \leq \left(\frac{5}{6}\right)^s [P(\tilde{x}_0) - P(x^*)],$$

where $x^* = argmin_x P(x)$.

4.2 Async-ProxSVRCD

In this section, we present Theorem 4.3, which states the convergence rate of Async-ProxSVRCD, as well as the conditions for them to achieve near linear speedup.

Assumption 3′:(Bounded and Independent Delay) The random delay variables $\tau_1, \tau_2, ...$ in inconsistent read setting in Eqn 9 are independent of each other and independent of $\mathcal{B}_k$, and their expectations are upper bounded by τ.

Theorem 4.3 *Suppose Assumptions 1, 2, and 3′ hold. In addition, we assume that the mini-batch size* $B \geq L/T$, *the step size* η *and the coordinate block number* m *satisfies* $\eta < \min\left\{\frac{1}{T}\frac{m^{\frac{3}{2}}-T\tau}{m^{\frac{3}{2}}+3m\tau+\tau^2}, \frac{1}{8T}, \frac{\mu\sqrt{m}}{2T\tau}\right\}$, *and the inner loop size* K *is sufficiently large so that*

$$\rho = \left(\frac{m}{\eta\mu K(1 - \frac{T\eta\tau}{\mu\sqrt{m}} - 4\eta T)} + \frac{4\eta T(K+1)}{(1 - \frac{T\eta\tau}{\mu\sqrt{m}} - 4\eta T)K}\right) < 1,$$

then Async-ProxSVRCD has linear convergence in expectation:

$$\mathbb{E}P(\tilde{x}_s) - P(x^*) \leq \rho^s [P(\tilde{x}_0) - P(x^*)],$$

where $x^* = argmin_x P(x)$.

Proof Sketch of Theorem 4.3:

We still use the notations g_k, u_k, v_k defined before. Let $\bar{x}_{k+1} = argmin_{x\in R^d}\left\{\frac{1}{2}\|x - x_k - \eta u_k\|^2 + \eta R(x)\right\}$. Recalling the update rule for x_k in Alg.1 and taking expectation with respect to j_k, we have

$$\mathbb{E}_{j_k}(x_{k+1} - x_k) = \frac{1}{m}(\bar{x}_{k+1} - x_k) \tag{12}$$

$$\mathbb{E}_{j_k}\|x_{k+1} - x_k\|^2 = \frac{1}{m}\|\bar{x}_{k+1} - x_k\|^2. \tag{13}$$

Step 1: By using Assumption 1, Eq.(12), Eq.(13), we have

$$\begin{aligned} &P(x^*) \geq \\ &m\mathbb{E}_{j_k}P(x_{k+1}) - (m-1)P(x_k) + (\eta - \frac{T\eta^2}{2})m\mathbb{E}_{j_k}\|g_k\|^2 \\ &+(v_k - \nabla F(x_k))^T(\bar{x}_{k+1} - x^*) + m\mathbb{E}_{j_k}(g_k)^T(x^* - x_k) \\ &+(\nabla f_{\mathcal{B}_k}(\hat{x}_k) - \nabla f_{\mathcal{B}_k}(x_k))^T(\bar{x}_{k+1} - x^*). \end{aligned}$$

Step 2: We decompose the term $-(\nabla f_{\mathcal{B}_k}(\hat{x}_k) - \nabla f_{\mathcal{B}_k}(x_k))^T(\bar{x}_{k+1} - x^*)$ and show that this term can be upper bounded by:

$$\begin{aligned} &\sum_{a=k-\tau}^{k-1}(\|x_{a+1} - x_a\|\|\bar{x}_{k+1} - x_k\| + \|x_{a+1} - x_a\|\|x_a - x^*\|) \\ &+ \sum_{a=k-\tau}^{k-1}\sum_{b=a}^{k-1}\|x_{a+1} - x_a\|\|x_{b+1} - x_b\|. \end{aligned}$$

By taking expectation w.r.t $j_1, \cdots, j_k$ gradually and using Eq.(12),Eq.(13), we can bound the three terms by considering the independent property (Assumption 3′). This is a key step for the proof and please see the details in the supplementary materials.

Step 3: Under the condition about η, and following the proof of ProxSVRCD, we can get the results.

