SIAM J. OPTIM.
Vol. 27, No. 4, pp. 2202–2229

PERTURBED ITERATE ANALYSIS FOR ASYNCHRONOUS STOCHASTIC OPTIMIZATION*

HORIA MANIA†, XINGHAO PAN†, DIMITRIS PAPAILIOPOULOS†, BENJAMIN RECHT‡, KANNAN RAMCHANDRAN§, AND MICHAEL I. JORDAN§

Abstract. We introduce and analyze stochastic optimization methods where the input to each update is perturbed by bounded noise. We show that this framework forms the basis of a unified approach to analyzing asynchronous implementations of stochastic optimization algorithms, by viewing them as serial methods operating on noisy inputs. Using our perturbed iterate framework, we provide new analyses of the HOGWILD! algorithm and asynchronous stochastic coordinate descent that are simpler than earlier analyses, remove many assumptions of previous models, and in some cases yield improved upper bounds on the convergence rates. We proceed to apply our framework to develop and analyze KROMAGNON: a novel, parallel, sparse stochastic variance-reduced gradient (SVRG) algorithm. We demonstrate experimentally on a 16-core machine that the sparse and parallel version of SVRG is in some cases more than four orders of magnitude faster than the standard SVRG algorithm.

Key words. stochastic optimization, asynchronous algorithms, parallel machine learning

AMS subject classifications. 65K10, 65Y05, 68W10, 68W20

DOI. 10.1137/16M1057000

1. Introduction.

Asynchronous parallel stochastic optimization algorithms have recently gained significant traction in algorithmic machine learning. A large body of recent work has demonstrated that near-linear speedups are achievable, in theory and practice, on many common machine learning tasks [12, 17, 22, 26, 29, 32, 33, 34]. Moreover, when these lock-free algorithms are applied to nonconvex optimization, significant speedups are still achieved with no significant loss of statistical accuracy. This behavior has been demonstrated in practice in state-of-the-art deep learning systems such as Google's Downpour SGD [11] and Microsoft's Project Adam [10].

Although asynchronous stochastic algorithms are simple to implement and enjoy excellent performance in practice, they are challenging to analyze theoretically. The current analyses require lengthy derivations and several assumptions that may not reflect realistic system behavior. Moreover, due to the difficult nature of the proofs, the algorithms analyzed are often simplified versions of those actually run in practice.

In this paper, we propose a general framework for deriving convergence rates for parallel, lock-free, asynchronous first-order stochastic algorithms. We interpret the algorithmic effects of asynchrony as perturbing the stochastic iterates with bounded noise. This interpretation allows us to show how a variety of asynchronous first-order algorithms can be analyzed as their serial counterparts operating on noisy inputs. The advantage of our framework is that it yields elementary convergence proofs, can

*Received by the editors January 19, 2016; accepted for publication (in revised form) June 30, 2017; published electronically October 10, 2017.
http://www.siam.org/journals/siopt/27-4/M105700.html
Funding: The work of the fourth author was funded by the United States Navy, Office of Naval Research (ONR), grants N00014-15-1-2620, N00014-13-1-0129, and N00014-14-1-0024, as well as by the National Science Foundation, grants CCF-1148243 and CCF-1217058.
†AMPLab and EECS, UC Berkeley, Berkeley, CA 94720 (hmania@eecs.berkeley.edu, xinghao@berkeley.edu, dimitrisp@berkeley.edu).
‡AMPLab, EECS, and Statistics, UC Berkeley, Berkeley, CA 94720 (brecht@berkeley.edu, jordan@cs.berkeley.edu).
§EECS, UC Berkeley, Berkeley, CA 94720 (kannanr@eecs.berkeley.edu).

remove or relax simplifying assumptions adopted in prior art, and can yield improved bounds when compared to earlier work.

We demonstrate the general applicability of our framework by providing new convergence analyses for HOGWILD!, i.e., the asynchronous stochastic gradient method (SGM), asynchronous stochastic coordinate descent (ASCD), and KROMAGNON: a novel asynchronous sparse version of the stochastic variance-reduced gradient (SVRG) method [19]. In particular, we provide a modified version of SVRG that allows for sparse updates, we show that this method can be parallelized in the asynchronous model, and we provide convergence guarantees using our framework. Experimentally, the asynchronous, parallel sparse SVRG achieves nearly linear speedups on a machine with 16 cores and is sometimes four orders of magnitude faster than the standard (dense) SVRG method.

1.1. Related work. The algorithmic tapestry of parallel stochastic optimization is rich and diverse, extending back at least to the late 1960s [9]. Much of the contemporary work in this space is built upon the foundational work of Bertsekas and Tsitsiklis [4] and Bertsekas, Tsitsiklis, and Athans [31]; in particular, the shared memory access model that we use in this work is inspired by the partially asynchronous model introduced in those manuscripts. Recent advances in parallel and distributed computing technologies have generated renewed interest in the theoretical understanding and practical implementation of parallel stochastic algorithms [2, 14, 18, 30, 35, 36].

The power of lock-free, asynchronous stochastic optimization on shared-memory multicore systems was first demonstrated in the work of [26]. The authors introduce HOGWILD!, a completely lock-free and asynchronous parallel SGM that exhibits nearly linear speedups for a variety of machine learning tasks. Inspired by HOGWILD!, several authors developed lock-free and asynchronous algorithms that move beyond SGM, such as the work of Liu et al. on parallel stochastic coordinate descent [22, 23]. Additional work on first-order methods and higher-order methods [12, 13, 16, 17, 32], with extensions to parallel iterative linear solvers [3, 24], has exhibited the wide range of settings for which linear speedups are possible in the asynchronous shared memory model.

2. Perturbed stochastic gradients.

Preliminaries and notation. We study parallel asynchronous iterative algorithms that minimize convex functions, $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^d$. We focus our analysis on functions f that are L-smooth and m-strongly convex. A function f is L-smooth if it is differentiable and has Lipschitz gradients

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\| \text{ for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^d,$$

where $\|\cdot\|$ denotes the Euclidean norm. Strong convexity with parameter $m > 0$ imposes a curvature condition on f:

$$f(\mathbf{x}) \geq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle + \frac{m}{2}\|\mathbf{x} - \mathbf{y}\|^2 \text{ for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^d.$$

Strong convexity implies that f has a unique minimum $\mathbf{x}^*$ and satisfies

$$\langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq m\|\mathbf{x} - \mathbf{y}\|^2.$$

In the following, we use i, j, and k to denote iteration counters, while reserving v and u to denote coordinate indices. We use $\mathcal{O}(1)$ to denote positive absolute constants.

The computational model considered in sections 3, 4, and 5 is the same as that of Niu et al. [26]: a number of cores have access to the same shared memory, and each of them can read and update components of $\mathbf{x}$ in the shared memory. The algorithms that we consider are asynchronous and lock-free: cores do not coordinate their reads or writes, and while a core is reading/writing other cores can update the shared variables in $\mathbf{x}$.

2.1. Perturbed iterates. A popular way to minimize convex functions is via *first-order stochastic* algorithms. These algorithms can be described using the following general iterative expression:

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \gamma \mathbf{g}(\mathbf{x}_j, s_j), \tag{2.1}$$

where s_j is a random variable independent of $\mathbf{x}_j$, and $\mathbf{g}$ is an unbiased estimator of the true gradient of f at $\mathbf{x}_j$: $\mathbb{E}_{s_j}\mathbf{g}(\mathbf{x}_j, s_j) = \nabla f(\mathbf{x}_j)$. The success of first-order stochastic techniques partly lies in their computational efficiency: the computational cost of using noisy gradient estimates is dwarfed by that of using true gradients.

A major advantage of the iterative formula in (2.1) is that—in combination with strong convexity and smoothness inequalities—one can easily track algorithmic progress and establish convergence rates to the optimal solution. Unfortunately, the progress of asynchronous parallel algorithms cannot be precisely described or analyzed using the above iterative framework. Processors do not read from memory actual iterates $\mathbf{x}_j$, as there is no global clock that synchronizes reads or writes while different cores write/read "stale" variables.

In subsequent sections, we show that the following simple perturbed variant of (2.1) can capture the algorithmic progress of asynchronous stochastic algorithms. Consider the following iteration:

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \gamma \mathbf{g}(\mathbf{x}_j + \mathbf{n}_j, s_j), \tag{2.2}$$

where $\mathbf{n}_j$ is a stochastic error term. Letting $\hat{\mathbf{x}}_j = \mathbf{x}_j + \mathbf{n}_j$, we have

$$\begin{aligned} \|\mathbf{x}_{j+1} - \mathbf{x}^*\|^2 &= \|\mathbf{x}_j - \gamma \mathbf{g}(\hat{\mathbf{x}}_j, s_j) - \mathbf{x}^*\|^2 \\ &= \|\mathbf{x}_j - \mathbf{x}^*\|^2 - 2\gamma\langle \mathbf{x}_j - \mathbf{x}^*, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle + \gamma^2\|\mathbf{g}(\hat{\mathbf{x}}_j, s_j)\|^2 \\ &= \|\mathbf{x}_j - \mathbf{x}^*\|^2 - 2\gamma\langle \hat{\mathbf{x}}_j - \mathbf{x}^*, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle \\ &\quad + \gamma^2\|\mathbf{g}(\hat{\mathbf{x}}_j, s_j)\|^2 + 2\gamma\langle \hat{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle, \end{aligned} \tag{2.3}$$

where in the last equation we added and subtracted the term $2\gamma\langle \hat{\mathbf{x}}_j, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle$.

We assume that $\hat{\mathbf{x}}_j$ and s_j are independent. However, in contrast to recursion (2.1), we no longer require $\mathbf{x}_j$ to be independent of s_j. The importance of this set of assumptions will become clear shortly.

We now take the expectation of both sides of (2.3). Since $\hat{\mathbf{x}}_j$ is independent of s_j, we use iterated expectations to obtain $\mathbb{E}\langle \hat{\mathbf{x}}_j - \mathbf{x}^*, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle = \mathbb{E}\langle \hat{\mathbf{x}}_j - \mathbf{x}^*, \nabla f(\hat{\mathbf{x}}_j)\rangle$. Moreover, since f is m-strongly convex, we know that

$$\langle \hat{\mathbf{x}}_j - \mathbf{x}^*, \nabla f(\hat{\mathbf{x}}_j)\rangle \geq m\|\hat{\mathbf{x}}_j - \mathbf{x}^*\|^2 \geq \frac{m}{2}\|\mathbf{x}_j - \mathbf{x}^*\|^2 - m\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|^2, \tag{2.4}$$

where the second inequality is a simple consequence of the triangle inequality. Now, let $a_j = \mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|^2$ and substitute (2.4) back into (2.3) to get

$$a_{j+1} \leq (1 - \gamma m)a_j + \gamma^2 \underbrace{\mathbb{E}\|g(\hat{\mathbf{x}}_j, s_j)\|^2}_{R_0^j} + 2\gamma m \underbrace{\mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|^2}_{R_1^j} + 2\gamma \underbrace{\mathbb{E}\langle \hat{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle}_{R_2^j}. \tag{2.5}$$

The recursive equation (2.5) is key to our analysis. We show that for given R_0^j, R_1^j, and R_2^j, we can obtain convergence rates through elementary algebraic manipulations. Observe that there are three "error" terms in (2.5): R_0^j captures the stochastic gradient decay with each iteration, R_1^j captures the mismatch between the true iterate and its noisy estimate, and R_2^j measures the size of the projection of that mismatch on the gradient at each step.

The key contributions of our work are to show that (1) this iteration can capture the algorithmic progress of asynchronous algorithms, and (2) the error terms can be bounded to obtain a $\mathcal{O}(\log(1/\epsilon)/\epsilon)$ rate for HOGWILD!, and linear rates of convergence for asynchronous SCD and asynchronous sparse SVRG.

Before turning to the specific analysis of these methods we show how rates of convergence can be obtained from (2.5), given upper bounds on the terms R_i^j. To this end we make several assumptions on the size of the stochastic gradients and on the nature of the perturbation $\mathbf{n}_j$. The assumptions might seem artificial, but we explain in subsequent sections why they hold for asynchronous implementations of SGM, SCD, and SVRG.

Assumption 1. There exists $M > 0$ such that $\|\mathbf{g}(\hat{\mathbf{x}}, s)\| \leq M$ for all $\hat{\mathbf{x}}$ and s encountered during a run of perturbed SGM.

Assumption 2. There exist τ and diagonal matrices $\mathbf{S}_i^j$ with entries in $\{-1, 0, 1\}$ such that

$$\mathbf{n}_j = \gamma \sum_{\substack{i=j-\tau \\ i \neq j}}^{j+\tau} \mathbf{S}_i^j \mathbf{g}(\hat{\mathbf{x}}_i, s_i). \tag{2.6}$$

Assumption 3. There exists a nonnegative symmetric function ν of pairs of samples (s_i, s_j) such that $\mathbb{E}_{s_i} \nu(s_i, s) \leq p_c < 1$ for any sample s, and

$$\mathbb{E}_{s_i,s_j} |\langle \mathbf{S}_i^j \mathbf{g}(\hat{\mathbf{x}}_i, s_i), \mathbf{S}_k^j \mathbf{g}(\hat{\mathbf{x}}_k, s_k)\rangle| \leq \mathbb{E}_{s_i,s_j} \nu(s_i, s_j) \|\mathbf{g}(\hat{\mathbf{x}}_i, s_i)\| \|\mathbf{g}(\hat{\mathbf{x}}_k, s_k)\|.$$

The first assumption can always be enforced in practice by restricting the optimization to a ℓ_∞ ball, which can be easily achieved by a coordinatewise thresholding operator. The second assumption requires the perturbation $\mathbf{n}_j$ to be a function of the gradients computed in a time interval around j. In subsequent sections we explain why this type of perturbation captures the behavior of asynchronous stochastic optimization methods. Finally, the third assumption requires the stochastic gradients computed at different time steps to be weakly correlated. This is true in the problems we consider because the stochastic gradients are sparse.

With Assumptions 1, 2, and 3 in place, we turn to bounding the error terms R_1^j and R_2^j (R_0^j is already upper bounded by M^2).

