Reducing Communication in Proximal Newton Methods for Sparse Least Squares Problems

Saeed Soori
Rutgers University
saeed.soori@rutgers.edu

Aditya Devarakonda
University of California Berkeley
aditya@eecs.berkeley.edu

Zachary Blanco
Rutgers University
zac.blanco@rutgers.edu

James Demmel
University of California Berkeley
demmel@eecs.berkeley.edu

Mert Gurbuzbalaban
Rutgers University
mert.gurbuzbalaban@rutgers.edu

Maryam Mehri Dehnavi
Rutgers University
maryam.mehri@rutgers.edu

ABSTRACT

Proximal Newton methods are iterative algorithms that solve l1-regularized least squares problems. Distributed-memory implementation of these methods have become popular since they enable the analysis of large-scale machine learning problems. However, the scalability of these methods is limited by the communication overhead on modern distributed architecture. We propose a stochastic variance-reduced proximal method along with iteration-overlapping and Hessian-reuse to find an efficient trade-off between computation complexity and data communication. The proposed RC-SFSITA algorithm reduces latency costs by a factor of k without altering bandwidth costs. RC-SFISTA is implemented on both MPI and Spark and compared to the state-of-the-art framework, ProxCoCoA. The performance of RC-SFISTA is evaluated on 1 to 512 nodes for multiple benchmarks and demonstrates speedups of up to 12× compared to ProxCoCoA with scaling properties that outperform the original algorithm.

ACM Reference Format:
Saeed Soori, Aditya Devarakonda, Zachary Blanco, James Demmel, Mert Gurbuzbalaban, and Maryam Mehri Dehnavi. 2018. Reducing Communication in Proximal Newton Methods for Sparse Least Squares Problems. In *ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3225058.3225131

1 INTRODUCTION

Mathematical optimization is one of the main pillars of machine learning where parameters of an optimization problem are computed based on observed data. A popular approach to estimate parameters in convex optimization problems is solving a regularized least squares problem [24, 29, 30] using proximal methods [11, 28]. Since classical optimization methods are iterative by nature, when operating on large amounts of distributed data, the processors need to communicate at each iteration. On modern computer architectures the *communication cost*, i.e. the cost of moving data between levels of the memory hierarchy or between processors over a network, is often orders of magnitude larger than *computation cost*, i.e. the cost of floating point operations (flops), and this gap is increasing [15]. Thus optimization methods should be reformulated to maintain an efficient trade-off between the algorithm's *computational complexity*, i.e. the total number of arithmetic or logical operations an algorithm performs, and data communication.

A common approach to improving the performance of optimization methods on distributed platforms is to reformulate the algorithm to reduce the number of iterations needed to reach the optimal solution. The methods are often referred as *communication-efficient* methods. Work such as [23, 31] attempt to improve the convergence properties by applying different solvers to locally stored data. CoCoA [31] uses a local solver on each machine and shares information between solvers with highly flexible communication schemes. ProxCoCoA [30], GLMNET [16], and BLITZ [20] propose communication-efficient algorithms for proximal methods, however, the methods do not necessarily preserve the exact arithmetic of the conventional algorithm.

Iteration-overlapping techniques [1, 6] reduce the overall communication cost in optimization methods by unrolling iterations in the algorithm to break the dependencies between vector updates. These work produce a solution identical to that of the conventional algorithm in exact arithmetic. k-step Krylov solvers [7, 10] compute k basis vectors at-once by unrolling k iterations of the standard algorithm. P-packSVM [35] proposes a SGD-based SVM algorithm which communicates every k iterations. CA-BCD [13] reduces communication for the class of l2-regularization least squares problems by rearranging the computations to execute k iterations per *communication round*, i.e. the total number of times that processors exchange *messages* over a network. SA-accBCD [14] applies a similar approach to proximal least squares problems. While these work reduce communication costs by reducing the number of communication rounds, they increase the amount of communicated data at each round, i.e. *message size*. Also, since these methods are deterministic, every iteration operates on all the data which leads to high computational complexity for *overdetermined* problems.

Overdetermined problems involving a large number of data points are often solved with random sampling, i.e. a stochastic approach, to reduce the overall computational complexity of the algorithm with reducing the size of data that the algorithm operates on. Examples of such work include stochastic formulations of gradient descent [18, 21] and proximal gradient methods [19]. However, the rate of convergence of a basic stochastic method is

ICPP 2018, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6510-9/18/08…$15.00
https://doi.org/10.1145/3225058.3225131

slower than that of the deterministic algorithm due to the variance introduced by random sampling [5]. Stochastic proximal methods such as Acc-Prox-SVRG [26] and Prox-SVRG [34] propose variance-reduction techniques to overcome this challenge. Even though the computational complexity of stochastic optimization methods is lower than deterministic formulations, the performance of stochastic methods are often bound by data communication required at each iteration of the algorithm.

This paper reformulates *Proximal Newton (PN)* methods for overdetermined problems to improve their performance on distributed platforms. PN methods use a first-order inner solver, which is the *Fast Iterative Shrinkage-Thresholding Algorithm* (FISTA) [2] in this work, to solve a subproblem iteratively at each iteration. We propose a *Reduced-Communication* Stochastic FISTA algorithm (RC-SFISTA) that reduces the communication cost of the inner solver in PN methods by overlapping k iterations without changing the message size. RC-SFISTA uses a stochastic variance-reduced method to also reduce the computational complexity of PN methods. The contributions of the paper are as follows:

- A novel stochastic variance-reduced formulation of the FISTA algorithm, called SFISTA, is introduced that reduces the computation complexity of PN methods. A convergence theorem is provided for the stochastic variance-reduced method which shows the same convergence rate as its deterministic formulation.
- Iterations in the proposed SFISTA method are overlapped to reduce communication rounds and latency costs without increasing the number of messages.
- A Hessian-reuse algorithm is developed that enables the reuse of data when solving a subproblem in SFISTA. The parameter S in the Hessian-reuse methods enables finding an efficient trade-off between computational complexity and communication costs in the algorithm.
- The upper bounds for parameters k in iteration-overlapping and S in the Hessian-reuse method are provided with respect to machine and algorithm specifications.
- RC-SFISTA is implemented on both MPI and Spark and is compared to the state-of-the-art framework ProxCoCoA. RC-SFISTA outperforms the classical method up to 12× with MPI on 256 processors. We also demonstrate that RC-SFISTA performs better than ProxCoCoA up to 12× on 256 workers for the tested datasets.

2 BACKGROUND

This section introduces PN methods for solving a class of optimization problems that arise frequently in machine learning applications. The performance model used in this work to evaluate the effectiveness of the proposed reformulations will also be discussed.