Remark: Theorem 4.3 actually shows that when m is large (or equivalent the block size is small) and $\tau \leq \min\left\{\sqrt{m}, 4\mu\sqrt{m}, m^{\frac{3}{2}}/2T\right\}$, Async-ProxSVRCD can achieve linear speedup. For the sequential ProxSVRCD, Corollary 4.3 in (Zhao et al. 2014) set $\eta = 1/16T$, $B = L/T$ and the inner loop size K in the same order of $\mathcal{O}(mT/\mu)$ to make $\rho < 1$. For Async-ProxSVRCD, if m is sufficiently large so that the delay satisfies $\tau \leq \min\left\{\sqrt{m}, 4\mu\sqrt{m}, m^{\frac{3}{2}}/2T\right\}$, we can set $\eta = 1/24T$ which guarantees the condition $\eta < \min\left\{\frac{1}{T}\frac{m^{\frac{3}{2}}-T\tau}{m^{\frac{3}{2}}+3m\tau+\tau^2}, \frac{1}{8T}, \frac{\mu\sqrt{m}}{2T\tau}\right\}$. Thus, the inner loop size K should be $\mathcal{O}(mT/\mu)$ to make $\rho < 1$, which is the same as sequential ProxSVRCD. Therefore, Async-ProxSVRCD can achieve near linear speedup. If we consider the indicative case (Shamir, Srebro, and Zhang 2014) in which $L/\mu = \sqrt{n}$, $L = \mathcal{O}(1)$ and $\mu = \mathcal{O}(\sqrt{1/n})$. The condition for the linear speedup can be simplified to $\tau \leq 4\sqrt{m/n}$. Even if $4\sqrt{m/n} < \tau \leq \sqrt{m}$, Async-ProxSVRCD still have a speedup of $\mathcal{O}(\sqrt{m/n})$ by setting $\eta \leq \frac{\mu\sqrt{m}}{2T\tau} = \frac{\sqrt{m}}{2\tau\sqrt{n}}$, since $\frac{m^{\frac{3}{2}}-T\tau}{m^{\frac{3}{2}}+3m\tau+\tau^2} > \frac{\sqrt{m}}{2\tau\sqrt{n}}$.

According to Theorem 4.3 and the above discussions, we provide the following corollary for a setup of the parameters in Async-ProxSVRCD which achieves near linear speedup.

Corollary 4.4 *Suppose Assumptions 1,2, and 3′ hold and the delay bound satisfies* $\tau \leq \min\left\{\sqrt{m}, 4\mu\sqrt{m}, m^{\frac{3}{2}}/2T\right\}$. *Let* $\eta = 1/24T$, $B = L/T$ *and* $K = \frac{216mT}{\mu}$, *then Async-ProxSVRCD has the following linear convergence rate:*

$$\mathbb{E}P(\tilde{x}_s) - P(x^*) \leq \left(\frac{5}{6}\right)^s [P(\tilde{x}_0) - P(x^*)],$$

where $x^* = argmin_x P(x)$.

By comparing the conditions of the linear speedup for asynchronous Proximal algorithms, we have the following findings: (1) Async-ProxSVRG relies on the data sparsity to alleviate the negative impact of communication delay τ; (2) Async-ProxSVRCD does not rely on the sparsity condition, however, it requires the block size is small or the input dimension is large, since in this way, the block-wise updates will become frequent and can also alleviate the delay of the whole parameter vector.

To sum up, based on a few widely used assumptions, we have proven the convergence properties of the asynchronous parallel implementations of ProxSVRG, and ProxSVRCD, and discussed the conditions for them to achieve near linear speedups as compared to their sequential (single-machine) counterparts. In the next section, we will report the results of our experiments to verify these theoretical findings.

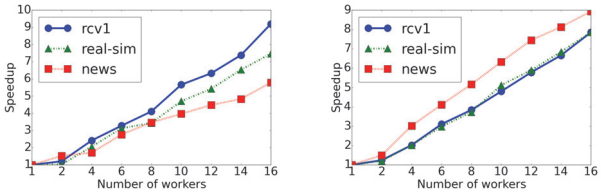
5 Experiments

In this section, we report our experimental results on the efficiency of the asynchronous proximal algorithms with variance reduction. In particular, we conducted binary classifications on three benchmark datasets: *rcv1, real-sim, news20* (Reddi et al. 2015), *new20* is the densest one with a much higher dimension and *rcv1* is the sparsest one. The detailed information about the three data sets is given in Table 1. We use the logistic loss function with both L_1 and L_2 regularizations with weight λ_1 and λ_2 respectively.

Table 1: Experimental Datasets

Dataset	*rcv1*	*real-sim*	*news20*
Data size n	20242	72309	19996
Feature size d	47236	20958	1355191
λ_1, λ_2	$10^{-5}, 10^{-4}$	$10^{-4}, 10^{-4}$	$10^{-6}, 10^{-4}$

Following the practices in (Xiao and Zhang 2014), we normalized the input vector of each data set before feeding it into the classifier, which leads to an upper bound of 0.25 for the Lipschitz constant L. The stopping criterion for all the algorithms under investigation is the optimization error smaller than 10^{-10} (i.e., $P(\tilde{x}_S) - P(x^*) < 10^{-10}$). For Async-ProxSVRG, we set step size $\eta = 0.04$, the mini-batch size $B = 200$, and the inner loop size $K = 2n$, where n is the data size. For Async-ProxSVRCD, we set step size $\eta = 0.04$, the number of block partitions $m = \frac{d}{100}$, the mini-batch size $B = 200$, and a larger inner loop size $K = 2nm$.