LEMMA 2.1. *Given Assumptions* 1, 2, *and* 3, *the error terms* R_1^j *and* R_2^j *are upper bounded as follows:*

$$R_1^j = \mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|^2 \leq \gamma^2 M^2 \left(2\tau + 4\tau^2 p_c\right) \text{ and } R_2^j = \mathbb{E}\langle \hat{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle \leq 2\gamma M^2 \tau p_c.$$

Proof. The expected squared norm of $\mathbf{n}_j = \hat{\mathbf{x}}_j - \mathbf{x}_j$ can be bounded as follows:

$$R_1^j = \gamma^2 \mathbb{E}\left\| \sum_{\substack{i=j-\tau \\ i \neq j}}^{j+\tau} \mathbf{S}_i^j \mathbf{g}(\hat{\mathbf{x}}_i, s_i) \right\|^2 \leq \gamma^2 \sum_i \mathbb{E}\|\mathbf{S}_i^j \mathbf{g}(\hat{\mathbf{x}}_i, s_i)\|^2 + \gamma^2 \sum_{\substack{i,k \\ i \neq k}} \mathbb{E}\left|\langle \mathbf{S}_i^j \mathbf{g}(\hat{\mathbf{x}}_i, s_i), \mathbf{S}_k^j \mathbf{g}(\hat{\mathbf{x}}_k, s_k)\rangle\right|.$$

Assumptions 1 and 3 imply that

$$R_1^j \le 2\tau \cdot \gamma^2 M^2 + \gamma^2 M^2 (2\tau)^2 p_c \le \gamma^2 M^2 \cdot \left(2\tau + 4\tau^2 p_c\right).$$

We can bound R_2^j in a similar way:

$$R_2^j = \gamma \sum_{\substack{i=j-\tau \\ i\neq j}}^{j+\tau} \mathbb{E}\langle \mathbf{S}_i^j \mathbf{g}(\hat{\mathbf{x}}_i, s_i), \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle \le 2\gamma M^2 \tau p_c. \qquad \square$$

Plugging the upper bounds provided by Lemma 2.1 into (2.5) readily yields the following convergence guarantee for perturbed SGM.

THEOREM 2.2. *When Assumptions* 1, 2, *and* 3 *hold and*

$$\tau = \mathcal{O}\left(\min\left\{\frac{1}{p_c}, \frac{M^2}{\epsilon m^2}\right\}\right),$$

perturbed SGM, with step size $\gamma = \mathcal{O}(1)\frac{\epsilon m}{M^2}$, *achieves error* $\mathbb{E}\|\mathbf{x}_T - \mathbf{x}^*\|^2 \le \epsilon$ *whenever*

$$T \ge \mathcal{O}(1)\frac{M^2}{\epsilon m^2}\log\left(\frac{a_0}{\epsilon}\right). \tag{2.7}$$

Proof. Recall that we denote $a_j = \mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|^2$. Recursion (2.5) together with Lemma 2.1 implies that

$$\begin{aligned} a_{j+1} &\le (1-\gamma m)a_j + \gamma^2 M^2 (1 + 2\gamma m(2\tau + 4\tau^2 p_c) + 4\tau p_c) \\ &\le (1-\gamma m)a_j + \mathcal{O}(1)\gamma^2 M^2 \\ &\le (1-\gamma m)^{j+1} a_0 + \mathcal{O}(1)\gamma \frac{M^2}{m}, \end{aligned}$$

where the second inequality follows by the assumption on the size of τ, and the final inequality follows by unrolling the recursion. Plugging in $\gamma = \mathcal{O}(1)\frac{\epsilon m}{M^2}$ yields the desired conclusion. $\square$

Theorem 2.2 shows that under Assumptions 1, 2, and 3 and $\tau = \mathcal{O}(p_c^{-1}, M^2/\epsilon m^2)$ perturbed SGM satisfies the same convergence rate (up to constants) as the standard SGM. In section 3 we show that HOGWILD! satisfies the necessary conditions for applying Theorem 2.2.

To obtain faster rates of convergence of perturbed SGM the size of the stochastic gradients should decay as the iterates approach the optimum. Such a property exists when the stochastic gradients are smooth in the following sense.

Assumption 4. There exist L_1 and L_2 such that for all $\mathbf{x}, \mathbf{y}$

$$\|\mathbf{g}(\mathbf{x}, s) - \mathbf{g}(\mathbf{y}, s)\| \le L_1 \|\mathbf{x} - \mathbf{y}\| \quad \text{and} \quad \mathbb{E}_s \|\mathbf{g}(\mathbf{x}, s)\|^2 \le L_2^2 \|\mathbf{x} - \mathbf{x}^*\|^2,$$

with $0 < L_2 \le L_1$ and $L_1 m \le L_2^2$ (the ordering between L_1 and L_2 is not essential; it allows us to make some computational simplifications later on).

The methods SCD and SVRG have this property, therefore achieving a linear convergence rate. Assumption 4 allows us to bound the term R_0^j as follows:

$$R_0^j = \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_j, s_j)\|^2 \le L_2^2 \mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}^*\|^2 \le 2L_2^2 a_j + 2L_2^2 \mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|^2 = 2L_2^2 a_j + 2L_2^2 R_1^j.$$

That is, we bound R_0^j in terms of R_1^j and a term involving a_j, which decreases as the algorithm progresses. As seen in Lemma 2.1, an upper bound on $\|\mathbf{g}(\hat{\mathbf{x}}, s)\|$ can be used to obtain upper bounds on $\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|$. In turn, such an upper bound translates into a bound on $\|\mathbf{g}(\hat{\mathbf{x}}, s)\|$. By repeating this procedure recursively, we can obtain the following upper bounds on R_0^j and R_1^j. The details are deferred to Appendix A.1.

LEMMA 2.3. *Fix $\ell \geq 1$, suppose $\tau \leq \frac{L_2}{m\ell}$, and set $\gamma = \frac{\theta m}{6L_2^2}$ for some $\theta \leq 1$. Then, when Assumptions* 1, 2, 3, *and* 4 *hold true,*

$$R_0^j \leq \mathcal{O}(1)\left(L_2^2 a_j + \theta^{2\ell} M^2\right) \quad \text{and} \quad R_1^j \leq \mathcal{O}(1)\left(\theta^2 a_j + \theta^{2\ell}\frac{M^2}{L_2^2}\right).$$

The Cauchy–Schwarz inequality implies the bound $R_2^j \leq \sqrt{R_0^j R_1^j}$. Unfortunately this approach yields a loose upper bound on R_2^j because upper bounding the inner product $\langle \mathbf{n}_j, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle$ by $\|\mathbf{n}_j\|\|\mathbf{g}(\hat{\mathbf{x}}_j, s_j)\|$ disregards the weak correlation of the stochastic gradients. The next lemma uses a slightly more involved argument to bound R_2^j. The proof can be found in Appendix A.1.

LEMMA 2.4. *Fix $\ell \geq 1$, suppose $\tau \leq \frac{L_2}{m\ell}$ and $\tau = \mathcal{O}(p_c^{-1/6})$, and set $\gamma = \frac{\theta m}{6L_2^2}$ for some $\theta \leq 1$. Then, when Assumptions* 1, 2, 3, *and* 4 *hold true,*

$$R_2^j \leq \mathcal{O}(1)\left(\theta m a_j + \theta^{2\ell}\frac{mM^2}{L_2^2}\right).$$

Plugging in the upper bounds provided by Lemmas 2.3 and 2.4 in (2.5) yields the following convergence guarantee.

THEOREM 2.5. *When Assumptions* 1, 2, 3, *and* 4 *hold, and*

$$\tau = \mathcal{O}\left(\min\left\{\frac{1}{\sqrt[6]{p_c}}, \frac{L_2}{m\log\left(\frac{M^2}{L_2^2\epsilon}\right)}\right\}\right),$$

perturbed SGM, with step size $\gamma = \mathcal{O}(1)\frac{m}{L_2^2}$, achieves error $\mathbb{E}\|\mathbf{x}_T - \mathbf{x}^*\|^2 \leq \epsilon$ whenever

$$T \geq \mathcal{O}(1)\frac{L_2^2}{m^2}\log\left(\frac{a_0}{\epsilon}\right). \tag{2.8}$$

Proof. We plug the upper bounds on R_0^j, R_1^j, and R_2^j provided by Lemmas 2.3 and 2.4 into our perturbed iterate recursive inequality, given by (2.5), to find that perturbed SGM satisfies

$$a_{j+1} \leq \Bigg(1 - \gamma m + \underbrace{\mathcal{O}(1)\left(\gamma^2 L_2^2 + \gamma m\theta^2 + \gamma\theta m\right)}_{=r(\gamma)}\Bigg) a_j + \mathcal{O}(1)\underbrace{\left(\gamma^2\theta^{2\ell}M^2 + \gamma\theta^{2\ell}\frac{mM^2}{L_2^2}\right)}_{=\delta(\gamma)}.$$

Now, our choice of step size guarantees that

$$a_{j+1} \leq \left(1 - \mathcal{O}(1)\frac{\theta m^2}{L_2^2}\right) a_j + \mathcal{O}(1)\theta^{2\ell}\frac{m^2M^2}{L_2^4} \leq \left(1 - \mathcal{O}(1)\frac{\theta m^2}{L_2^2}\right)^{j+1} a_0 + \mathcal{O}(1)\theta^{2\ell}\frac{M^2}{L_2^2}.$$

We choose $\ell = \mathcal{O}(1)\log\left(\frac{M^2}{L_2^2\epsilon}\right)$ so that $\mathcal{O}(1)\theta^{2\ell}\frac{M^2}{L_2^2} = \epsilon/2$.

The desired result now follows from solving $\left(1 - \frac{\mathcal{O}(1)m^2}{L_2^2}\right)^{j+1} a_0 \leq \epsilon/2$ for j. ∎

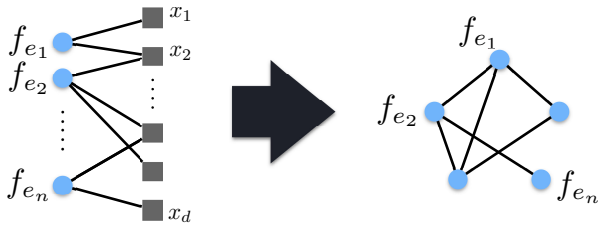


FIG. 1. *The bipartite graph on the left has as its leftmost vertices the n function terms and as its rightmost vertices the coordinates of $\mathbf{x}$. A term f_{e_i} is connected to a coordinate x_j if hyperedge e_i contains j (i.e., if the ith term is a function of that coordinate). The graph on the right depicts a conflict graph between the function terms. The vertices denote the function terms, and two terms are joined by an edge if they conflict on at least one coordinate in the bipartite graph.*

In sections 4 and 5 we argue that ASCD and SVRG satisfy the necessary conditions for applying Theorem 2.5, therefore obtaining guarantees of linear convergence of these methods.

3. Analyzing Hogwild! In this section, we provide a simple analysis of HOGWILD!, the asynchronous implementation of SGM. We focus on functions f that are decomposable into n terms:

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_{e_i}(\mathbf{x}), \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^d$, and each $f_{e_i}(\mathbf{x})$ depends only on the coordinates indexed by the subset e_i of $\{1, 2, \ldots, d\}$. For simplicity we assume that the terms of f are differentiable; our results can be readily extended to nondifferentiable f_{e_i}.

We refer to the sets e_i as *hyperedges* and denote the set of hyperedges by $\mathcal{E}$. We sometimes refer to the f_{e_i} as the *terms* of f. As shown in Figure 1, the hyperedges induce a bipartite graph between the n terms and the d variables in $\mathbf{x}$, yielding a *conflict graph* between the n terms. Let $\overline{\Delta}_{\mathrm{C}}$ be the average degree in the conflict graph, that is, the average number of terms that are in conflict with a single term. We assume that $\overline{\Delta}_{\mathrm{C}} \geq 1$; otherwise we could decompose the problem into smaller independent subproblems. As we will see, under our perturbed iterate analysis framework the convergence rate of asynchronous algorithms depends on $\overline{\Delta}_{\mathrm{C}}$.

HOGWILD! (Algorithm 1) is a method of parallelizing SGM in the asynchronous setting [26]. It is deployed on multiple cores that have access to a shared memory, where the optimization variable $\mathbf{x}$ and the data points defining the f terms are stored. During its execution each core samples uniformly at random a hyperedge s from $\mathcal{E}$. It reads the coordinates $v \in s$ of the shared vector $\mathbf{x}$, evaluates ∇f_s at the point read, and finally adds $-\gamma \nabla f_s$ to the shared variable.

During the execution of HOGWILD!, cores do not synchronize or follow an order between reads and writes. Moreover, they access (i.e., read or write) a set of coordinates in $\mathbf{x}$ without the use of any locking mechanisms that would ensure a conflict-free execution. Therefore the reads/writes of distinct cores can intertwine in arbitrary ways, e.g., while a core updates a subset of variables, before completing its task; other cores can read/write the same subset of variables.

In [26], the authors analyzed a variant of HOGWILD! in which several simplifying assumptions were made. Specifically, in [26], (1) only a single coordinate per sampled

Algorithm 1 HOGWILD!

1: **while** number of sampled hyperedges $\leq T$ **do in parallel**
2: sample a random hyperedge s
3: $[\hat{\mathbf{x}}]_s$ = an inconsistent read of the shared variable $[\mathbf{x}]_s$
4: $[\mathbf{u}]_s = -\gamma \cdot \mathbf{g}([\hat{\mathbf{x}}]_s, s)$
5: **for** $v \in s$ **do**
6: $[\mathbf{x}]_v = [\mathbf{x}]_v + [\mathbf{u}]_v$ // atomic write
7: **end for**
8: **end while**

hyperedge is updated (i.e., the for loop in HOGWILD! is replaced with a single coordinate update); (2) the authors assume *consistent reads* (i.e., they assume that while a core is reading shared variables, no writes from other cores occur); (3) the authors make an implicit assumption on the uniformity of the processing times of samples (explained in the following) that does not generically hold in practice. These simplifications alleviate some of the challenges in analyzing HOGWILD! and allowed the authors to provide a convergence result. As we show in the current paper, however, these simplifications are not necessary to obtain a convergence analysis. Our perturbed iterates framework can be used in an elementary way to analyze the original version of HOGWILD!, yielding improved bounds compared to earlier analyses.

3.1. Ordering the samples. A subtle but important point in the analysis of HOGWILD! is the need to define an order for the sampled hyperedges. A key point of difference of our work is that *we order the samples based on the order in which they were sampled*, not the order in which cores complete the processing of the samples.

DEFINITION 3.1. *We denote by s_i the ith sampled hyperedge in a run of Algorithm* 1.

Therefore, s_i denotes the sample obtained when line 2 in Algorithm 1 is executed for the ith time. This is different from the original work of [26], in which the samples were ordered according to their completion time. The issue with such an ordering is that the distribution of the samples, conditioned on the ordering, is not always uniform; for example, hyperedges of small cardinality are more likely to be "early" samples. A uniform distribution is needed for the theoretical analysis of stochastic gradient methods, a point that is disregarded in [26]. Our ordering according to sampling time resolves this issue by guaranteeing uniformity of the samples in a trivial way.