2.1 The proximal Newton method

Composite optimization problems arise frequently in machine learning and data analytics applications. For example, consider:

$$\min_{w \in \mathbb{R}^d} \; F(w) \equiv f(w) + g(w) \tag{1}$$

where $g : \mathbb{R}^d \to \mathbb{R}$ is a continuous, convex, possibly non-smooth function and $f : \mathbb{R}^d \to \mathbb{R}$ is a convex function, twice continuously differentiable, with an L-Lipschitz gradient, expressed as:

$$f(w) = \frac{1}{m} \sum_{i=1}^{m} f_i(w) \tag{2}$$

This is a general class of problems that includes numerous machine learning problems including logistic regression and regularized least squares problems, or more general *empirical risk minimization* problems.

PN methods shown in Algorithm 1 are used to solve the optimization problem in (1). These methods could be seen as generalization of the classical proximal gradient methods where the curvature of the function, i.e. Hessian, is used to select a search direction [22]. PN methods define a subproblem at each iteration which can be minimized using a first-order inner solver shown in line-4 of Algorithm 1. First-order methods that use *proximal mapping* to handle the non-smooth segment of the objective function in (1) are very popular and perform well in practice [4, 8].

Algorithm 1: Proximal Newton Method

1 **Input:** w_0, $\{\gamma_n\}$
2 **repeat**
3 Update H_n, an approximation of Hessian
4 $z_n = \underset{y}{\operatorname{argmin}} \frac{1}{2}(y - w_n)^T \boldsymbol{H_n}(y - w_n) + \nabla f(w_n)^T (y - w_n) + g(y)$ } Inner Solver
5 $\Delta w_n = z_n - w_n$
6 $w_{n+1} = w_n + \gamma_n \Delta w_n$
7 **until** *Stopping conditions are satisfied*
8

L1-regularized least squares problem. The general problem in (1) can represent a large class of regression problems. In particular, we focus on the l1-regularized least squares problem:

$$f(w) = \frac{1}{2m} \sum_{i=1}^{m} (x_i^T w - y_i)^2, \quad g(w) = \lambda \|w\|_1 \tag{3}$$

where $x_i \in \mathbb{R}^d$ is the i-th data point and $y_i \in \mathbb{R}$ is the corresponding label. We set $X = [x_1, ..., x_m] \in \mathbb{R}^{d \times n}$ as the input data matrix, where rows are the features and columns are the samples, and $y \in \mathbb{R}^n$ holds the labels. $w \in \mathbb{R}^d$ is the optimization variable, and $\lambda \in \mathbb{R}$ is the regularization (penalty) parameter. In this case, the gradient and Hessian of $f(w)$ is given by:

$$\nabla f(w) = \frac{1}{m}(XX^T w - Xy) \quad H = \frac{1}{m} XX^T \tag{4}$$

where $H \in \mathbb{R}^{d \times d}$ is the Hessian and with defining $R \in \mathbb{R}^d$ as $R = \frac{1}{m} Xy$, for the l1-regularized least squares problem the gradient of f will be

$$\nabla f(w) = Hw - R \tag{5}$$

2.2 The inner solver

The subproblem in PN methods can be solved using a first-order method. While inner solvers like coordinate descent [33] are used in PN methods, this work uses FISTA which is the most popular method in the family of accelerated proximal methods. FISTA has

the same convergence rate as the accelerated coordinate descent methods [22]. The FISTA algorithm is shown in Algorithm 2 where the proximal mapping is defined as:

Algorithm 2: FISTA

1 **Input:** $w_0 = w_{-1} = \mathbf{0} \in \mathbb{R}^d$, $t_0 = 1$, γ.
2 **for** $n = 1, \ldots, N$ **do**
3 $\quad t_n = \frac{1+\sqrt{1+t_{n-1}^2}}{2}$
4 $\quad v_n = w_{n-1} + \frac{t_{n-1}-1}{t_n}(w_{n-1} - w_{n-2})$
5 $\quad w_n = \text{Prox}_\gamma(v_n - \gamma \nabla f(v_n))$
6 **output** w_N

$$\text{Prox}_\gamma(w) = \underset{x}{\text{argmin}}\{\frac{1}{2\gamma}\|x - w\|^2 + g(x)\} \tag{6}$$

In this work, we use FISTA as an inner solver for PN methods. We focus on optimizing the performance of FISTA for l1-regularized least squares problems and demonstrate that improving the inner solver performance improves the performance of the PN method by reducing its overall computation and communication cost.

2.3 Performance model

This work proposes novel formulations of the PN solvers to improve their performance when executed on distributed hardware platforms. The following elaborates the performance model used to demonstrate the effectiveness of the proposed method. The cost of an algorithm includes arithmetic and computation. Traditionally, algorithms have been analyzed with floating point operation costs. However, communication costs are essential in analyzing algorithms in large-scale simulations [17]. The cost of floating point operations and communication, including bandwidth and latency, can be combined to obtain the performance model. In this work we use a simplified model known as the $\alpha - \beta$ model [12]. In distributed-memory settings, communication cost includes the "bandwidth cost", i.e. the number of words sent either between levels of a memory hierarchy or between processors over a network, and the "latency cost", i.e. the number of messages sent, where a message either consists of a group of contiguous words being sent or is used for interprocess synchronization. The total execution time of an algorithm is:

$$T = \gamma F + \alpha L + \beta W \tag{7}$$

where T is the overall execution time and γ, α, and β are machine-specific parameters that represent the cost of one floating point operation, the cost of sending a message, and the cost of moving a word. Parameters F, L, and W represent the number of flops, the number of messages communicated between processors, and the number of words moved respectively.

3 THE REDUCED-COMMUNICATION STOCHASTIC FISTA (RC-SFISTA)

We propose RC-SFISTA that improves the performance of the FISTA algorithm as well as PN methods on distributed platforms. RC-SFISTA is developed to maintain an efficient trade-off between computation and communication costs in PN methods. We first introduce a novel variance-reduced stochastic formulation of FISTA (SFISTA) to reduce its computational complexity with random sampling. The communication cost of the formulated SFISTA is then reduced in a subsequent step with iteration-overlapping. A Hessian-reuse method is also proposed that provides an effective trade-off between computational complexity and communication costs for the inner solvers in SFISTA. The extension and applicability of the proposed RC-SFISTA method for second-order optimization methods, specifically PN methods, are elaborated at the end.