(a) Async-ProxSVRG (b) Async-ProxSVRCD

Figure 1: Results for the speedups of asynchronous algorithms

We implement Async-ProxSVRG and Async-ProxSVRCD in the consistent read setting and the inconsistent read setting, respectively.

Figures 1(a) and 1(b) show the speedups of Async-ProxSVRG and Async-ProxSVRCD, respectively. From the figures, we have the following observations. (1) On all the three datasets, Async-ProxSVRG has near linear speedup compared to its sequential counterpart. The speedup on *rcv1* is the largest, while that on *news20* is the smallest. This observation is consistent with our theoretical findings that Async-ProxSVRG has better performance on sparser data. (2) Async-ProxSVRCD also achieves nice speedup. The speedup is more significant for *news20* than that for the other two data sets. This is consistent with our theoretical discussions - the sufficient condition for the linear speedup of Async-ProxSVRCD is easier to be satisfied for high-dimensional datasets. As literature also reported other asynchronous algorithms, such as Async-ProxSGD and Async-ProxSCD, we also compare with them to test the performance of our algorithms. For saving space, we put the detailed results in the supplementary materials.

In summary, our experimental results well validate our theoretical findings, and indicate that the asynchronous proximal algorithms with variance reduction are very efficient and could have good applications in practice.

6 Conclusion

In this paper, we have studied the asynchronous parallelization of two widely used proximal gradient algorithms with variance reduction, i.e., ProxSVRG and ProxSVRCD. We have proved their convergence rates, discussed their speedups, and verified our theoretical findings through experiments. Overall speaking, these asynchronous proximal algorithms can achieve linear speedup under certain conditions, and can be highly efficient when being used to solve large scale R-ERM problems. As for future work, we plan to make the following explorations. First, we will extend the study in this paper to the non-convex case, both theoretically and experimentally. Second, we will study the asynchronous parallelization of more proximal algorithms.

7 Acknowledgments

Zhi-Ming Ma was partially supported by National Center for Mathematics and Interdisciplinary Sciences (NCMIS) of China and NSF of China (11526214).

References

Agarwal, A., and Duchi, J. C. 2011. Distributed delayed stochastic optimization. In *NIPS*, 873–881.

Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Senior, A.; Tucker, P.; Yang, K.; Le, Q. V.; et al. 2012. Large scale distributed deep networks. In *NIPS*, 1223–1231.

Feyzmahdavian, H. R.; Aytekin, A.; and Johansson, M. 2015. An asynchronous mini-batch algorithm for regularized stochastic optimization. *arXiv preprint arXiv:1505.04824*.

Johnson, R., and Zhang, T. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, 315–323.

Langford, J.; Li, L.; and Zhang, T. 2009. Sparse online learning via truncated gradient. In *NIPS*, 905–912.

Lian, X.; Huang, Y.; Li, Y.; and Liu, J. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *NIPS*, 2719–2727.

Liu, J., and Wright, S. J. 2015. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization* 25(1):351–376.

Liu, J.; Wright, S. J.; Ré, C.; Bittorf, V.; and Sridhar, S. 2013. An asynchronous parallel stochastic coordinate descent algorithm. *arXiv preprint arXiv:1311.1873*.

Liu, J.; Wright, S. J.; and Sridhar, S. 2014. An asynchronous parallel randomized kaczmarz algorithm. *arXiv preprint arXiv:1401.4780*.

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

Rakhlin, A.; Shamir, O.; and Sridharan, K. 2011. Making gradient descent optimal for strongly convex stochastic optimization. *arXiv preprint arXiv:1109.5647*.

Recht, B.; Re, C.; Wright, S.; and Niu, F. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 693–701.

Reddi, S. J.; Hefny, A.; Sra, S.; Póczos, B.; and Smola, A. J. 2015. On variance reduction in stochastic gradient descent and its asynchronous variants. In *NIPS*, 2629–2637.

Shalev-Shwartz, S., and Tewari, A. 2011. Stochastic methods for l 1-regularized loss minimization. *The Journal of Machine Learning Research* 12:1865–1892.

Shamir, O.; Srebro, N.; and Zhang, T. 2014. Communication-efficient distributed optimization using an approximate newton-type method. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1000–1008.

Tran, K.; Hosseini, S.; Xiao, L.; Finley, T.; and Bilenko, M. 2015. Scaling up stochastic dual coordinate ascent. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1185–1194. ACM.

Wright, S. J. 2015. Coordinate descent algorithms. *Mathematical Programming* 151(1):3–34.

Xiao, L., and Zhang, T. 2014. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization* 24(4):2057–2075.

Zhao, T.; Yu, M.; Wang, Y.; Arora, R.; and Liu, H. 2014. Accelerated mini-batch randomized block coordinate descent method. In *NIPS*, 3329–3337.