3.2. Defining read iterates and clarifying independence assumptions. Since the shared memory variable can change inconsistently during reads and writes, we also have to be careful about the notion of *iterates* in HOGWILD!.

DEFINITION 3.2. *We denote by $\overline{\mathbf{x}}_i$ the contents of the shared memory before the ith execution of line* 2. *Moreover, we denote by $\hat{\mathbf{x}}_i \in \mathbb{R}^d$ the vector that in coordinates $v \in s_i$ contains exactly what the core that sampled s_i read. We then define $[\hat{\mathbf{x}}_i]_v = [\overline{\mathbf{x}}_i]_v$ for all $v \notin s_i$. Note that we do not assume* consistent reads*; i.e., the contents of the shared memory can potentially change while a core is reading.*

At this point we would like to briefly discuss an *independence* assumption made by all prior work. In the following section, we explain why this assumption is not always true in practice. In Appendix B, we show how to remove the independence assumption, but for ease of exposition we continue to adopt it in our main text.

Assumption 5. The vector $\hat{\mathbf{x}}_i$ is independent of the sampled hyperedge s_i.

This independence assumption is important when establishing the convergence rate of the algorithm, and has been made explicitly or implicitly in prior work [12, 22, 23, 26]. Specifically, when proving convergence rates for these algorithms we use the fact that $\mathbb{E}_{s_i} \langle \hat{\mathbf{x}}_i - \mathbf{x}^*, g(\hat{\mathbf{x}}_i, s_i) \rangle = \langle \hat{\mathbf{x}}_i - \mathbf{x}^*, \nabla f(\hat{\mathbf{x}}_i) \rangle$, which follows from the independence of $\hat{\mathbf{x}}_i$ and s_i. However, observe that although $\overline{\mathbf{x}}_i$ is independent of s_i by construction, this is not the case for the vector $\hat{\mathbf{x}}_i$ read by the core that sampled s_i. Consider the scenario of two consecutively sampled hyperedges in Algorithm 1 that overlap on a subset of coordinates. Then, say one core is reading the coordinates of the shared variables indexed by its hyperedge, while the second core is updating a subset of these coordinates. In this case, the values read by the first core depend on the support of the sampled hyperedge.

One way to rigorously enforce the independence of $\hat{\mathbf{x}}_i$ and s_i is to require the processors to read the *entire* shared variable $\mathbf{x}$ before sampling a new hyperedge. However, this might not be reasonable in practice, as the dimension of $\mathbf{x}$ tends to be considerably larger than the sparsity of the hyperedges. As mentioned earlier, in Appendix B we show how to overcome the issue of dependence and thereby remove Assumption 5. To ease readability, however, in the main text we adopt Assumption 5.

3.3. The perturbed iterates view of asynchrony. In this work, we assume that all writes are atomic, in the sense that they will be successfully recorded in the shared memory at some point. Atomicity is a reasonable assumption in practice, as it can be strictly enforced through compare-and-swap operations [26].

Assumption 6. Every write in line 6 of Algorithm 1 will complete successfully.

This assumption says that all writes are recorded in the shared memory by the end of the execution, in the form of coordinatewise updates. Due to commutativity and Assumption 6, after a total of T hyperedges are processed the shared memory contains

$$\underbrace{\overbrace{\mathbf{x}_0 - \gamma \mathbf{g}(\hat{\mathbf{x}}_0, s_0)}^{\mathbf{x}_1} - \cdots - \gamma \mathbf{g}(\hat{\mathbf{x}}_{T-1}, s_{T-1})}_{\mathbf{x}_T}, \tag{3.2}$$

where $\mathbf{x}_0$ is the initial guess and $\mathbf{x}_i$ is defined as the vector that contains *all* gradient updates up to sample s_{i-1}.

Remark 1. Throughout this section we denote $\mathbf{g}(\mathbf{x}, s_j) = \nabla f_{s_j}(\mathbf{x})$, which we assume to be bounded: $\|\mathbf{g}(\mathbf{x}, s)\| \leq M$ (as in Assumption 1). Such a uniform bound on the norm of the stochastic gradient is true when operating on a bounded ℓ_∞ ball; this can in turn be enforced by a simple, coordinatewise thresholding operator.

Remark 2. Observe that although a core is only reading the subset of variables that are indexed by its sampled hyperedge, in (3.2) we use the entire vector $\hat{\mathbf{x}}$ as the input to the sampled gradient. We can do this since $\mathbf{g}(\hat{\mathbf{x}}_k, s_k)$ is independent of the coordinates of $\hat{\mathbf{x}}_k$ outside the support of hyperedge s_k.

Using the above definitions, we define the perturbed iterates of HOGWILD! as

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma \mathbf{g}(\hat{\mathbf{x}}_i, s_i) \tag{3.3}$$

for $i = 0, 1, \ldots, T-1$, where s_i is the ith uniformly sampled hyperedge. Observe that all but the first and last of these iterates are "fake": there might not be an actual

time during the execution when they exist in the shared memory. However, $\mathbf{x}_0$ is what is stored in memory before the execution starts, and $\mathbf{x}_T$ is exactly what is stored in shared memory at the end of the execution.

We observe that the iterates in (3.3) place HOGWILD! in the perturbed gradient framework introduced in section 2:

$$a_{j+1} \leq (1-\gamma m)a_j + \gamma^2 \underbrace{\mathbb{E}\|g(\hat{\mathbf{x}}_j, s_j)\|^2}_{R_0^j} + 2\gamma m \underbrace{\mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|^2}_{R_1^j} + 2\gamma \underbrace{\mathbb{E}\langle \hat{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle}_{R_2^j}.$$

We are only left to bound the three error terms R_0^j, R_1^j, and R_2^j. Before we proceed, we note that for the technical soundness of our theorems, we have to also define a random variable that captures the *system randomness*. In particular, let ξ denote a random variable that encodes the randomness of the system (i.e., random delays between reads and writes, gradient computation time, etc.). Although we do not explicitly use ξ, its distribution is required implicitly for computing the expectations in the convergence analysis because the random samples $s_0, s_1, \ldots, s_{T-1}$ do not fully determine the output of Algorithm 1. However, $s_0, \ldots, s_{T-1}$ along with ξ completely determines the time of all reads and writes. Finally, we state one more assumption that is needed in our analysis and is also made in prior work.

Assumption 7 (bounded overlaps). Two hyperedges s_i and s_j overlap in time if they are processed concurrently at some point during the execution of HOGWILD!. The time during which a hyperedge s_i is being processed begins when the sampling function is called and ends after the last coordinate of $\mathbf{g}(\hat{\mathbf{x}}_i, s_i)$ is written to the shared memory. We assume that there exists a number $\tau \geq 0$, such that the maximum number of sampled hyperedges that can overlap in time with a particular sampled hyperedge cannot be more than τ.

The usefulness of the above assumption is that it essentially abstracts away all system details relative to delays, processing overlaps, and number of cores into a single parameter. Intuitively, τ can be perceived as a proxy for the number of cores; i.e., we would expect that no more than roughly $O(\#\text{cores})$ sampled hyperedges overlap in time with a single hyperedge. Observe that if τ is small, then we expect the distance between $\mathbf{x}_j$ and the noisy iterate $\hat{\mathbf{x}}_j$ to be small. In our perturbed iterate framework, if we set $\tau = 0$, we obtain the classical iterative formula of serial SGM.

To quantify the distance between $\hat{\mathbf{x}}_j$ (i.e., the iterate read by the core that sampled s_j) and $\mathbf{x}_j$ (i.e., the "fake" iterate used to establish convergence rates), we observe that any difference between them is caused solely by hyperedges that overlap in time with s_j. To see this, let s_i be an "earlier" sample, i.e., $i < j$, that does not overlap with s_j in time. This implies that the processing of s_i finishes before s_j starts being processed. Hence, the *full* contribution of $\gamma\mathbf{g}(\hat{\mathbf{x}}_i, s_i)$ will be recorded in both $\hat{\mathbf{x}}_j$ and $\mathbf{x}_j$ (for the latter this is true by definition). Similarly, if $i > j$ and s_i does not overlap with s_j in time, then neither $\hat{\mathbf{x}}_j$ nor $\mathbf{x}_j$ (for the latter, again by definition) contains *any* of the coordinate updates involved in the gradient update $\gamma\mathbf{g}(\hat{\mathbf{x}}_i, s_i)$. Assumption 7 ensures that if $i < j - \tau$ or $i > j + \tau$, the sample s_i *does not* overlap in time with s_j.

By the above discussion, and due to Assumption 7, there exist diagonal matrices $\mathbf{S}_i^j$ with diagonal entries in $\{-1, 0, 1\}$ such that

$$\hat{\mathbf{x}}_j - \mathbf{x}_j = \sum_{i=j-\tau,\, i\neq j}^{j+\tau} \gamma \mathbf{S}_i^j \mathbf{g}(\hat{\mathbf{x}}_i, s_i). \tag{3.4}$$

We have therefore shown that the assumptions we made about HOGWILD! imply that Assumption 2 in section 2 also holds true. The diagonal matrices account for any possible pattern of (potentially) partial updates that can occur while hyperedge s_j is being processed. We would like to note that the above notation bears resemblance to the coordinate-update mismatch formulation of asynchronous coordinate-based algorithms, as in [21, 23, 28].

We also remark that HOGWILD! satisfies Assumption 3 with $\nu(s_i, s_j) = \mathbb{1}(s_i \cap s_j)$ and $p_c = \frac{2\overline{\Delta}_C}{n}$. Therefore, Theorem 2.2 translates into the following convergence guarantee for HOGWILD!.

THEOREM 3.3. *If the number of samples that overlap in time with a single sample during the execution of* HOGWILD! *is bounded as*

$$\tau = \mathcal{O}\left(\min\left\{\frac{n}{\overline{\Delta}_C}, \frac{M^2}{\epsilon m^2}\right\}\right),$$

HOGWILD!, *with step size* $\gamma = \mathcal{O}(1)\frac{\epsilon m}{M^2}$, *achieves error* $\mathbb{E}\|\mathbf{x}_T - \mathbf{x}^*\|^2 \leq \epsilon$ *whenever*

$$T \geq \mathcal{O}(1)\frac{M^2 \log\left(\frac{a_0}{\epsilon}\right)}{\epsilon m^2}.$$

Since the iteration bound in the theorem is (up to a constant) the same as that of serial SGM, our result implies a linear speedup. We note that an improved rate of $O(1/\epsilon)$ can be obtained by appropriately diminishing step sizes per epoch (see, e.g., [15, 26]). Furthermore, observe that although the $\frac{M^2}{\epsilon m^2}$ bound on τ might seem restrictive, it is—up to a logarithmic factor—proportional to the total number of iterations required by HOGWILD! (or even serial SGM) to reach ϵ accuracy. Assuming that the average degree of the conflict graph is constant, and that we perform a constant number of passes over the data, i.e., $T = c \cdot n$, then τ can be as large as $\tilde{\mathcal{O}}(n)$, i.e., nearly linear in the number of function terms.[1]

3.4. Comparison with the original Hogwild! analysis of [26]. Let us summarize the key points of improvement compared to the original HOGWILD! analysis:

1. Our analysis is elementary and compact, and follows simply by bounding the R_0^j, R_1^j, and R_2^j terms, after introducing the perturbed gradient framework in section 2.

2. We do not assume consistent reads: while a core is reading from the shared memory, other cores are allowed to read or write.

3. In [26] the authors analyze a simplified version of HOGWILD! where for each sampled hyperedge only a randomly selected coordinate is updated. Here we analyze the "full-update" version of HOGWILD!.

4. We order the samples by the order in which they are sampled, not by completion time. This allows us to rigorously prove our convergence bounds, without assuming anything about the distribution of the processing time of each hyperedge. This is unlike [26], where there is an implicit assumption of uniformity with respect to processing times.

5. The previous work of [26] establishes a nearly linear speedup for HOGWILD! if τ is bounded as $\tau = \mathcal{O}\big(\sqrt[4]{n/\Delta_{\mathrm{R}}\Delta_{\mathrm{L}}^2}\big)$, where Δ_{R} is the maximum right degree of the bipartite graph shown in Figure 1, and Δ_{L} is the maximum left degree of the same

[1] $\tilde{\mathcal{O}}$ hides logarithmic terms.

graph. Observe that $\Delta_{\mathrm{R}} \cdot \Delta_{\mathrm{L}}^2 \geq \Delta_{\mathrm{L}} \cdot \Delta_{\mathrm{C}}$, where Δ_{C} is the maximum degree of the conflict graph. Here, we obtain a linear speedup for up to $\tau = \mathcal{O}\left(\min\left\{n/\overline{\Delta}_{\mathrm{C}}, M^2/\epsilon m^2\right\}\right)$, where $\overline{\Delta}_{\mathrm{C}}$ is only the average degree of the conflict graph in Figure 1. Our bound on the delays can be orders of magnitude better than that of [26].

4. Asynchronous stochastic coordinate descent. In this section, we use the perturbed gradient framework to analyze the convergence of asynchronous parallel stochastic coordinate descent (ASCD). This algorithm has been previously analyzed in [22, 23]. We show that the algorithm admits an elementary treatment in our perturbed iterate framework, under the same assumptions made for HOGWILD!.

Algorithm 2 ASCD

while iterations $\leq T$ **do in parallel**
  $\hat{\mathbf{x}}$ = an inconsistent read of the shared variable $\mathbf{x}$
  Sample a coordinate s
  $u_s = -\gamma \cdot d[\nabla f(\mathbf{x})]_s$
  $[\mathbf{x}]_s = [\mathbf{x}]_s + u_s$ // atomic write
end while

ASCD, shown in Algorithm 2, is a linearly convergent algorithm for minimizing strongly convex functions f. At each iteration a core samples one of the coordinates, computes a full gradient update for that coordinate, and proceeds to update a single element of the shared memory variable $\mathbf{x}$. The challenge in analyzing ASCD, compared to HOGWILD!, is that, in order to show linear convergence, we need to show that the error due to the asynchrony between cores decays fast enough when the iterates arrive close to the optimal solution. In addition to the assumptions made in the analysis of HOGWILD!, to prove the linear convergence of ASCD we just need to show that ASCD satisfies Assumption 4 in section 2.