3.1 Reducing computational complexity with a stochastic formulation

We reduce the computational complexity of FISTA with random sampling at each iteration to significantly decrease the number of floating point operations in the algorithm. The stochastic variance-reduced FISTA (SFISTA) algorithm is developed for the general optimization problem in (1) and afterwards for the l1-regularized least squares problem. FISTA is made stochastic by estimating the gradient in line 5 of Algorithm 2 with:

$$\nabla \hat{f}(v_n) = \frac{1}{\bar{m}} \sum_{i \in \mathbb{I}_n} \nabla f_i(v_n) \tag{8}$$

where $\mathbb{I}_n$ is a subset of size $\bar{m} = \lfloor bm \rfloor$ from $1, \ldots, m$ chosen uniformly at random and $0 < b < 1$ is the sampling rate. Even though estimating the gradient in (8) reduces the computational complexity of FISTA, the convergence rate of the stochastic method is slower due to the variance introduced by sampling. Therefore, we use a variance-reduction method where the gradient is estimated by:

$$\nabla \hat{f}(v_n) = \frac{1}{\bar{m}} \left(\sum_{i \in \mathbb{I}_n} \nabla f_i(v_n) - \sum_{i \in \mathbb{I}_n} \nabla f_i(\hat{w}_s) \right) + \nabla f(\hat{w}_s) \tag{9}$$

and $\hat{w}_s$ is the value of w in every N iterations. The last term in (9) computes the full gradient using all data samples at every N iteration which reduces the variance and allows us to preserve the convergence rate of FISTA. The SFISTA algorithm is shown in Algorithm 3. An approach similar to [27] can be used to prove the convergence of SFISTA. The main theorem which guarantees the convergence is as follows:

Theorem 1. Consider Algorithm 3 with mini-batch size of $\bar{m}$ and the Lipschitz constant L where the step size γ is a positive number such that:

$$\gamma^{-1} \geq \max\left(\frac{L}{2} + \sqrt{\frac{1}{4} + \frac{4L^2(m - \bar{m})}{\bar{m}(m-1)}}, L \right) \tag{10}$$

and

$$\gamma < \left(1 - \frac{t_{N-1}^2}{t_N^2}\right) \frac{\bar{m}(m-1)}{8L(m - \bar{m})} \tag{11}$$

then,

$$\mathbb{E}[F(w_N)] - F(w^*) \leq \frac{\mathbb{E}[F(\hat{w}_s)] - F(w^*)}{t_N^2(1-\eta)^N} + \frac{C^2}{2\gamma t_N^2 (1-\eta)^N} \tag{12}$$

for some $C \geq 0$ and $0 < \eta < 1 - \frac{t_{N-1}^2}{t_N^2}$. $\mathbb{E}[]$ denotes the expectation with respect to $\mathbb{I}_n$ and w^* is the optimal solution to problem (1). Since $t_N = O(N)$, η can be chosen to be close enough to 0 so that SFISTA converges to the optimal solution with a rate of $O(1/N^2)$:

$$\mathbb{E}[F(w_N)] - F(w^*) \leq \frac{C_1}{N^2}\left(\mathbb{E}[F(\hat{w}_s)] - F(w^*)\right) + \frac{C_2^2}{\gamma N^2} \tag{13}$$

for positive constants C_1 and C_2. Also, with additional constraints, such as strong convexity of the objective function, a better rate for stochastic PN methods with variance-reduction is obtainable [34].

Algorithm 3: SFISTA

1 **Input:** $\hat{w}_0 = w_{-1} = \mathbf{0} \in \mathbb{R}^d$, $t_0 = 1$, γ
2 **for** $s = 0, \ldots$ **do**
3 $w_0 = \hat{w}_s$
4 **for** $n = 1, \ldots, N$ **do**
5 $t_n = \frac{1+\sqrt{1+t_{n-1}^2}}{2}$
6 $v_n = w_{n-1} + \frac{t_{n-1}-1}{t_n}(w_{n-1} - w_{n-2})$
7 $w_n = \text{Prox}_\gamma(v_n - \gamma \nabla \hat{f}(v_n))$
8 $\hat{w}_{s+1} = w_N$
9 **output** w_N

The total number of floating point operations for the proposed stochastic formulation of FISTA with a sampling rate of b is reduced by a factor of $1/b$ which reduces the computational complexity without changing the convergence rate.

SFISTA for the l1-regularized least squares problem. This work focuses on the problem of l1-regularized least squares where the gradient is computed based on (4) and the gradient and the Hessian are related by (5). As seen in (9), the gradient is estimated with random sampling of the input data at each iteration. For l1-regularized least squares since the non-smooth function is a l1-norm operator, we write the update in line-7 of Algorithm 3 as:

$$w_n = \mathbb{S}_{\lambda\gamma}(v_n - \gamma \nabla \hat{f}(v_n)) \tag{14}$$

where $\mathbb{S}_\alpha(\beta) = sign(\beta)\max(|\beta| - \alpha, 0)$. We also use the same symbol $\mathbb{I}_n$ to represent the sampling matrix $\mathbb{I}_n = [e_{i_1}, e_{i_2}, ..., e_{i_{\bar{m}}}] \in \mathbb{R}^{m \times \bar{m}}$ where $\{i_h \in [m] | h = 1, ..., \bar{m}\}$ is chosen uniformly at random and $e_i \in \mathbb{R}^m$ is all zeros except entry i. Finally, to simplify SFISTA, we rewrite the update for v_n as:

$$v_n = w_{n-1} + \mu_n(w_{n-1} - w_{n-2}) \tag{15}$$

where $\mu_n = \frac{t_{n-1}-1}{t_n}$. The SFISTA algorithm for l1-regularized least squares problem is shown in Algorithm 4.

As demonstrated, SFISTA for l1-regularized least squares converges to the optimal solution with a rate of $O(1/N^2)$. Therefore, it keeps the convergence rate of FISTA and reduces number of required floating point operations at each iteration.

3.2 Reducing communication costs with overlapping iterations and Hessian-reuse

SFISTA solves the l1-regularized least squares problem by iteratively computing a Hessian matrix and thereafter updating local variables. Since SFISTA communicates data at each iteration, it becomes communication-bound when ran on distributed memory systems for large datasets. We reduce the communication overhead of SFISTA for the l1-regularized least squares problem by (1) proposing an iteration-overlapping technique that reduces latency costs in SFISTA by $O(k)$ without altering bandwidth costs and convergence behavior; (2) and introducing a subproblem that reduces total number of iterations N needed for SFISTA to convergence which results in lower bandwidth and latency costs. The combination of these techniques, which we call the *Reduced-Communication SFISTA (RC-SFISTA)* algorithm, reduces the number of iterations in the algorithm that require data communication over the network.

Algorithm 4: SFISTA for l1-regularized least squares problem

1 **Input:** $\hat{w}_0 = w_{-1} = \mathbf{0} \in \mathbb{R}^d$, $t_0 = 1$, γ
2 **for** $s = 0, \ldots$ **do**
3 $w_0 = \hat{w}_s$
4 **for** $n = 1, \ldots, N$ **do**
5 $t_n = \frac{1+\sqrt{1+t_{n-1}^2}}{2}$
6 $\mu_n = \frac{t_{n-1}-1}{t_n}$
7 $v_n = w_{n-1} + \mu_n(w_{n-1} - w_{n-2})$
8 $g_n = \frac{1}{m}\left(X\mathbb{I}_n\mathbb{I}_n^T X^T v_n - X\mathbb{I}_n\mathbb{I}_n^T y\right)$
9 $\theta_n = v_n - \gamma g_n$
10 $w_n = \mathbb{S}_{\lambda\gamma}(\theta_n)$
11 $\hat{w}_{s+1} = w_N$
12 **output** w_N

Latency, arithmetic, and bandwidth costs in SFISTA. We use the model discussed in (7) to analyze the performance of SFISTA. Table 1 summarizes the latency, bandwidth, and flop cost of SFISTA. As demonstrated, the performance and scalability of SFISTA is limited by latency and bandwidth costs which both increase with number of iterations N and number of processors P. The RC-SFISTA formulation proposed in this section reduces the latency cost by a factor of $O(k)$ while preserving the bandwidth cost.