We define $\hat{\mathbf{x}}_i$ as in the previous section, but now the samples s_i are coordinates sampled uniformly at random from $\{1, 2, \ldots, d\}$. After T samples have been processed completely, the following vector is contained in shared memory:

$$\underbrace{\overbrace{\mathbf{x}_0 - \gamma d[\nabla f(\hat{\mathbf{x}}_0)]_{s_0}\mathbf{e}_{s_0}}^{\mathbf{x}_1} - \cdots - \gamma d[\nabla f(\hat{\mathbf{x}}_{T-1})]_{s_{T-1}}\mathbf{e}_{s_{T-1}}}_{\mathbf{x}_T},$$

where $\mathbf{x}_0$ is the initial guess, $\mathbf{e}_{s_j}$ is the standard basis vector with a one at position s_j, and $[\nabla f(\mathbf{x})]_{s_j}$ and denotes the s_jth coordinate of the gradient of f evaluated at $\mathbf{x}$. Similar to HOGWILD! in section 3, ASCD satisfies the following iterative formula:

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \gamma \cdot d \cdot [\nabla f(\hat{\mathbf{x}}_j)]_{s_j}\mathbf{e}_{s_j} = \mathbf{x}_j - \gamma \cdot \mathbf{g}(\hat{\mathbf{x}}_j, s_j).$$

Note that $\mathbb{E}_{s_j}\mathbf{g}(\hat{\mathbf{x}}_j, s_j) = \nabla f(\hat{\mathbf{x}}_j)$, and thus ASCD's iterates $a_j = \mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|^2$ satisfy the recursion of (2.5):

$$a_{j+1} \leq (1 - \gamma m)\, a_j + \gamma^2 \underbrace{\mathbb{E}\|g(\hat{\mathbf{x}}_j, s_j)\|^2}_{R_0^j} + 2\gamma m \underbrace{\mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|^2}_{R_1^j} + 2\gamma \underbrace{\mathbb{E}\langle \hat{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle}_{R_2^j}.$$

We observe that Assumption 4 holds true, with $L_1 = dL$ and $L_2 = \sqrt{d}L$, where L is the Lipschitz constant. We also note that Assumption 3 holds true with $\nu(s_i, s_j) = \mathbb{1}(s_i = s_j)$, which satisfies $\mathbb{E}_{s_i}\nu(s_i, s_j) = d^{-1}$. We define the largest distance between

the optimal vector and the vector read by the cores during the execution of the algorithm as $\hat{a}_0 := \max_{0\le k\le T} \|\hat{\mathbf{x}}_k - \mathbf{x}^*\|^2$, a value which should be thought of as proportional to $a_0 = \|\mathbf{x}_0 - \mathbf{x}^*\|^2$. This intuition can be enforced by restricting the optimization to an ℓ_∞ ball, as discussed in sections 2 and 3. Then, Assumption 1 holds true with $M = dL\sqrt{\hat{a}_0}$. Finally, we denote the condition number of f by $\kappa := L/m$. Then, Theorem 2.5 translates into the following convergence guarantee for ASCD.

THEOREM 4.1. *Let the maximum number of coordinates that can be concurrently processed while a core is processing a single coordinate be at most*

$$\tau = \mathcal{O}\left(\min\left\{\frac{\kappa\sqrt{d}}{\log\left(\frac{d\hat{a}_0}{\epsilon}\right)}, \sqrt[6]{d}\right\}\right).$$

Then, ASCD with step size $\gamma = \frac{\mathcal{O}(1)}{dL\kappa}$ *achieves error* $\mathbb{E}\|\mathbf{x}_T - \mathbf{x}^*\|^2 \le \epsilon$ *whenever*

$$T \ge \mathcal{O}(1)\cdot d\kappa^2 \log\left(\frac{a_0}{\epsilon}\right).$$

Using the recursive inequality (2.5), serial SCD, with the same step size as in the above theorem, can be shown to achieve the same accuracy as ASCD in the same number of steps. Hence, as long as the proxy for the number of cores is bounded as $\tau = \mathcal{O}(\min\{\kappa\sqrt{d}\log(da_0/\epsilon)^{-1}, \sqrt[6]{d}\})$, our theorem implies a linear speedup with respect to this simple convergence bound. We would like to note, however, that the coordinate descent literature sometimes uses more refined properties of the function to be optimized that can lead to potentially better convergence bounds, especially in terms of function value accuracy, i.e., $f(\mathbf{x}_k) - f(\mathbf{x}^*)$ (see, e.g., [8, 22, 23]).

We also remark that between the two bounds on τ, the second one, i.e., $\mathcal{O}(\sqrt[6]{d})$, is the more restrictive, as the first one is proportional—up to log factors—to the square root of the number of iterations, which is usually $\Omega(d)$. We explain in Appendix A.1 how this loose bound might be improved but leave the tightness of the bound as an open question for future work.

5. Sparse and asynchronous SVRG. The SVRG algorithm, presented in [19], is a variance-reduction approach to stochastic gradient descent with strong theoretical guarantees and empirical performance. In this section, we present a parallel, asynchronous, and sparse variant of SVRG. We also present a convergence analysis that proceeds in a nearly identical way to that of ASCD.

5.1. Serial sparse SVRG. The original SVRG algorithm of [19] runs for a number of epochs; the per epoch iteration is given as follows:

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \gamma\left(\mathbf{g}(\mathbf{x}_j, s_j) - \mathbf{g}(\mathbf{y}, s_j) + \nabla f(\mathbf{y})\right), \tag{5.1}$$

where $\mathbf{y}$ is the last iterate of the previous epoch, and as such is updated at the end of every epoch. Here f is of the same form as in (3.1):

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_{e_i}(\mathbf{x}),$$

and $\mathbf{g}(\mathbf{x}, s_j) = \nabla f_{s_j}(\mathbf{x})$, with hyperedges $s_j \in \mathcal{E}$ sampled uniformly at random. As is common in the SVRG literature, we further assume that the individual f_{e_i} terms are L-smooth. The theoretical innovation in SVRG is having an SGM flavored algorithm, with small amortized cost per iteration, where the variance of the gradient estimate

is smaller than that of standard SGM. For a certain selection of learning rate, epoch size, and number of iterations, Johnson and Zhang [19] establish that SVRG attains a linear rate.

Observe that when optimizing a decomposable f with sparse terms, in contrast to SGM, the SVRG updates will be dense due to the term $\nabla f(\mathbf{y})$. From a practical perspective, when the SGM updates are sparse—the case in several applications [26]—the cost of writing a sparse update in shared memory is significantly smaller than applying the dense gradient update term $\nabla f(\mathbf{y})$. Furthermore, dense updates cause significantly more memory conflicts in an asynchronous execution, amplifying the error terms in (2.5), and introducing time delays due to memory contention.

A sparse version of SVRG can be obtained by letting the support of the update be determined by that of $\mathbf{g}(\mathbf{y}, s_j)$:

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \gamma \left(\mathbf{g}(\mathbf{x}_j, s_j) - \mathbf{g}(\mathbf{y}, s_j) + \mathbf{D}_{s_j} \nabla f(\mathbf{y})\right) = \mathbf{x}_j - \gamma \mathbf{v}_j, \tag{5.2}$$

where $\mathbf{D}_{s_j} = \mathbf{P}_{s_j} \mathbf{D}$, and $\mathbf{P}_{s_j}$ is the projection on the support of s_j and $\mathbf{D} = \operatorname{diag}\left(p_1^{-1}, \ldots, p_d^{-1}\right)$ is a $d \times d$ diagonal matrix. The weight p_v is equal to the probability that index v belongs to a hyperedge sampled uniformly at random from $\mathcal{E}$. These probabilities can be computed from the right degrees of the bipartite graph shown in Figure 1. The normalization ensures that $\mathbb{E}_{s_j} \mathbf{D}_{s_j} \nabla f(\mathbf{y}) = \nabla f(\mathbf{y})$ and thus that $\mathbb{E}\mathbf{v}_j = \nabla f(\mathbf{x}_j)$. We will establish the same upper bound on $\mathbb{E}\|\mathbf{v}_j\|^2$ for sparse SVRG as that used in [19] to establish a linear rate of convergence for dense SVRG.

LEMMA 5.1. *The second moment of the updates in* (5.2) *of sparse SVRG satisfies*

$$\mathbb{E}\|\mathbf{v}_j\|^2 \leq 2\mathbb{E}\|\mathbf{g}(\mathbf{x}_j, s_j) - \mathbf{g}(\mathbf{x}^*, s_j)\|^2 + 2\mathbb{E}\|\mathbf{g}(\mathbf{y}, s_j) - \mathbf{g}(\mathbf{x}^*, s_j)\|^2 - 2\nabla f(\mathbf{y})^\top \mathbf{D} \nabla f(\mathbf{y}).$$

Proof. By definition, $\mathbf{v}_j = \mathbf{g}(\mathbf{x}_j, s_j) - \mathbf{g}(\mathbf{y}, s_j) + \mathbf{D}_{s_j} \nabla f(\mathbf{y})$. Therefore

$$\begin{aligned}\mathbb{E}\|\mathbf{v}_j\|^2 &= \mathbb{E}\|\mathbf{g}(\mathbf{x}_j, s_j) - \mathbf{g}(\mathbf{y}, s_j) + \mathbf{D}_{s_j} \nabla f(\mathbf{y})\|^2 \\ &\leq 2\mathbb{E}\|\mathbf{g}(\mathbf{x}_j, s_j) - \mathbf{g}(\mathbf{x}^*, s_j)\|^2 + 2\mathbb{E}\|\mathbf{g}(\mathbf{y}, s_j) - \mathbf{g}(\mathbf{x}^*, s_j) - \mathbf{D}_{s_j} \nabla f(\mathbf{y})\|^2.\end{aligned}$$

We expand the second term to find that

$$\begin{aligned}&\mathbb{E}\|\mathbf{g}(\mathbf{y}, s_j) - \mathbf{g}(\mathbf{x}^*, s_j) - \mathbf{D}_{s_j} \nabla f(\mathbf{y})\|^2 \\ &= \mathbb{E}\|\mathbf{g}(\mathbf{y}, s_j) - \mathbf{g}(\mathbf{x}^*, s_j)\|^2 - 2\mathbb{E}\langle \mathbf{g}(\mathbf{y}, s_j) - \mathbf{g}(\mathbf{x}^*, s_j), \mathbf{D}_{s_j} \nabla f(\mathbf{y})\rangle + \mathbb{E}\|\mathbf{D}_{s_j} \nabla f(\mathbf{y})\|^2.\end{aligned}$$

Since $\mathbf{g}(\mathbf{x}, s_j)$ is supported on s_j for all $\mathbf{x}$, we have

$$\mathbb{E}\langle \mathbf{g}(\mathbf{y}, s_j) - \mathbf{g}(\mathbf{x}^*, s_j), \mathbf{D}_{s_j} \nabla f(\mathbf{y})\rangle = \mathbb{E}\langle \mathbf{g}(\mathbf{y}, s_j) - \mathbf{g}(\mathbf{x}^*, s_j), \mathbf{D} \nabla f(\mathbf{y})\rangle = \nabla f(\mathbf{y})^\top \mathbf{D} \nabla f(\mathbf{y}),$$

where the second equality follows by the property of iterated expectations. The conclusion follows because $\mathbb{E}\|\mathbf{D}_{s_j} \nabla f(\mathbf{y})\|^2 = \nabla f(\mathbf{y})^\top \mathbf{D} \nabla f(\mathbf{y})$. ∎

Observe that the last term in the variance bound is a nonnegative quadratic form, and hence we can drop it and obtain the same second moment bound as that obtained in [19] for dense SVRG. Hence, Lemma 5.1 has the following corollary.

COROLLARY 5.2. *Sparse SVRG attains the same convergence rate as SVRG* [19].

We note that usually the convergence rates for SVRG are obtained for function value differences. However, since our perturbed iterate framework of section 2 is based on iterate differences, we rederive a convergence bound for iterates. The proof of the next result is deferred to Appendix A.2.

THEOREM 5.3. *Sparse SVRG, with step size* $\gamma = \mathcal{O}(1)\frac{1}{L\kappa}$ *and epoch size* $S = \mathcal{O}(1)\kappa^2$, *reaches accuracy* $\mathbb{E}\|\mathbf{y}_E - \mathbf{x}^*\|^2 \leq \epsilon$ *after* $E = \mathcal{O}(1)\log(a_0/\epsilon)$ *epochs, where* $\mathbf{y}_E$ *is the last iterate of the final epoch, and* $a_0 = \|\mathbf{x}_0 - \mathbf{x}^*\|^2$ *is the initial distance squared to the optimum.*

5.2. KroMagnon: Asynchronous parallel sparse SVRG. We now present an asynchronous implementation of sparse SVRG. The implementation, which we refer to as KROMAGNON, is given in Algorithm 3.

Algorithm 3 KROMAGNON

x = **y** = **x**$_0$
for Epoch = 1 : E **do**
  Compute in parallel **z** = $\nabla f(\mathbf{y})$
  while number of sampled hyperedges $\leq S$ **do in parallel**
    sample a random hyperedge s
    $[\hat{\mathbf{x}}]_s$ = an inconsistent read of the shared variable $[\mathbf{x}]_s$
    $[\mathbf{u}]_s = -\gamma \cdot (\nabla f_s([\mathbf{x}]_s) - \nabla f_s([\mathbf{y}]_s) - \mathbf{D}_s\mathbf{z})$
    for $v \in s$ **do**
      $[\mathbf{x}]_v = [\mathbf{x}]_v + [\mathbf{u}]_v$ // atomic write
    end for
  end while
  y = **x**
end for

Let $\mathbf{v}(\hat{\mathbf{x}}_j, s_j) = \mathbf{g}(\hat{\mathbf{x}}_j, s_j) - \mathbf{g}(\mathbf{y}, s_j) + \mathbf{D}_{s_j}\nabla f(\mathbf{y})$ be the perturbed gradient update vector. Then, after processing a total of T hyperedges, the shared memory contains

$$\underbrace{\overbrace{\mathbf{x}_0 - \gamma\mathbf{v}(\hat{\mathbf{x}}_0, s_0)}^{\mathbf{x}_1} - \cdots - \gamma\mathbf{v}(\hat{\mathbf{x}}_{T-1}, s_{T-1})}_{\mathbf{x}_T}. \tag{5.3}$$

We define the "fake" iterates as $\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma\mathbf{v}(\hat{\mathbf{x}}_i, s_i)$ for $i = 0, 1, \ldots, T-1$, where s_i is the ith uniformly sampled hyperedge. Since $\mathbb{E}\mathbf{v}(\hat{\mathbf{x}}_j, s_j) = \nabla f(\mathbf{x}_j)$, KROMAGNON also satisfies recursion (2.5):

$$a_{j+1} \leq (1-\gamma m)a_j + \gamma^2 \underbrace{\mathbb{E}\|\mathbf{v}(\hat{\mathbf{x}}_j, s_j)\|^2}_{R_0^j} + 2\gamma m \underbrace{\mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|^2}_{R_1^j} + 2\gamma \underbrace{\mathbb{E}\langle \hat{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{v}(\hat{\mathbf{x}}_j, s_j)\rangle}_{R_2^j}.$$

To prove the convergence of KROMAGNON we note that just as HOGWILD!, it satisfies Assumptions 2 and 3. We can suppose Assumption 1 holds. From Lemma 5.1 we know that KROMAGNON almost satisfies Assumption 4 with $L_1 = L_2 = L$. The only difference is that we now have to track a term involving the iterates **y** which are updated every epoch. The arguments used in the proof of Theorem 2.5 can be extended to account for these iterates. The details are deferred to Appendix A.2.