Overlapping iterations in SFISTA to reduce latency costs. We propose a novel formulation of SFISTA that allows for iterations of the algorithm to be overlapped to reduce communication costs. Available work that overlap iterations in some optimization methods, they do not support FISTA, reduce latency costs at the expense of increasing the amount of data moved among processors over a network, i.e. message cost. However, our reformulation of SFISTA does not alter the message costs and provides an $O(k)$ reduction in latency costs of the algorithm with overlapping iterations. By leveraging the fact that multiple instances of the Hessian could be generated at once, we unroll k-consecutive iterations in SFISTA. The recurrence updates in SFISTA should be unrolled for k iterations so that updates to the optimization variable can be postponed for k iterations. Lets define $\Delta w_n = \mathbb{S}_{\lambda\gamma}(\theta_n) - w_{n-1}$. We start by changing the loop index in Algorithm 4 from n to $nk + j$ where n is the outer loop index, k is the recurrence unrolling parameter, and j is inner loop index. Assuming we are at iteration $nk + 1$ and v_{nk}, g_{nk}, θ_{nk} and Δw_{nk} have been computed and if $\Delta v_{nk} = v_{nk+1} - v_{nk}$, then the updates for the next iteration are:

Table 1: Latency, flops, and bandwidth costs for N iterations of RC-SFISTA and SFISTA. Parameters d, $\tilde{m}$, k, and S represent # columns, # sampled rows, the iteration-overlapping parameter, and the inner loop parameter; f is the matrix non-zero fill-in.

Algorithm	Latency cost (L)	Flops cost (F)	Bandwidth cost (W)
SFISTA	$O(Nlog(P))$	$O\left(\frac{Nd^2\tilde{m}f}{P}\right)$	$O(Nd^2log(P))$
RC-SFISTA	$O\left(\frac{Nlog(P)}{k}\right)$	$O\left(\frac{Nd^2\tilde{m}f}{P} + Sd^2\right)$	$O\left(Nd^2log(P)\right)$

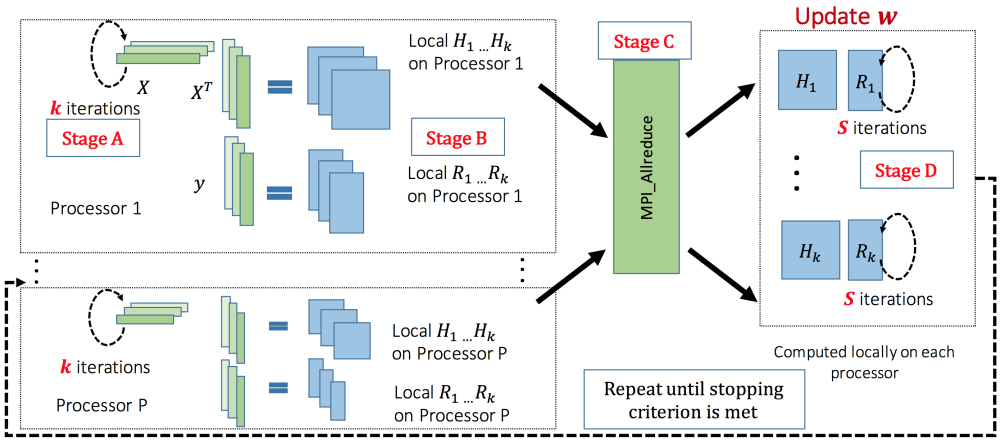


Figure 1: A high-level description of RC-SFISTA implemented on a distributed memory system of P processors.

$$\begin{aligned}\Delta w_{nk+1} &= \mathbb{S}_{\lambda\gamma}(\theta_{nk+1}) - w_{nk}\\ g_{nk+1} &= \frac{1}{\tilde{m}}\left[X\mathbb{I}_{nk+1}\mathbb{I}_{nk+1}^T X^T(v_{nk} + \Delta v_{nk}) - X\mathbb{I}_{nk+1}\mathbb{I}_{nk+1}^T y\right]\\ \theta_{nk+1} &= v_{nk} + \Delta v_{nk} - \gamma g_{nk+1}\end{aligned}$$

thus, for iteration $nk+2$ the updates are computed as follows:

$$\begin{aligned}\Delta w_{nk+2} &= \mathbb{S}_{\lambda\gamma}(\theta_{nk+2}) - (w_{nk} + \Delta w_{nk+1})\\ g_{nk+2} &= \frac{1}{\tilde{m}}\left[X\mathbb{I}_{nk+2}\mathbb{I}_{nk+2}^T X^T(v_{nk} + \Delta v_{nk} + \Delta v_{nk+1})\right.\\ &\quad \left. - X\mathbb{I}_{nk+2}\mathbb{I}_{nk+2}^T y\right]\\ \theta_{nk+2} &= v_{nk} + \Delta v_{nk} + \Delta v_{nk+1} - \gamma g_{nk+2}\end{aligned}$$

Therefore, by induction we will have

$$g_{nk+j} = \frac{1}{\tilde{m}}\left[X\mathbb{I}_{nk+j}\mathbb{I}_{nk+j}^T X^T\left(v_{nk} + \sum_{i=0}^{j-1}\Delta v_{nk+i}\right) - X\mathbb{I}_{nk+j}\mathbb{I}_{nk+j}^T y\right]$$

$$\theta_{nk+j} = v_{nk} + \sum_{i=0}^{j-1}\Delta v_{nk+i} - \gamma g_{nk+j}$$

$$\Delta w_{nk+j} = \mathbb{S}_{\lambda\gamma}(\theta_{nk+j}) - (w_{nk} + \sum_{i=1}^{j-1}\Delta w_{nk+i}) \quad (16)$$

and Δv_{n+k} is obtained via

$$\begin{aligned}\Delta v_{nk+j} &= (1+\mu_{nk+j+1})\left[w_{nk+j} - w_{nk+j-1}\right]\\ &\quad -\mu_{nk+j}\left[w_{nk+j-1} - w_{nk+j-2}\right] \quad (17)\\ &= (1+\mu_{nk+j+1})\Delta w_{nk+j} - \mu_{nk+j}\Delta w_{nk+j-1}\end{aligned}$$

With this approach, updates are postponed for k consecutive iterations. Communication in (16) is avoided by computing the following matrices for k iterations and storing them on all processors:

$$H_{nk+j} = \frac{1}{\tilde{m}}X\mathbb{I}_{nk+j}\mathbb{I}_{nk+j}^T X^T, \quad R_{nk+j} = \frac{1}{\tilde{m}}X\mathbb{I}_{nk+j}\mathbb{I}_{nk+j}^T y \quad (18)$$

H_n is the approximated Hessian at iteration n for the l1-regularized least squares problem and the processors over the network communicate only every k iterations. This will reduce the number of messages transfered by a factor of $O(k)$. The resulting method does not change the convergence of SFISTA and keeps the same number of arithmetic operations.

The RC-SFISTA algorithm is shown in Algorithm 5. Algorithm parts that relate to implementing iteration-overlapping is highlighted in red. As shown, the number of iterations in line 2 is reduced by a factor of k. The inner loop in line 3 computes local matrices H and R and the inner loop at line 7 updates local variables without communication. Table 1 shows the latency, bandwidth, and flop cost of RC-SFISTA. With iteration-overlapping, the latency of SFISTA is reduced by a factor of k while bandwidth costs do not change which can potentially lead to a k-fold speedup.

The Hessian-reuse method. We propose a novel technique called *Hessian-reuse* to further reduce the number of outer iterations in RC-SFISTA. The objective of Hessian-reuse, which solves a local subproblem, is to find an efficient trade-off between data communication and *local operations*, operations that are computed on the same processor and do not lead to inter-processor data communication. By solving a subproblem local to each processor, shown in lines 9-15 of Algorithm 5, we expect the overall problem to converge faster, i.e. the number of iterations corresponding to the for-loop in

line 2 which require inter-processor data communication to reduce. The following discusses the Hessian-reuse method and analyzes why a better convergence is expected with this formulation.

Every update in SFISTA requires matrices H_n and R_n which can be reused repeatedly to update local variables. Reusing the Hessian could contribute more to minimizing the objective function and in particular it will reduce the total number of iterations N in SFISTA. Since both latency and bandwidth costs increase with the number of iterations, this approach can reduce communication overhead. Lets first see the subproblem in PN methods which is provided by:

$$\begin{aligned} z_k &= \underset{y}{\operatorname{argmin}} \frac{1}{2}(y - w_n)^T \boldsymbol{H_n}(y - w_n) + \nabla f(w_n)^T (y - w_n) + g(y) \\ &= \underset{y}{\operatorname{argmin}} \Phi(y) + g(y) \end{aligned} \tag{19}$$

where $\Phi(y)$ is the smooth segment of the objective function. To solve this minimization problem, RC-SFISTA requires the gradient of the smooth segment of this objective problem given by $\nabla\Phi(y) = H_n y - R_n$ which has the same equation for the gradient in l1-regularized least squares problem in (5). This means applying SFISTA to solve the subproblem at (19) is identical to applying the SFISTA recurrence updates while using the same H_n and R_n to compute the gradient.

While we apply iteration-overlapping, the single update rules for SFISTA (lines 6-10 Algorithm 4) are changed into a new inner loop update. In other words, the update rules are changed to:

$$\Delta v^s_{nk+j} = (1 + \mu_{s+1})\Delta w^s_{nk+j} - \mu_s \Delta w^s_{nk+j-1} \tag{20}$$

$$g^s_{nk+j} = H_{nk+j}\left(v_{nk} + \sum_{i=0}^{j-1} \Delta v^s_{nk+i}\right) - R_{nk+j} \tag{21}$$

$$\theta^s_{nk+j} = v_{nk} + \sum_{i=0}^{j-1} \Delta v^s_{nk+i} - \gamma g^s_{nk+j} \tag{22}$$

$$\Delta w^s_{nk+j} = \mathbb{S}_{\lambda\gamma}(\theta^s_{nk+j}) - (w_{nk} + \sum_{i=1}^{j-1} \Delta w^s_{nk+i}) \tag{23}$$

where $\Delta w^s_n = \mathbb{S}_{\lambda\gamma}(\theta^s_n) - w^s_{n-1}$ and $\Delta w^1_n = \Delta w_n$. Using the same Hessian for multiple iterations could be seen as sampling the same data points from matrices X and y. This is doable as long as inner loop parameter S is small compared to the total iterations N. Otherwise, the subproblem is over-solved and executes redundant flops which could increase the overall runtime of the algorithm. The applied Hessian-reuse method is shown with blue in Algorithm 5. The inner loop at Line 9 updates the local variables for S iterations redundantly on all processors. The parameter S is tuned to find an efficient trade-off between computation and communication.

As discussed, with iteration-overlapping, RC-SFISTA reduces the latency costs by a factor of $O(k)$ without increasing bandwidth costs. With the Hessian-reuse method, RC-SFISTA increases the algorithm's computation complexity to further reduce communication rounds. Thus, the parameter S should be tuned to find an efficient trade-off between computation complexity and data communication. The algorithm's cost is shown in Table 1.

3.3 Extension to proximal Newton methods

RC-SFISTA could be used both as an inner solver inside PN methods or as an independent solver for the l1-regularized least squares problem. RC-SFISTA approximates the Hessian with random sampling which follows the same logic in line 3 of Algorithm 1. Then both methods minimize a quadratic subproblem that follows (19). Therefore, RC-SFISTA could also be used as an inner solver for PN methods. In this case, the iteration-overlapping approach reduces latency costs by a factor of $O(k)$ and if used independently it benefits from both iteration-overlapping and Hessian-reuse.

Algorithm 5: RC-SFISTA for the l1-regularized least squares problem

1 **Input:** $X \in \mathbb{R}^{d\times m}$, $y \in \mathbb{R}^m$, $w_0 = w_{-1} = \mathbf{0} \in \mathbb{R}^d$, $k \in \mathbb{N}$, $b \in (0, 1]$, $t_0 = 1$, $\bar{m} = \lfloor bm \rfloor$,$\gamma$
2 **for** $n = 0, \ldots, \frac{N}{k}$ **do**
3 **for** $j = 1, \ldots, k$ **do**
4 Generate $I_{nk+j} = [e_{i_1}, e_{i_2}, \ldots, e_{i_{\bar{m}}}] \in \mathbb{R}^{m\times\bar{m}}$ where $\{i_h \in [m] | h = 1, \ldots, \bar{m}\}$ is chosen uniformly at random
5 $H_{nk+j} = \frac{1}{\bar{m}} X I_{nk+j} I^T_{nk+j} X^T$, $R_{nk+j} = \frac{1}{\bar{m}} X I_{nk+j} I^T_{nk+j} y$
6 set $G = [H_{nk+1} | H_{nk+2} | \ldots | H_{(n+1)k}]$ and $R = [R_{nk+1} | R_{nk+2} | \ldots | R_{(n+1)k}]$ and send them to all processors.
7 **for** $j = 1, \ldots, k$ **do**
8 H_{nk+j} and R_{nk+j} are $d \times d$ and $d \times 1$ blocks of G and R respectively
9 **for** $s = 1, \ldots, S$ **do**
10 update Δv^s_{nk+j-1} based on (20)
11 update g^s_{nk+j} based on (21)
12 update θ^s_{nk+j} based on (22)
13 update Δw^s_{nk+j} based on (23)
14 $w^s_{nk+j} = \Delta w^s_{nk+j} + w^{s-1}_{nk+j}$
15 $w_{nk+j} = w^S_{nk+j}$
16 **output** w_N

4 IMPLEMENTATION ON DISTRIBUTED ARCHITECTURES

This section presents the implementation of RC-SFISTA on a distributed architecture. The theoretical upper bounds for parameters related to iteration overlapping and the Hessian-reuse implementations are also provided based on machine specification.