THEOREM 5.4. *Let the maximum number of samples that can overlap in time with a single sample be bounded as*

$$\tau = \mathcal{O}\left(\min\left\{\frac{\kappa}{\log\left(\frac{M^2}{L^2\epsilon}\right)}, \sqrt[6]{\frac{n}{\overline{\Delta}_C}}\right\}\right).$$

Then, KROMAGNON, *with step size* $\gamma = \mathcal{O}(1)\frac{1}{L\kappa}$ *and epoch size* $S = \mathcal{O}(1)\kappa^2$, *attains* $\mathbb{E}\|\mathbf{y}_E - \mathbf{x}^*\|^2 \leq \epsilon$ *after* $E = \mathcal{O}(1)\log(a_0/\epsilon)$ *epochs, where* $\mathbf{y}_E$ *is the last iterate of the final epoch, and* $a_0 = \|\mathbf{x}_0 - \mathbf{x}^*\|^2$ *is the initial distance squared to the optimum.*

TABLE 1

The problems and data sets used in our experimental evaluation. We test KROMAGNON*, dense SVRG, and* HOGWILD! *on three different tasks: linear regression, logistic regression, and vertex cover. We test the algorithms on sparse data sets, of various sizes and feature dimensions.*

Problem	Dataset	# data	# features	Sparsity
Linear regression	Synthetic	3M	10K	20
Logistic regression	Synthetic	3M	10K	20
	rcv1 [20] from [1]	$\approx$ 700K	$\approx$ 42K	$\approx$ 73
	url [25] from [1]	$\approx$ 3.2M	$\approx$ 2.4M	$\approx$ 116
Vertex cover[2]	eswiki-2013 [5, 6, 7]	$\approx$ 970K	$\approx$ 23M	$\approx$ 24
	wordassociation-2011 [5, 6, 7]	$\approx$ 10.6K	$\approx$ 72K	$\approx$ 7

We would like to note that the total number of iterations in this bound is—up to a universal constant—the same as that of serial sparse SVRG as presented in Theorem 5.3. Again, as with HOGWILD! and ASCD, this implies a linear speedup with respect the simple bound of Theorem 5.3.

Similar to our ASCD analysis, we remark that between the two bounds on τ, the second one is the more restrictive. The first one is, up to logarithmic factors, equal to the square root of the total number of iterations per epoch; we expect that the size of the epoch is proportional to n, the number of function terms (or data points). This suggests that the first bound is proportional to $\tilde{\mathcal{O}}(\sqrt{n})$ for most reasonable applications. Moreover, the second bound is certainly loose; we argue that it can be tightened using a more refined analysis.

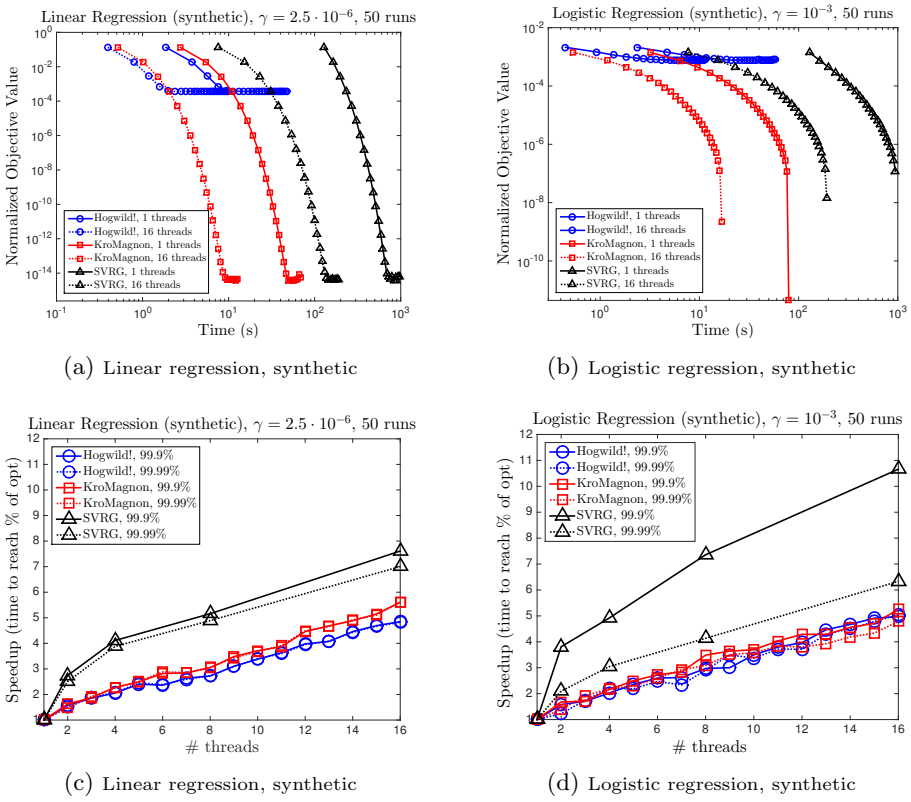
6. Empirical evaluation of KroMagnon. In this section we evaluate KROMAGNON empirically. Our two goals are to demonstrate that (1) KROMAGNON is faster than dense SVRG, and (2) KROMAGNON has speedups comparable to those of HOGWILD!. We implemented HOGWILD!, asynchronous dense SVRG, and KROMAGNON in Scala, and tested them on the problems and datasets listed in Table 1. Each algorithm was run for 50 epochs, using up to 16 threads. For the SVRG algorithms, we recompute $\mathbf{y}$ and the full gradient $\nabla f(\mathbf{y})$ every two epochs. We normalize the objective values such that the objective at the initial starting point has a value of one, and the minimum attained across all algorithms and epochs has a value of zero. Experiments were conducted on a Linux machine with 2 Intel Xeon E5-2670 Processors (2.60GHz, eight cores each) with 250GB memory.

Comparison with dense SVRG. We were unable to run dense SVRG on the url and eswiki-2013 datasets due to the large number of features. Figures 2a, 2b, and 4a show that KROMAGNON is one to two orders of magnitude faster than dense SVRG. In fact, running dense SVRG on 16 threads is slower than KROMAGNON on a single thread. Moreover, as seen in Figure 3a, KROMAGNON on 16 threads can be up to four orders of magnitude faster than serial dense SVRG. Both dense SVRG and KROMAGNON attain similar optima.

Speedups. We measured the time each algorithm takes to achieve 99.9% and 99.99% of the minimum achieved by that algorithm. Speedups are computed relative to the runtime of the algorithm on a single thread. Although the speedup of KROMAGNON varies across datasets, we find that KROMAGNON has comparable

[2]Following [22], we optimize a quadratic penalty relaxation for vertex cover

$$\min_{x\in[0,1]^{|V|+|E|}} \sum_{v\in V} x_v + \frac{\beta}{2} \sum_{(u,v)\in E} (x_u + x_v - x_{u,v} - 1)^2 + \frac{1}{2\beta} \sum_{v\in V} x_v^2 + \sum_{e\in E} x_e^2.$$



(a) Linear regression, synthetic

(b) Logistic regression, synthetic

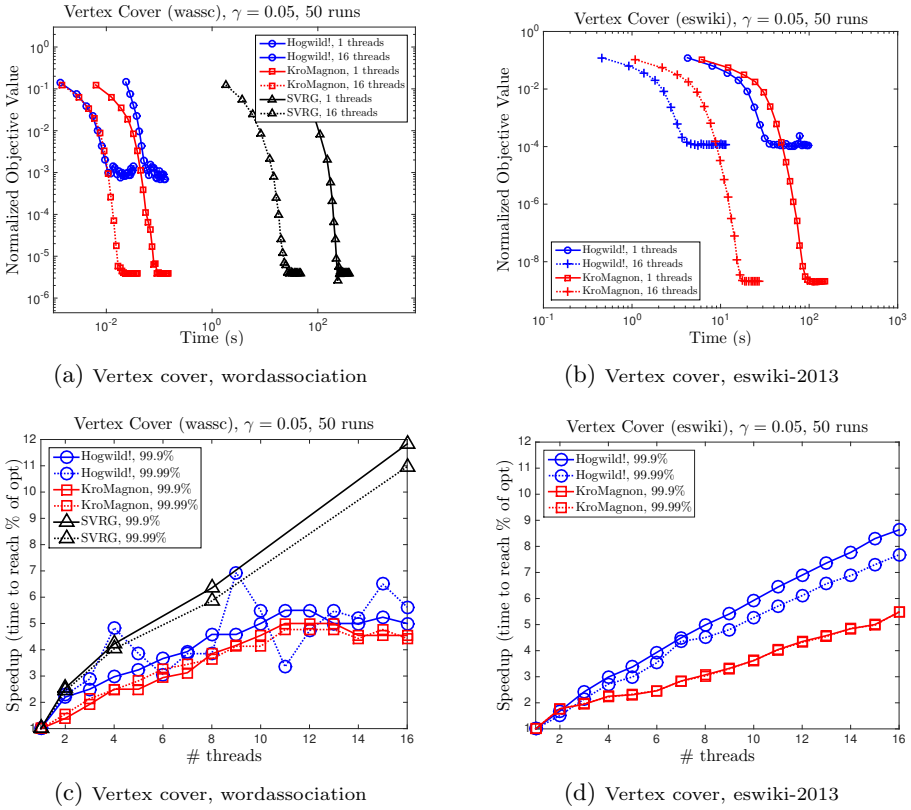
(c) Linear regression, synthetic

(d) Logistic regression, synthetic

FIG. 2. *Linear and logistic regression on synthetic data. In subfigures* (a) *and* (b) *we plot the convergence with respect to the normalized objective value as a function of wall-clock time, and in* (c) *and* (d) *the speedup with respect to the number of threads. These experiments are all for linear and logistic regression problems on synthetic data, in which we have* 3 *million data points, each with* $10K$ *features, and each data point has* 20 *nonzero entries. We observe that* KROMAGNON *is significantly faster than parallel and dense SVRG, while they both can attain better objective values compared to constant step size* HOGWILD!. *Moreover, we observe that the speedup gains of* HOGWILD! *and* KROMAGNON *scale reasonably well for up to* 16 *threads.*

speedups with HOGWILD! on all datasets, as shown in Figures 2c, 2d, 3c, 3d, 4c, and 4d. We further observe that dense SVRG has better speedup scaling. This happens because the per-iteration complexity of HOGWILD! and KROMAGNON is significantly cheaper to the extent that the additional overhead associated with having extra threads leads to some speedup loss; this is not the case for dense SVRG, as the per-iteration cost is higher.

7. Discussion. We have introduced a novel framework for analyzing parallel asynchronous stochastic gradient optimization algorithms. The main advantage of our framework is that it is straightforward to apply to a range of first-order stochastic algorithms, requiring only elementary derivations. Moreover, in our analysis we lift, or relax, many of the assumptions made in prior art; e.g., we do not assume consistent reads, and we analyze full stochastic gradient updates. We use our framework to analyze HOGWILD! and ASCD, and further introduce and analyze KROMAGNON, a new asynchronous sparse SVRG algorithm.



(a) Vertex cover, wordassociation

(b) Vertex cover, eswiki-2013

(c) Vertex cover, wordassociation

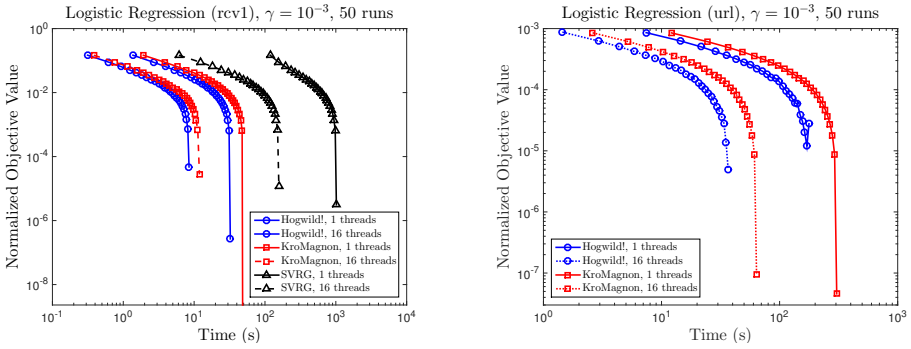
(d) Vertex cover, eswiki-2013

FIG. 3. *Vertex cover on the wordassociation-2011 and eswiki-2013 datasets. Subfigure* (a) *shows the convergence of the algorithms on wordassociation-2011, a small graph with less than* 11,000 *vertices.* KROMAGNON *on a single thread is* 3–4 *orders of magnitude faster than dense SVRG on this dataset. Convergence of* KROMAGNON *and* HOGWILD! *on the eswiki-2013 dataset is shown in subfigure* (b)*; we were unable to run dense SVRG on this larger graph. Subfigures* (c) *and* (d) *show the speedups of the algorithms on the two datasets. In subfigure* (c)*, both* HOGWILD! *and* KROMAGNON *exhibit poorer speedups than dense SVRG because of the rapid convergence on the smaller wordassociation-2011 dataset. In subfigure* (d)*, we observe that* HOGWILD! *achieves a speedup of up to* $8x$ *and* KROMAGNON *up to* $5x$.

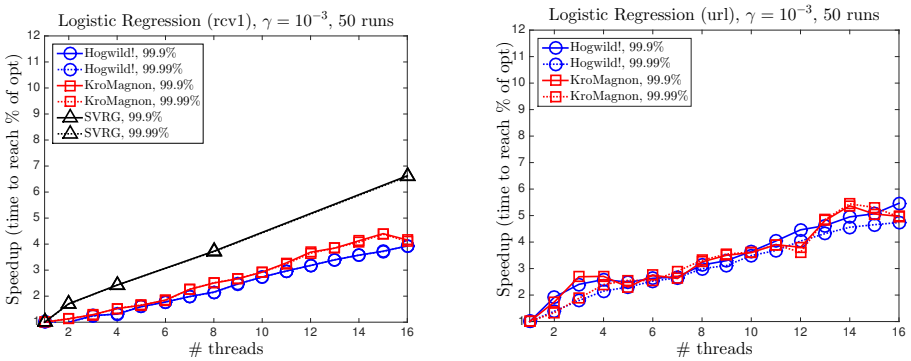
We conclude with some open problems:

1. It would be interesting to obtain tighter bounds for the convergence of function values of the algorithms presented. How do the "errors" due to asynchrony influence the convergence rate of function values? In this case the number of iterations required to reach a target accuracy should scale with the condition number of the objective, not its square. Moreover, the literature on stochastic coordinate descent establishes convergence results in terms of coordinatewise Lipschitz constants—a more refined smoothness quantity than the full-function smoothness. It would be worthwhile to know whether our framework can be adapted to take these parameters into account.

2. Our perturbed iterates framework relies fundamentally on the strong convexity assumption. However, asynchronous algorithms are known to perform well on nonstrongly convex (and even nonconvex) objectives. Can we generalize our framework to simply convex or smooth functions? Under what assumptions, or simple



(a) Logistic regression, rcv1 (b) Logistic regression, url



(c) Logistic regression, rcv1 (d) Logistic regression, url

FIG. 4. *Logistic regression on the rcv*1 *and url datasets. Subfigure* (a) *shows the convergence of the algorithms on the rcv*1 *dataset. For a given objective value,* KROMAGNON *is* 1–2 *orders of magnitude faster than dense SVRG. On the larger url dataset (subfigure* (b)*), we were unable to run dense SVRG. Note that some of the effect of asynchrony can be observed in these experiments: the objective values obtained by* KROMAGNON, HOGWILD!*, and dense SVRG are slightly different on* 1 *thread compared to* 16 *threads. Speedups of the algorithms are shown in subfigures* (c) *and* (d). KROMAGNON *has a slightly better speedup than* HOGWILD! *on rcv*1 *and the same speedup on url.*

families of functions, can we show convergence for nonconvex problems?

3. As previously explained, we believe that the upper bounds on τ—the proxy for the number of cores—in our ASCD and KROMAGNON analyses are amenable to improvements. It is an open problem to explore the extent of such improvements.

4. Our analysis offers sensible upper bounds only in the presence of sparsity. It seems, however, that to obtain speedup results for HOGWILD!, it is only necessary to have small correlation between randomly sampled gradients. In what practical setups do randomly selected gradients have sufficiently small correlation? Does that immediately imply linear speedups in the same way that update sparsity does?

5. In this work we analyzed three similar stochastic first-order methods. It is an open problem to apply our framework and provide an elementary analysis for a greater variety of stochastic gradient–type optimization algorithms, such as AdaGrad-type schemes (similar to [12]) or stochastic dual coordinate methods (similar to [17]).

6. Capturing the effects of asynchrony as noise on the algorithmic input seems to be applicable to settings beyond stochastic optimization. As shown recently for

a combinatorial graph problem, a similar viewpoint enables the analysis of an asynchronous graph clustering algorithm [27]. It is an interesting endeavor to explore the extent to which a perturbed iterate viewpoint is suitable for analyzing general asynchronous iterative algorithms.

Appendix A. Omitted proofs.

A.1. Perturbed SGM.

A.1.1. Bounding R_0^j and R_1^j when Assumption 4 holds. We start with a simple upper bound on the norm of the gradient updates. From Assumption 4 about the L_2 smoothness of the stochastic gradients we infer that

$$\mathbb{E}_{s_k} \|\mathbf{g}(\hat{\mathbf{x}}_k, s_k)\|^2 = L_2^2 \|\hat{\mathbf{x}}_k - \mathbf{x}^*\|^2 \leq 2L_2^2 \|\mathbf{x}_j - \mathbf{x}^*\|^2 + 2L_2^2 \|\mathbf{x}_j - \hat{\mathbf{x}}_k\|^2. \tag{A.1}$$

Let T be the total number of perturbed SGM iterations, and let us define the set

$$\mathcal{S}_r^j = \{\max\{j - r\tau, 0\}, \ldots, j-1, j, j+1, \ldots, \min\{j + r\tau, T\}\},$$

which has cardinality at most $2r\tau + 1$ and contains all indices around j within $r\tau$ steps, as sketched in Figure 5.

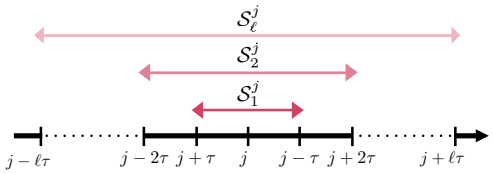


FIG. 5. *The set* $\mathcal{S}_r^j = \{\max\{j - r\tau, 0\}, \ldots, j-1, j, j+1, \ldots, \min\{j + r\tau, T\}\}$ *comprises the indices around* j *(including* j*) within* $r\tau$ *steps. The cardinality of such a set is* $2r\tau + 1$*. Here,* $\mathcal{S}_0^j = \{j\}$.

Due to Assumption 2 there exist diagonal matrices $\mathbf{S}_i^j$ with entries in $\{-1, 0, 1\}$ such that, for any index k in the set $\mathcal{S}_r^j$, we have

$$\hat{\mathbf{x}}_k - \mathbf{x}_j = \gamma \sum_{i \in \mathcal{S}_{r+1}^j} \mathbf{S}_{i,k}^j \mathbf{g}(\hat{\mathbf{x}}_i, s_i). \tag{A.2}$$

This equation implies that the difference between the iterate $\mathbf{x}_j$ at time j and the iterate $\hat{\mathbf{x}}_k$ at time k can be expressed as a linear combination of any updates that occurred during the time interval defined by $\mathcal{S}_{r+1}^j$.

From (A.2) we see that $\|\hat{\mathbf{x}}_k - \mathbf{x}_j\|$, for any $k \in \mathcal{S}_r^j$, can be upper bounded in terms of the magnitude of the updates that occur in $\mathcal{S}_{r+1}^j$. On the other hand, (A.1) shows that the magnitude of the gradient steps can be upper bounded in terms of the size of the mismatches. These observations, which are fundamental to our approach, are made explicit in the following lemma.

LEMMA A.1. *For any* $j \in \{0, \ldots, T\}$*, we have*

$$\max_{k \in \mathcal{S}_r^j} \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_k, s_k)\|^2 \leq 2L_2^2 \left(a_j + \max_{k \in \mathcal{S}_r^j} \mathbb{E}\|\hat{\mathbf{x}}_k - \mathbf{x}_j\|^2 \right), \tag{A.3}$$

$$\max_{k \in \mathcal{S}_r^j} \mathbb{E}\|\hat{\mathbf{x}}_k - \mathbf{x}_j\|^2 \leq (3\gamma\tau(r+1))^2 \max_{k \in \mathcal{S}_{r+1}^j} \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_k, s_k)\|. \tag{A.4}$$

Proof. The first inequality is a consequence of (A.1). For the second, as mentioned previously, we have $\hat{\mathbf{x}}_k - \mathbf{x}_j = \sum_{i \in \mathcal{S}^j_{r+1}} \mathbf{S}^j_{i,k} \gamma \mathbf{g}(\hat{\mathbf{x}}_i, s_i)$ when $k \in \mathcal{S}^j_r$. Hence,

$$\mathbb{E}\|\hat{\mathbf{x}}_k - \mathbf{x}_j\|^2 = \gamma^2 \cdot \mathbb{E}\left\{ \left\| \sum_{i \in \mathcal{S}^j_{r+1}} \mathbf{S}^j_{i,k} \mathbf{g}(\hat{\mathbf{x}}_i, s_i) \right\|^2 \right\} \le \gamma^2 \cdot \mathbb{E}\left\{ |\mathcal{S}^j_{r+1}| \sum_{i \in \mathcal{S}^j_{r+1}} \|\mathbf{g}(\hat{\mathbf{x}}_i, s_i)\|^2 \right\}$$
$$\le \gamma^2 \cdot |\mathcal{S}^j_{r+1}|^2 \max_{i \in \mathcal{S}^j_{r+1}} \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_i, s_i)\|^2 \le (3\gamma\tau(r+1))^2 \max_{i \in \mathcal{S}^j_{r+1}} \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_i, s_i)\|^2,$$

where the first inequality follows due to Jensen's inequality, and the last inequality uses the bound $|\mathcal{S}^j_{r+1}| \le 2(r+1)\tau + 1 \le 3\tau(r+1)$. ∎

Remark 3. The τ^2 factor in the upper bound on $\max \mathbb{E}\|\hat{\mathbf{x}}_k - \mathbf{x}_j\|^2$ in Lemma A.1 might be loose. We believe that it should instead be τ, when τ is smaller than some measure of the correlation parameter p_c. If the sparsity of the steps $\mathbf{g}(\hat{\mathbf{x}}_i, s_i)$ can be exploited, we suspect that the condition $\tau = \mathcal{O}(p_c^{-1/6})$ in Theorem 2.5 could be improved to $\tau = \mathcal{O}(p_c^{-1/4})$.

For simplicity, let $G_r = \max_{k \in \mathcal{S}^j_r} \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_k, s_k)\|^2$ and $\Delta_r = \max_{k \in \mathcal{S}^j_r} \mathbb{E}\|\hat{\mathbf{x}}_k - \mathbf{x}_j\|^2$. Observe that $R^j_0 = \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_j, s_j)\|^2 = G_0$ and $R^j_1 = \mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}_j\|^2 = \Delta_0$. To obtain bounds for our first two error terms, R^j_0 and R^j_1, we will expand the recursive relations that are implied by Lemma A.1.

LEMMA A.2. *Fix $\ell \ge 1$, suppose $\tau \le \frac{L_2}{m\ell}$, and set $\gamma = \frac{\theta m}{6L_2^2}$ for some $\theta \le 1$. Then, when Assumptions* 1, 2, 3, *and* 4 *hold true,*

$$R^j_0 \le \mathcal{O}(1)\left(L_2^2 a_j + \theta^{2\ell} M^2\right) \quad \text{and} \quad R^j_1 \le \mathcal{O}(1)\left(\theta^2 a_j + \theta^{2\ell} \frac{M^2}{L_2^2}\right).$$

Proof. Let $A = 2L_2^2 a_j$, $B = 2L_2^2$, and $C = (\gamma\tau)^2$. Then, we can rewrite the bounds of Lemma A.1 as $G_r \le A + B \cdot \Delta_r$ and $\Delta_r \le 3^2(r+1)^2 \cdot C \cdot G_{r+1}$, which implies $G_r \le A + 3^2(r+1)^2 BC \cdot G_{r+1}$. We can now upper bound $R^j_0 = G_0$ by applying the previous inequality ℓ times. If we expand the formulas, we get

$$R^j_0 = G_0 \le A \sum_{i=0}^{\ell-1} (3^i \cdot i!)^2 (BC)^i + (3^\ell \cdot \ell!)^2 (BC)^\ell G_\ell. \tag{A.5}$$

Since $\gamma = \frac{\theta m}{6L_2^2}$ and $\tau \le \frac{L_2}{m\ell}$ (the choice of τ is made so that the sum in (A.5) is small), we have $BC = 2L_2^2\gamma^2\tau^2 \le 2L_2^2 \frac{\theta^2 m^2}{6^2 L_2^4} \frac{L_2^2}{m^2\ell^2} \le \frac{1}{2 \cdot 3^2 \cdot \ell^2}$. Using the upper bound $k! \le k^k$ on each term of the sum (A.5), and plugging in the upper bound on BC, we get

$$\sum_{i=0}^{\ell-1} (3^i \cdot i!)^2 (BC)^i \le \sum_{i=0}^{\ell-1} \frac{(i!)^2}{2^i \ell^{2i}} \le \sum_{i=0}^{\ell-1} \frac{1}{2^i} \left(\frac{i}{\ell}\right)^{2i} \le 2.$$

Similarly, we can upper bound the last term of (A.5) by $(3^\ell \cdot \ell!)^2 (BC)^\ell \le 2^{-\ell}\theta^{2\ell}$. Finally, $G_\ell \le M^2$. Combining these bounds gives us $R^j_0 \le \mathcal{O}(1)\left(L_2^2 a_j + \theta^{2\ell} M^2\right)$.

We can now bound R^j_1. By definition, $R^j_1 = \Delta_0$, and from Lemma A.1 we have that $\Delta_0 \le 3^2 \cdot C \cdot G_1$. We can bound G_1 similarly to G_0 as

$$G_1 \le A \sum_{i=0}^{\ell-1} (3^i \cdot (i+1)!)^2 (BC)^i + (3^\ell \cdot (\ell+1)!)^2 (BC)^\ell G_{\ell+1}.$$

As before, $BC \le \frac{\theta^2}{2\cdot 3^2\ell^2}$. Since $(i+1)! \le 2\ell^i$ for any $0 \le i \le \ell$, it follows that $\sum_{i=0}^{\ell-1}(3^i\cdot(i+1)!)^2(BC)^i \le \mathcal{O}(1)$, and $(3^\ell\cdot(\ell+1)!)^2(BC)^\ell \le \mathcal{O}(1)\theta^{2\ell}$. Therefore, because $G_{\ell+1} \le M^2$, we obtain $G_1 \le \mathcal{O}(1)(L_2^2 a_j + \theta^{2\ell}M^2)$. Since $C = (\gamma\tau)^2 \le \frac{\theta^2}{L_2^2}$, it follows that $R_1 \le \mathcal{O}(1)\left(\theta^2 a_j + \theta^{2\ell}(M/L_2)^2\right)$. ∎

A.1.2. Bounding R_2^j.

LEMMA A.3. *Fix $\ell \ge 1$, suppose $\tau \le \frac{L_2}{m\ell}$ and $\tau = \mathcal{O}(p_c^{-1/6})$, and set $\gamma = \frac{\theta m}{6L_2^2}$ for some $\theta \le 1$. Then, when Assumptions 1, 2, 3, and 4 hold true,*

$$R_2^j \le \mathcal{O}(1)\left(\theta m a_j + \theta^{2\ell}\frac{mM^2}{L_2^2}\right).$$

Proof. Because of Assumptions 2 and 3 we can upper bound R_2^j as follows:

$$R_2^j = \mathbb{E}\langle \hat{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle \le \gamma \cdot \sum_{\substack{i=j-\tau \\ i\ne j}}^{j+\tau} \mathbb{E}\|\mathbf{g}(\hat{\mathbf{x}}_i, s_i)\| \cdot \|\mathbf{g}(\hat{\mathbf{x}}_j, s_j)\| \cdot \nu(s_i, s_j). \tag{A.6}$$

The random variable $\nu(s_i, s_j)$ encodes weak correlation of the gradient steps. Recall that by Assumption 3 we have $\mathbb{E}\nu(s_i, s_j) = p_c < 1$ whenever $i \ne j$. To take advantage of this fact we use smoothness to replace the iterates $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$, by $\hat{\mathbf{x}}_{j-3\tau}$. The latter iterate is independent of both s_i and s_j by Assumption 2. This independence allows us to "untangle" the expectation of $\nu(s_i, s_j)$ from the sum in (A.6).