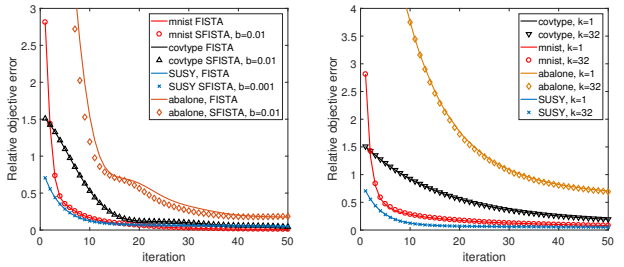
4.1 Distributed implementation

An overview of implementing RC-SFISTA on distributed architectures is shown in Figure 1. We assume that the data matrix X is sparse with "fdm" non-zeros that are uniformly distributed, where $0 < f < 1$ represents the percentage of non-zero fill-in in the data matrix X. Also, X and y are partitioned column- and row-wise respectively on P processors. In *stage A*, every processor randomly samples columns from X and the corresponding rows from y for k iterations. Using the sampled data, k instances of matrices H and R are computed locally on each processor in *stage B*. Processor

contributions are combined using MPI_Allreduce during *stage C*. Since the result from MPI_Allreduce is stored on all processors, RC-SFISTA will compute k iterations of solution updates without any communication between processors and each processor solves k subproblems for S iterations at *stage D*. This process is repeated until a stopping criterion is met.

4.2 RC-SFISTA parameter bounds

The iteration-overlapping parameter k significantly improves the performance of RC-SFISTA if latency costs dominate the overall runtime of the algorithm. Also, the inner loop parameter S reduces the total number of iterations in RC-SFISTA at the cost of more flops, often leading to better performance. In this section, the bounds for parameters k and S are derived based on algorithm and machine specifications.



(a) The effect of sampling rate. **(b) The effect of parameter k.**

Figure 2: Convergence of RC-SFISTA for different b and k.

RC-SFISTA runtime. We use the model in (7) to analyze the performance of RC-SFISTA. The total number of flops for RC-SFISTA is dominated by the matrix-matrix multiplication in line 5 and the Hessian-reuse in lines 10-14 in Algorithm 5. Also, RC-SFISTA only communicates a matrix of size d^2 every k iterations in line 6 requiring $O(d^2k)$ words to be moved between processors with $\frac{N}{k}$ messages. Thus, the total runtime of Algorithm 5 is:

$$T = \gamma\left(\frac{Nd^2\bar{m}f}{P} + Sd^2\right) + \alpha\left(\frac{Nlog(P)}{k}\right) + \beta\left(Nd^2log(P)\right) \quad (24)$$

The iteration-overlapping parameter k. The theoretical upper bound for k depends on machine specifications and dataset dimensions. Since k only appears in latency costs in (24), increasing k will always lead to a lower running time. However, the performance obtained from iteration-overlapping is more significant if the latency cost dominates the total runtime. By considering latency and bandwidth, the following upper bound is achieved:

$$k \leq \frac{\alpha}{\beta d^2} \quad (25)$$

This upper bound shows that RC-SFISTA leads to better performance on distributed platforms with a higher rate of latency to bandwidth ratio ($\frac{\alpha}{\beta}$). Comparing the first two terms in (24), i.e. flops and latency costs, we have:

$$k \leq \frac{\alpha NPlog(P)}{\gamma[Nd^2\bar{m}f + Sd^2P]}. \quad (26)$$

Equation 26 shows that for matrices with a lower sparsity degree f, k can be larger. In particular, if the dataset is very sparse ($f \sim 0$) we can write:

$$kS \leq \frac{\alpha Nlog(P)}{\gamma d^2}. \quad (27)$$

The upper bound in (27) provides a trade-off between the computational complexity and communication cost of RC-SFISTA where increasing the value of the iteration-overlapping parameter k results in a tighter bound for S.

The inner loop parameter S. Equation (27) shows that for higher values of k, a lower value for S is expected. Specifically, if the upper bound (25) is used, then:

$$S \leq \frac{\beta Nlog(P)}{\gamma}. \quad (28)$$

Equation (28) states that the upper bound for S depends on machine-specific parameters β and γ. Therefore, RC-SFISTA achieves a better performance on distributed architectures with a higher ratio of $\frac{\beta}{\gamma}$.

Dataset	Row numbers	Column numbers	Percentage of nnz (f)	Size (nnz)
abalone	4177	8	100%	258.7KB
SUSY	5M	18	25.39%	2.47GB
covtype	581,012	54	22.12%	71.2MB
mnist	60,000	780	19.22%	114.8MB
epsilon	400,000	2000	100%	12.16GB

Table 2: The datasets for experimental study.

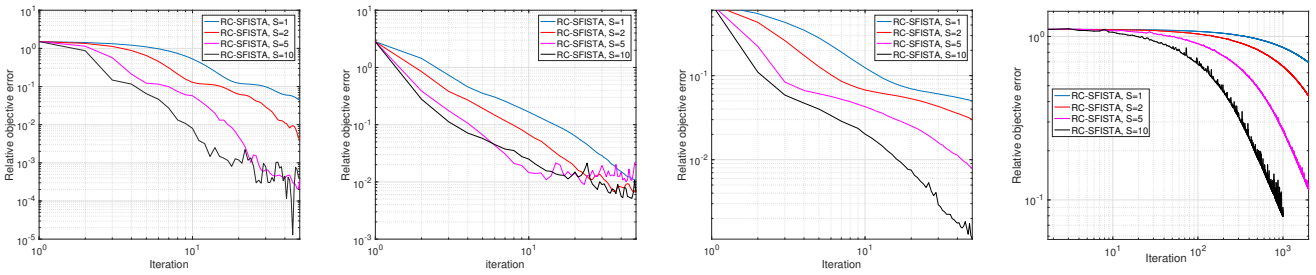
5 RESULTS

This section presents the experimental setup and performance results. We show that k does not change the convergence behavior of SFISTA. Experimental results are provided that show the inner loop parameter S improves the convergence of RC-SFISTA. Afterwards, speedup results of the proposed method as an independent solver and inner solver for PN methods is discussed. Finally, we demonstrate that our method performs better than the state-of-the-art framework ProxCoCoA up to 12×.