For clarity, we note that when $j < 3\tau$, we have $\hat{\mathbf{x}}_{j-3\tau} = \mathbf{x}_0$. From Assumption 4 on the smoothness of the stochastic gradients we get the following bounds:

$$\begin{aligned}\|\mathbf{g}(\hat{\mathbf{x}}_j, s_j)\| &\le \|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\| + L_1\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\|,\\ \|\mathbf{g}(\hat{\mathbf{x}}_i, s_i)\| &\le \|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_i)\| + L_1\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\|.\end{aligned}$$

Then, the expectation of a term $\|\mathbf{g}(\hat{\mathbf{x}}_i, s_i)\| \cdot \|\mathbf{g}(\hat{\mathbf{x}}_j, s_j)\| \cdot \nu(s_i, s_j)$ in the sum (A.6) is upper bounded by

$$\begin{aligned}\mathbb{E}\Big\{\Big(&\|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_i)\|\,\|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\| + L_1^2\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\|\,\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\|\\ &+ L_1\|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\|\,\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\| + L_1\|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_i)\|\,\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\|\Big)\cdot\nu(s_i, s_j)\Big\}.\end{aligned}$$

We first bound the second term using Cauchy–Schwarz and the property of iterated expectation, to exploit the expectation of the $\nu(s_i, s_j)$ term

$$\begin{aligned}&\mathbb{E}\{\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\| \cdot \|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\| \cdot \nu(s_i, s_j)\}\\ &\qquad\le \sqrt{\mathbb{E}\{\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\|^2\} \cdot \mathbb{E}\{\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\|^2 \cdot \nu(s_i, s_j)\}}\\ &\qquad= \sqrt{\mathbb{E}\{\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\|^2\} \cdot \mathbb{E}_{\sim s_j}\{\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\|^2 \cdot \mathbb{E}_{s_j}\{\nu(s_i, s_j)\}\}}\\ &\qquad= \sqrt{p_c}\sqrt{\mathbb{E}\{\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\|^2\} \cdot \mathbb{E}\{\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\|^2\}}\\ &\qquad\le \mathcal{O}(1)\sqrt{p_c}\cdot\gamma^2\tau^2 \underbrace{\max_{k\in\mathcal{S}_4^j}\mathbb{E}\{\|\mathbf{g}(\hat{\mathbf{x}}_k, s_k)\|^2\}}_{G_4},\end{aligned}$$

where the first equality follows due to $\hat{\mathbf{x}}_j$ being independent of s_j; hence the expectation with respect to s_j can be applied to $\nu(s_i, s_j)$. The last inequality follows from our arguments in the proof of Lemma A.1 because both mismatches $\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j$ can be written as linear combinations of gradient steps indexed by $\mathcal{S}_4^j$ as in (A.2). Similarly the third term satisfies the inequality

$$\begin{aligned}\mathbb{E}\{\|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\| \cdot \|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\| \cdot \nu(s_i, s_j)\} \\ \leq \sqrt{\mathbb{E}\{\|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\|^2\} \cdot \mathbb{E}\{\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\|^2 \cdot \nu(s_i, s_j)\}} \\ = \mathcal{O}(1)\sqrt{p_c} \cdot \gamma\tau G_4.\end{aligned}$$

The same bound applies for the fourth term $\mathbb{E}\{\|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_i)\| \cdot \|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\| \cdot \nu(s_i, s_j)\}$, while the first term can be easily bounded as

$$\mathbb{E}\{\|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\| \cdot \|\mathbf{g}(\hat{\mathbf{x}}_{j-3\tau}, s_i)\| \cdot \nu(s_i, s_j)\} \leq \sqrt{p_c} G_4.$$

Putting all pieces together, and using the prescribed value of $\gamma = \frac{\theta m}{6L_2^2}$, we have that

$$R_2 \leq \mathcal{O}(1)\sqrt{p_c}(\gamma\tau)\left(1 + L_1\gamma\tau + (L_1\gamma\tau)^2\right) G_4 \leq \mathcal{O}(1)\sqrt{p_c} \cdot \gamma \cdot \tau^3 \cdot G_4.$$

The first inequality follows because we are summing over 2τ terms in (A.6). To see why the second inequality is true, note that $L_1\gamma \leq \frac{\theta}{6} \leq 1$ (because $L_1 m$ is assumed to be at most L_2^2). Therefore $1 + L_1\gamma\tau + (L_1\gamma\tau)^2 \leq 1 + \tau + \tau^2 \leq 3\tau^2$. As in the proof of Lemma 2.3, we can bound G_4 by

$$\begin{aligned}G_4 &\leq \mathcal{O}(1)A\sum_{i=0}^{\ell-1}(3^i \cdot (i+4)!)^2(BC)^i + \mathcal{O}(1)(3^\ell \cdot (\ell+4)!)^2(BC)^\ell G_{\ell+4} \\ &\leq \mathcal{O}(1)(L_2^2 a_j + \theta^{2\ell} M^2).\end{aligned}$$

The result follows, assuming $\tau = \mathcal{O}(p_c^{-1/6})$ and $\gamma = \frac{\theta m}{6L_2^2}$. ∎

Remark 4. We believe that if we use the same bounding technique that we applied for R_2^j on R_0^j and R_1^j, then we can significantly improve the restrictive bound on τ.

A.2. KroMagnon.

A.2.1. Convergence of serial sparse SVRG.

LEMMA A.4. *Let the step size be $\gamma = \frac{1}{4L\kappa}$ and the length of an epoch be $8\kappa^2$. Then, $\mathbb{E}\|\mathbf{y}_k - \mathbf{x}^*\|^2 \leq 0.75^k \cdot \mathbb{E}\|\mathbf{y}_0 - \mathbf{x}^*\|^2$, where $\mathbf{y}_k$ is the iterate at the end of the kth epoch.*

Proof. We bound the distance to the optimum after one epoch of length $8\kappa^2$:

$$\begin{aligned}&\mathbb{E}\|\mathbf{x}_{j+1} - \mathbf{x}^*\|^2 = \mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|^2 - 2\gamma\mathbb{E}\langle\mathbf{x}_j - \mathbf{x}^*, \mathbf{v}_j\rangle + \gamma^2\mathbb{E}\|\mathbf{v}_j\|^2 \\ &\quad\leq \mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|^2 - 2\gamma\mathbb{E}\langle\mathbf{x}_j - \mathbf{x}^*, \nabla f(\mathbf{x}_j)\rangle + 2\gamma^2\mathbb{E}\|\mathbf{g}(\mathbf{x}_j, s_j) - \mathbf{g}(\mathbf{x}^*, s_j)\|^2 \\ &\qquad + 2\gamma^2\mathbb{E}\|\mathbf{g}(\mathbf{y}, s_j) - \mathbf{g}(\mathbf{x}^*, s_j)\|^2 \\ &\quad\leq \mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|^2 - 2\gamma\mathbb{E}\langle\mathbf{x}_j - \mathbf{x}^*, \nabla f(\mathbf{x}_j)\rangle + 2\gamma^2L^2\mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|^2 + 2\gamma^2L^2\mathbb{E}\|\mathbf{y} - \mathbf{x}^*\|^2 \\ &\quad\leq (1 - 2\gamma m + 2\gamma^2L^2)\mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|^2 + 2\gamma^2L^2\mathbb{E}\|\mathbf{y} - \mathbf{x}^*\|^2.\end{aligned}$$

The first inequality follows from Lemma 5.1 and an application of iterated expectations to obtain $\mathbb{E}\langle\mathbf{x}_j - \mathbf{x}^*, \mathbf{v}_j\rangle = \mathbb{E}\langle\mathbf{x}_j - \mathbf{x}^*, \nabla f(\mathbf{x}_j)\rangle$. The second inequality follows from the smoothness of $\mathbf{g}(\mathbf{x}, s_j)$, and the third follows since f is m-strongly convex.

We can rewrite the inequality as $a_{j+1} \le (1-2\gamma m+2\gamma^2L^2)a_j + 2\gamma^2L^2a_0$, because by construction $\mathbf{y} = \mathbf{x}_0$. Let $\gamma = \frac{1}{4L\kappa}$. Then, $1-2\gamma m+2\gamma^2L^2 \le 1-\frac{1}{4\kappa^2}$ and

$$\sum_{i=0}^{j}(1-2\gamma m+2\gamma^2L^2)^i \le \sum_{i=0}^{j}\left(1-\frac{1}{4\kappa^2}\right)^i \le \sum_{i=0}^{\infty}\left(1-\frac{1}{4\kappa^2}\right)^i = 4\kappa^2,$$

since $\frac{1}{4\kappa^2} \le \frac{1}{4}$. Therefore

$$\begin{aligned} a_{j+1} &\le \left(1-\frac{1}{4\kappa^2}\right)a_j + 2\gamma^2L^2a_0 \le \left(1-\frac{1}{4\kappa^2}\right)^{j+1}a_0 + \sum_{i=0}^{j-1}\left(1-\frac{1}{4\kappa^2}\right)^i \cdot 2\gamma^2L^2a_0 \\ &\le \left(1-\frac{1}{4\kappa^2}\right)^{j+1}a_0 + 4\kappa^2\gamma^2L^2a_0 = \left[\left(1-\frac{1}{4\kappa^2}\right)^{j+1} + \frac{1}{4}\right]a_0. \end{aligned}$$

Setting the length of an epoch to be $j = 2\cdot(4\kappa^2)$ gives us $a_{j+1} \le (1/2+1/4)\cdot a_0 = 0.75\cdot a_0$, and the conclusion follows. ∎

A.2.2. Bounding R_0^j and R_1^j. It is easy to see that due to Lemma 5.1 we get the following bound on the norm of the gradient estimate.

LEMMA A.5. *For any k and j we have*

$$\mathbb{E}\,\|\mathbf{v}(\hat{\mathbf{x}}_k, s_k)\|^2 \le 4L^2\left(a_j + a_0 + \mathbb{E}\|\mathbf{x}_j - \hat{\mathbf{x}}_k\|^2\right). \tag{A.7}$$

Proof. Due to Lemma 5.1 we have $\mathbb{E}\|\mathbf{v}(\hat{\mathbf{x}}_j, s_j)\|^2 \le 2L^2\mathbb{E}\|\hat{\mathbf{x}}_j - \mathbf{x}^*\|^2 + 2L^2\mathbb{E}\|\mathbf{y} - \mathbf{x}^*\|^2$. Then, since $\mathbf{y} = \mathbf{x}_0$, we obtain the result by applying the triangle inequality once. ∎

LEMMA A.6. *Suppose $\tau \le \frac{\kappa}{\ell}$ and $\gamma = \frac{\theta}{12L\kappa}$. Then the error terms R_0^j and R_1^j of* KROMAGNON *satisfy the following inequalities:*

$$R_0^j \le \mathcal{O}(1)\left(L^2(a_j+a_0) + \theta^{2\ell}M^2\right) \quad \text{and} \quad R_1^j \le \mathcal{O}(1)\left(\theta^2(a_j+a_0) + \theta^{2\ell}\frac{M^2}{L^2}\right).$$

Proof. Let $A = 4L^2(a_j+a_0)$, $B = 4L^2$, and $C = (\gamma\tau)^2$. Then, these inequalities can be rewritten as

$$G_r \le A + B\Delta_r \quad \text{and} \quad \Delta_r \le 3^2(r+1)^2CG_{r+1}. \tag{A.8}$$

Then, arguments analogous to those given in the proof of Lemma 2.3 show that if $\gamma = \frac{\theta}{12L\kappa}$ and $\tau \le \kappa/\ell$, where $\kappa = \frac{L}{m}$ is the condition number and $\theta \le 1$, the desired inequalities hold true. ∎

A.2.3. Bounding R_2^j.

LEMMA A.7. *Suppose $\tau \le \frac{\kappa}{\ell}$ and $\tau = \mathcal{O}\big(\sqrt[6]{\frac{n}{\Delta_C}}\big)$, and let $\gamma = \frac{\theta}{12L\kappa}$. Then,*

$$R_2^j \le \mathcal{O}(1)\left(\theta\cdot m\cdot(a_j+a_0) + \theta^{2\ell}\frac{M^2}{L\kappa}\right).$$

Proof. Since Assumption 2 holds true for KROMAGNON, we bound R_2^j as follows:

$$R_2^j = \mathbb{E}\langle\hat{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{v}(\hat{\mathbf{x}}_j, s_j)\rangle \le \gamma\cdot\sum_{i=j-\tau, i\ne j}^{j+\tau}\mathbb{E}\|\mathbf{v}(\hat{\mathbf{x}}_i, s_i)\|\cdot\|\mathbf{v}(\hat{\mathbf{x}}_j, s_j)\|\cdot\mathbb{1}(s_i\cap s_j\ne\emptyset). \tag{A.9}$$

The random variable $\mathbb{1}(s_i \cap s_j \neq \emptyset)$ encodes the sparsity of the gradient steps. As in the proof of Lemma 2.4, we replace $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$ in the sum by $\hat{\mathbf{x}}_{j-3\tau}$. When $j < 3\tau$ we define $\hat{\mathbf{x}}_{j-3\tau} = \mathbf{x}_0$. Since f_{e_i} are L-smooth, we have

$$\|\mathbf{v}(\hat{\mathbf{x}}_j, s_j)\| \leq \|\mathbf{v}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\| + L\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\|,$$
$$\|\mathbf{v}(\hat{\mathbf{x}}_i, s_i)\| \leq \|\mathbf{v}(\hat{\mathbf{x}}_{j-3\tau}, s_i)\| + L\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\|.$$

Then, the expectation of a term $\|\mathbf{v}(\hat{\mathbf{x}}_i, s_i)\| \cdot \|\mathbf{v}(\hat{\mathbf{x}}_j, s_j)\| \cdot \mathbb{1}(s_i \cap s_j)$ in the sum (A.6) is upper bounded by

$$\mathbb{E}\Big\{\Big(\|\mathbf{v}(\hat{\mathbf{x}}_{j-3\tau}, s_i)\|\ \|\mathbf{v}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\| + L^2\|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\|\ \|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\| + L\|\mathbf{v}(\hat{\mathbf{x}}_{j-3\tau}, s_j)\|\ \|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_i\| + L\|\mathbf{v}(\hat{\mathbf{x}}_{j-3\tau}, s_i)\|\ \|\hat{\mathbf{x}}_{j-3\tau} - \hat{\mathbf{x}}_j\|\Big) \cdot \mathbb{1}(s_i \cap s_j)\Big\}.$$