5.1 Experimental setup

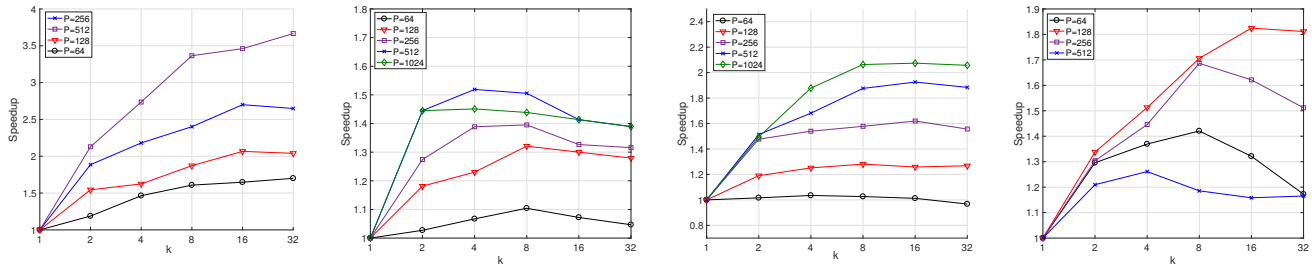
Table 2 shows the datasets used for our experiments [9]. The datasets are from dense and sparse machine learning applications and vary in size and sparsity [9] . RC-SFISTA is implemented in C/C++ using Intel MKL 11.1 for (sparse/dense) BLAS routines and MPI 2.1 for parallel processing. We use the compressed sparse row format to store the data of sparse datasets; abalone is stored as dense. Our experiments are conducted on the XSEDE Comet CPU nodes [32]. Because of resource constraints on Comet, for experiments with less than 64 nodes we use one processor per node, while for larger runs multiple processors per node were used. For example, to execute RC-SFISTA on 256 processors, we use 64 nodes and 4 processors per node.

Regularization parameter λ. The parameter λ should be chosen based on the prediction accuracy of the dataset and can affect convergence rates. We tune λ so that our experiments have reasonable



(a) covtype (b) mnist (c) SUSY (d) epsilon

Figure 3: Convergence of RC-SFISTA for different values of inner loop parameter *S*.



(a) covtype (b) mnist (c) SUSY (d) epsilon

Figure 4: Speedup results for RC-SFISTA compared to SFISTA for different values of the iteration-overlapping parameter *k*.

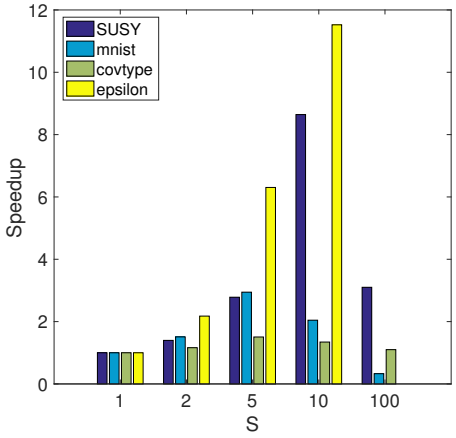


Figure 5: Speedup of RC-SFISTA vs. SFISTA for different *S*.

running time. The final tuned value for λ is 0.0001 for *epsilon* and 0.1 for all other benchmarks.

Stopping criteria. The relative objective error e_n is used as the stopping criteria where $e_n = |\frac{F(w_n)-F(w^*)}{F(w^*)}|$. Algorithm 5 returns when the relative objective error achieves a value less than a user-specified tolerance *tol* which in this paper is chosen to provide a reasonable execution time. The optimal solution, w^* is computed using *Templates for First-Order Conic Solvers* (TFOCS) which is competitive with state-of-the-art methods [3]. TFOCS uses a first order method where the tolerance for its stopping criteria is 10^{-8}.

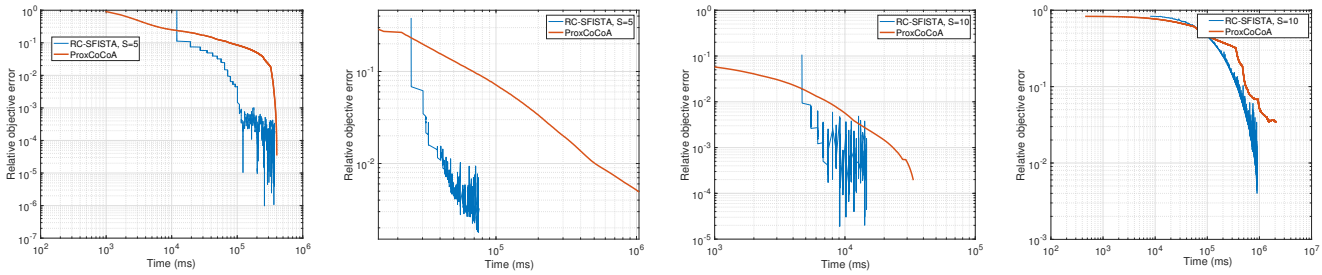
5.2 Convergence results

This section shows the effect of the sampling rate b on the convergence of SFISTA where the algorithm's computation cost is significantly reduced with smaller values of b. We also demonstrate the effect of parameters k and S on convergence rate and show that RC-SFISTA is numerically stable.

The effect of b on convergence. The relative objective error for RC-SFISTA for different values of sampling rate b is shown in Figure 2 (a) while setting k and S to 1. The convergence rates are almost identical compared to FISTA. Smaller values for b result in a smaller mini-batch size $\bar{m}$ and therefore a lower computation cost.

The effect of k on convergence. The iteration-overlapping parameter k does not change the convergence of RC-SFISTA since it is the same as SFISTA in exact arithmetic. For fair comparison, random sampling is fixed by using the same random generator seed, thus, in both scenarios the same data points are used in the algorithm. The convergence properties of RC-SFISTA for different values of k is shown in Figure 2 (b). The experiments demonstrate that changing k does not affect the stability and relative objective error. We tested the convergence rate and stability behavior of the algorithm for up to $k = 128$ and a similar trend was observed.

The effect of S on convergence. The convergence behavior of RC-SFISTA for different values of S is shown in Figure 3. Increasing S reduces the total number of iterations in RC-SFISTA to reach the optimal solution. As seen in the figure, even for small values of S,



(a) covtype (b) mnist (c) SUSY (d) epsilon

Figure 6: Relative objective error of RC-SFISTA compared to ProxCoCoA on 256 processors.

the improvement in convergence is noticeable. However, according to (12), larger values of S degrade the convergence behavior of RC-SFISTA which can also been seen in the figure for S equal to 10 for all benchmarks.

5.3 Speedup comparison

This section shows the speedup for RC-SFISTA compared to SFISTA for different values of k and S. The tolerance parameter *tol* is set to 0.01 in all the experiments. We show that increasing k reduces latency costs by a factor of k and improves performance of RC-SFISTA on a distributed architecture. We also provide speedup results for the inner loop parameter S and show the trade-off between the computation cost of RC-SFISTA and data communication.