Then, since $\mathbb{E}\mathbb{1}(s_i \cap s_j \neq \emptyset) \leq \frac{2\overline{\Delta}_{\mathrm{C}}}{n}$ (recall that $\overline{\Delta}_{\mathrm{C}}$ is the average conflict degrees), R_2^j can be shown to satisfy the inequality

$$R_2^j \leq \mathcal{O}(1)\sqrt{\frac{\overline{\Delta}_{\mathrm{C}}}{n}}\gamma\tau^3\left(L^2(a_j + a_0) + \theta^{2\ell}M^2\right)$$

as in the proof of Lemma 2.4. The conclusion follows because $\tau = \mathcal{O}\big(\sqrt[6]{\frac{n}{\overline{\Delta}_{\mathrm{C}}}}\big)$ and $\gamma = \frac{\theta}{12L\kappa} = \frac{m\theta}{12L^2}$. ∎

A.2.4. Proof of Theorem 5.4. After plugging in the upper bounds on R_0^j, R_1^j, and R_2^j in the main recursion satisfied by KroMagnon, we find that

$$a_{j+1} \leq \left(1 - \gamma m + \mathcal{O}(1)\left(\gamma^2L^2 + \gamma m\theta^2 + \gamma\theta m\right)\right)a_j + \mathcal{O}(1)\left(\gamma^2L^2 + \gamma m\theta^2 + \gamma\theta m\right)a_0 + \gamma^2\mathcal{O}(1)\theta^{2\ell}M^2 + \gamma\mathcal{O}(1)\theta^{2\ell}\frac{M^2}{L\kappa}.$$

If we set $\gamma = \mathcal{O}(1)\frac{\theta}{L\kappa}$, i.e., the same step size as serial sparse SVRG (Theorem 5.3), then the result becomes

$$a_{j+1} \leq \left(1 - \mathcal{O}(1)\frac{\theta}{\kappa^2}\right)a_j + \mathcal{O}(1)\frac{\theta^2}{\kappa^2}a_0 + \theta^{2\ell+1}\mathcal{O}(1)\frac{M^2}{L^2\kappa^2}$$
$$\leq \left[\left(1 - \mathcal{O}(1)\frac{\theta}{\kappa^2}\right)^{j+1} + \mathcal{O}(1)\theta\right]a_0 + \theta^{2\ell}\mathcal{O}(1)\frac{M^2}{L^2}.$$

We choose $\theta = \mathcal{O}(1) \leq 1/2$ to be a sufficiently small constant, so that the term $\mathcal{O}(1)\theta$ in the brackets is at most 0.5. Then, we can choose $j = \mathcal{O}(1)\kappa^2$, so that the entire coefficient in the brackets is at most 0.75. Finally, we set $\ell = \mathcal{O}(1)\log\left(M^2/L^2\epsilon\right)$, so that the last term is smaller than $\epsilon/8$. Let $\mathbf{y}_k$ be the iterate after the kth epoch, and let $A_k = \mathbb{E}\|\mathbf{y}_k - \mathbf{x}^*\|^2$. Therefore, KroMagnon satisfies the recursion

$$A_{k+1} \leq 0.75 \cdot A_k + \frac{\epsilon}{8} \leq (0.75)^{k+1}A_0 + \frac{\epsilon}{2}.$$

This implies that $\mathcal{O}(1)\log(a_0/\epsilon)$ epochs are sufficient to reach ϵ accuracy, where a_0 is $\|\mathbf{x}_0 - \mathbf{x}^*\|^2$ the initial distance squared to the optimum.

Appendix B. Removing the independence assumption. Our main analysis for Hogwild! relied on the independence between $\hat{\mathbf{x}}_i$ and s_i (Assumption 5). Here,

we show how to lift this assumption, conditional on the fact that each of the f_{e_i} terms is L-smooth. Observe that the following is no longer true: $\mathbb{E}\langle \hat{\mathbf{x}}_i - \mathbf{x}^*, \mathbf{g}(\hat{\mathbf{x}}_i, s_i)\rangle = \mathbb{E}\langle \hat{\mathbf{x}}_i - \mathbf{x}^*, \nabla f(\hat{\mathbf{x}}_i)\rangle$ since we cannot use iterated expectations, precisely due to the lack of independence of the samples and the read variables. However, $\overline{\mathbf{x}}_i$, defined in section 3, is still independent of s_i. We expand our derivation in (2.3) in the following way:

$$a_{j+1} \leq a_j - 2\gamma\mathbb{E}\langle \overline{\mathbf{x}}_i - \mathbf{x}^*, \mathbf{g}(\overline{\mathbf{x}}_j, s_j)\rangle + \gamma^2\mathbb{E}\|g(\hat{\mathbf{x}}_j, \xi_j)\|^2 + 2\gamma\mathbb{E}\langle \overline{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\overline{\mathbf{x}}_j, \xi_j)\rangle + 2\gamma\mathbb{E}\langle \mathbf{x}_j - \mathbf{x}^*, \mathbf{g}(\overline{\mathbf{x}}_j, s_j) - \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle.$$

Since $\overline{\mathbf{x}}_j$ and s_j are independent by construction, we use iterated expectations to get $\mathbb{E}\langle \overline{\mathbf{x}}_i - \mathbf{x}^*, \mathbf{g}(\overline{\mathbf{x}}_j, s_j)\rangle = \mathbb{E}\langle \overline{\mathbf{x}}_i - \mathbf{x}^*, \nabla f(\overline{\mathbf{x}}_j)\rangle$. As before, the strong convexity of f and the triangle inequality imply that $\langle \overline{\mathbf{x}}_j - \mathbf{x}^*, \nabla f(\overline{\mathbf{x}}_j)\rangle \geq \frac{m}{2}\|\mathbf{x}_j - \mathbf{x}^*\|^2 - m\|\overline{\mathbf{x}}_j - \mathbf{x}_j\|^2$. Putting everything together we get the following recursion for the sequence a_j:

$$a_{j+1} \leq (1-\gamma m)a_j + \gamma^2 \underbrace{\mathbb{E}\|g(\hat{\mathbf{x}}_j, \xi_j)\|^2}_{R_0^j} + 2\gamma m \underbrace{\mathbb{E}\|\overline{\mathbf{x}}_j - \mathbf{x}_j\|^2}_{R_1^j} + 2\gamma \underbrace{\mathbb{E}\langle \overline{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\overline{\mathbf{x}}_j, \xi_j)\rangle}_{R_2^j} + 2\gamma \underbrace{\mathbb{E}\langle \mathbf{x}_j - \mathbf{x}^*, \mathbf{g}(\overline{\mathbf{x}}_j, s_j) - \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle}_{R_3^j}.$$

The reader can verify that although R_1^j and R_2^j are defined now in terms of $\overline{\mathbf{x}}_j$, the upper bounds derived in section 3 still hold. We bound R_3^j as follows

$$\mathbb{E}\langle \mathbf{x}_j - \mathbf{x}^*, \mathbf{g}(\overline{\mathbf{x}}_j, s_j) - \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\rangle \leq \mathbb{E}\|\mathbf{x}_j - \mathbf{x}^*\|\|\mathbf{g}(\overline{\mathbf{x}}_j, s_j) - \mathbf{g}(\hat{\mathbf{x}}_j, s_j)\| \leq \frac{ma_j}{4} + \frac{L^2}{m}\mathbb{E}\|\overline{\mathbf{x}}_j - \hat{\mathbf{x}}_j\|^2,$$

where the last inequality follows by the smoothness of the gradient steps and the arithmetic-geometric mean inequality. Therefore

$$a_{j+1} \leq \left(1 - \frac{\gamma m}{2}\right) a_j + \gamma^2 \underbrace{\mathbb{E}\|g(\hat{\mathbf{x}}_j, \xi_j)\|^2}_{R_0^j} + 2\gamma m \underbrace{\mathbb{E}\|\overline{\mathbf{x}}_j - \mathbf{x}_j\|^2}_{R_1^j} + 2\gamma \underbrace{\mathbb{E}\langle \overline{\mathbf{x}}_j - \mathbf{x}_j, \mathbf{g}(\overline{\mathbf{x}}_j, \xi_j)\rangle}_{R_2^j} + 2\gamma\frac{L^2}{m} \underbrace{\mathbb{E}\|\overline{\mathbf{x}}_j - \hat{\mathbf{x}}_j\|^2}_{R_4^j}.$$

Since we can upper bound R_4^j by the same bound derived for R_3^j, we obtain the following convergence result for HOGWILD!.

THEOREM B.1. *If the number of samples that overlap in time with a single sample during the execution of* HOGWILD! *is bounded as* $\tau = \mathcal{O}\left(\min\left\{n/\Delta_C, M^2/\epsilon L^2\right\}\right)$, *then* HOGWILD! *with step size* $\gamma = \mathcal{O}(1)\frac{\epsilon m}{M^2}$ *reaches an accuracy of* $\mathbb{E}\|\mathbf{x}_k - \mathbf{x}^*\|^2 \leq \epsilon$ *after* $T \geq \mathcal{O}(1)\frac{M^2\log(a_0/\epsilon)}{\epsilon m^2}$ *iterations.*

The only difference between this result and that in our main section is that we can guarantee speedup for a smaller range of τ.

REFERENCES

[1] *LIBSVM Data: Classification (binary class)*, http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[2] A. AGARWAL AND J. C. DUCHI, *Distributed delayed stochastic optimization*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2011, pp. 873–881.

[3] H. Avron, A. Druinsky, and A. Gupta, *Revisiting asynchronous linear solvers: Provable convergence rate through randomization*, in Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, 2014, pp. 198–207.
[4] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice–Hall, Englewood Cliffs, NJ, 1989.
[5] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, *Ubicrawler: A scalable fully distributed web crawler*, Software: Practice & Experience, 34 (2004), pp. 711–726.
[6] P. Boldi, M. Rosa, M. Santini, and S. Vigna, *Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks*, in Proceedings of the 20th International Conference on World Wide Web, 2011, pp. 587–596.
[7] P. Boldi and S. Vigna, *The WebGraph framework* I*: compression techniques*, in Proceedings of the 13th International Conference on World Wide Web, 2004, pp. 595–602.
[8] S. Bubeck, *Convex optimization: Algorithms and complexity*, Found. Trends Mach. Learn., 8 (2015), pp. 231–357.
[9] D. Chazan and W. Miranker, *Chaotic relaxation*, Linear Algebra and Appl., 2 (1969), pp. 199–222.
[10] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, *Project Adam: Building an efficient and scalable deep learning training system*, in 11th USENIX Symposium on Operating Systems Design and Implementation, 2014, pp. 571–582.
[11] 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, MIT Press, Cambridge, MA, 2012, pp. 1223–1231.
[12] J. Duchi, M. I. Jordan, and B. McMahan, *Estimation, optimization, and parallelism when data is sparse*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2013, pp. 2832–2840.
[13] H. R. Feyzmahdavian, A. Aytekin, and M. Johansson, *An asynchronous mini-batch algorithm for regularized stochastic optimization*, in Proceedings of the IEEE Conference on Decision and Control, 2015, pp. 1384–1389.
[14] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, *Large-scale matrix factorization with distributed stochastic gradient descent*, in Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2011, pp. 69–77.
[15] E. Hazan and S. Kale, *Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization*, J. Mach. Learn. Res., 15 (2014), pp. 2489–2512.
[16] M. Hong, *A Distributed, Asynchronous and Incremental Algorithm for Nonconvex Optimization: An ADMM Approach*, IEEE Trans. on Control of Network Systems, 2017.
[17] C.-J. Hsieh, H.-F. Yu, and I. S. Dhillon, *Passcode: Parallel Asynchronous Stochastic Dual Coordinate Descent*, Proceedings of the 32nd International Conference on Machine Learning, 2015.
[18] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, *Communication-efficient distributed dual coordinate ascent*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2014, pp. 3068–3076.
[19] R. Johnson and T. Zhang, *Accelerating stochastic gradient descent using predictive variance reduction*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2013, pp. 315–323.
[20] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, *RCV*1*: A new benchmark collection for text categorization research*, J. Mach. Learn. Res., 5 (2004), pp. 361–397.
[21] X. Lian, Y. Huang, Y. Li, and J. Liu, *Asynchronous parallel stochastic gradient for nonconvex optimization*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2015, pp. 2737–2745.
[22] J. Liu, S. Wright, C. Re, V. Bittorf, and S. Sridhar, *An asynchronous parallel stochastic coordinate descent algorithm*, in Proceedings of the 31st International Conference on Machine Learning, 2014, pp. 469–477.
[23] J. Liu and S. J. Wright, *Asynchronous stochastic coordinate descent: Parallelism and convergence properties*, SIAM J. Optim., 25 (2015), pp. 351–376, https://doi.org/10.1137/140961134.
[24] J. Liu, S. J. Wright, and S. Sridhar, *An Asynchronous Parallel Randomized Kaczmarz Algorithm*, preprint, https://arxiv.org/abs/1401.4780, 2014.
[25] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, *Identifying suspicious URLs: An application of large-scale online learning*, in Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 681–688.

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

[27] X. Pan, D. Papailiopoulos, S. Oymak, B. Recht, K. Ramchandran, and M. I. Jordan, *Parallel correlation clustering on big graphs*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2015, pp. 82–90.

[28] Z. Peng, Y. Xu, M. Yan, and W. Yin, *Arock: An algorithmic framework for asynchronous parallel coordinate updates*, SIAM J. Sci. Comput., 38 (2016), pp. A2851–A2879, https://doi.org/10.1137/15M1024950.

[29] B. Recht, C. Re, J. Tropp, and V. Bittorf, *Factoring nonnegative matrices with linear programs*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2012, pp. 1214–1222.

[30] P. Richtárik and M. Takáč, *Parallel coordinate descent methods for big data optimization*, Math. Program., 156 (2016), pp. 433–484.

[31] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans, *Distributed asynchronous deterministic and stochastic gradient optimization algorithms*, IEEE Trans. Automat. Control, 31 (1986), pp. 803–812.

[32] Y.-X. Wang, V. Sadhanala, W. Dai, W. Neiswanger, S. Sra, and E. P. Xing, *Parallel and distributed block-coordinate Frank-Wolfe algorithms*, Proceedings of the 33rd International Conference on Machine Learning, 2016, pp. 1548–1557.

[33] H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon, *Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion*, Proc. VLDB Endowment, 7 (2014), pp. 975–986.

[34] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin, *A fast parallel SGD for matrix factorization in shared memory systems*, in Proceedings of the 7th ACM conference on Recommender Systems, 2013, pp. 249–256.

[35] M. Zinkevich, J. Langford, and A. J. Smola, *Slow learners are fast*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2009, pp. 2331–2339.

[36] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, *Parallelized stochastic gradient descent*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2010, pp. 2595–2603.