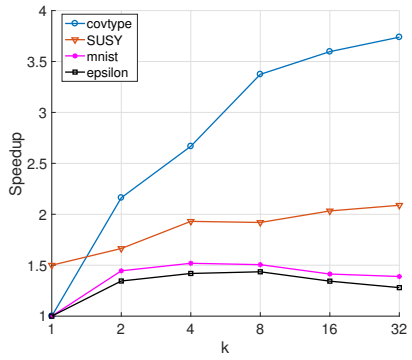


Figure 7: Speedup results for PN method with RC-SFISTA as inner solver compared to FISTA used as inner solver.

The effect of parameter k on speedup. The speedup of RC-SFISTA compared to SFISTA for different number of processors (P) and different values of k is shown in Figure 4. Parameter S is set to one to only analyze the effect of k on the total runtime of the algorithm. As shown in the figure, increasing k results in up to 4× speedup for all datasets by reducing latency costs by a factor of k. However, for larger values of k the performance of RC-SFISTA degrades for the dataset *epsilon* since the computation cost dominates the overall running time of the algorithm.

The value of k depends on machine specifications and the dataset size. For example, the machine parameters α and β for the XSEDE Comet nodes used in the experiments are 10^{-6} and 1.42×10^{-10}, thus, the theoretical upper bound (25) for the *covtype* dataset is 2. However, according to (24) all values of k reduce the total runtime of the algorithm, thus, RC-SFISTA continues to scale even for larger values of k.

The effect of S on speedup. The speedup results of RC-SFISTA compared to SFISTA on 256 processors for different values of S are shown in Figure 5. The value of parameter k is tuned for all benchmarks. Increasing S will result in better convergence properties by reducing the total number of iterations. S is increased until an efficient trade-off between computation cost and data communication is reached. As shown in Figure 5, for larger values of S, the cost of redundant computation overwhelms the total cost of the algorithm and the speedup decreases. For example, RC-SFISTA shows a speedup of 3× compared to SFISTA for the *mnist* dataset when $S = 5$, while it achieves a speedup of 2× when the inner loop parameter increases to 10. The upper bound for parameter S depends on both the architecture and the algorithm iterations N. Since on XSEDE Comet nodes the value of γ is 4×10^{-10}, based on the upper bound in (27) with values $k = 1$, $P = 256$, and $N = 200$ for the *mnist* dataset we have $S < 7$. As shown in Figure 5, $S = 5$ gives the best speedup for *mnist* on 256 processors.

5.4 Comparison to ProxCoCoA

In this section, we compare RC-SFISTA with ProxCoCoA, a state-of-the-are framework for solving large-scale l1-regularized least squares problems. Since ProxCoCoA is optimized and implemented in Apache Spark's MLlib we also implemented RC-SFISTA using Apache Spark MLlib[25]. When analyzing the performance of the algorithms, we measure the relative objective error in terms of wall-clock time. For all the experiments, the value of S is tuned for best performance. The results with 256 workers on 256 processors are shown in Figure 6. ProxCoCoA has a slow convergence for all datasets, however, RC-SFISTA converges faster and reaches a lower relative objective error compared to ProxCoCoA. Table 3 summarizes the speedup of RC-SFISTA compared to ProxCoCoA on 256 workers. The parameter *tol* is set to 0.01 for all benchmarks and the sampling rate b is set to 1% for RC-SFISTA.

Dataset	SUSY	covtype	mnist	epsilon
Speedup	1.57×	4.74×	12.15×	3.53×

Table 3: Speedup of RC-SFISTA compared to ProxCoCoA.

5.5 Speedup results for PN methods

This section shows the results for RC-SFISTA used as an inner solver in PN methods (Algorithm 1). The speedups are normalized over the PN method with FISTA as an inner solver. The Hessian approximation for both algorithms is obtained using uniform sampling by initializing all processors with the same seed for the random number generator. The parameter S for RC-SFISTA and the number of inner solver iterations for PN methods are tuned for best performance. The speedup results on 512 processors are shown in Figure 7. As demonstrated, as long as the latency cost dominates the communication cost, increasing k results in a better speedup.

6 CONCLUSION

The performance of proximal Newton methods used for solving l1-regularized least squares problems is limited by the performance of the inner solver used in these algorithms. PN methods do not scale well on distributed platforms when operating on large datasets. We propose a novel inner solver, RC-SFISTA, that leverages randomized sampling to overlap iterations and reduce latency costs by a factor of k. The RC-SFSITA algorithm, keeps the convergence behavior and preserves the overall bandwidth cost. The performance of the inner solver is further improved by solving local subproblems on each processor at cost of more floating point operations. Our experiments show that RC-SFISTA provides up to 12× speedup compared to SFISTA and the state-of-the-art method ProxCoCoA for the tested datasets on distributed platforms.

7 ACKNOWLEDGEMENTS

This work is supported by the U.S. National Science Foundation (NSF) Award Numbers CCF-1657175 and DMS-1723085, DOE AC02-05CH11231, and Cray. The work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation grant number ACI-1548562.

REFERENCES

[1] Grey Ballard, E Carson, J Demmel, M Hoemmen, Nicholas Knight, and Oded Schwartz. 2014. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numerica* 23 (2014), 1–155.

[2] Amir Beck and Marc Teboulle. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences* 2, 1 (2009), 183–202.

[3] Stephen R Becker, Emmanuel J Candès, and Michael C Grant. 2011. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical programming computation* 3, 3 (2011), 165–218.

[4] Dimitri P Bertsekas. 2011. Incremental proximal methods for large scale convex optimization. *Mathematical programming* 129, 2 (2011), 163.

[5] Léon Bottou, Frank E Curtis, and Jorge Nocedal. 2016. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838* (2016).

[6] Erin Carson, Nicholas Knight, and James Demmel. 2013. Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods. *SIAM Journal on Scientific Computing* 35, 5 (2013), S42–S61.

[7] Erin Claire Carson. 2015. *Communication-avoiding Krylov subspace methods in theory and practice.* University of California, Berkeley.

[8] Antonin Chambolle and Thomas Pock. 2011. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision* 40, 1 (2011), 120–145.

[9] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)* 2, 3 (2011), 27.

[10] A.T. Chronopoulos and C.D. Swanson. 1996. Parallel iterative S-step methods for unsymmetric linear systems. *Parallel Comput.* 22, 5 (1996), 623 – 641. DOI: http://dx.doi.org/https://doi.org/10.1016/0167-8191(96)00022-1

[11] Ingrid Daubechies, Michel Defrise, and Christine De Mol. 2004. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on pure and applied mathematics* 57, 11 (2004), 1413–1457.

[12] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. 2012. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM Journal on Scientific Computing* 34, 1 (2012), A206–A239.

[13] Aditya Devarakonda, Kimon Fountoulakis, James Demmel, and Michael W Mahoney. 2016. Avoiding communication in primal and dual block coordinate descent methods. *arXiv preprint arXiv:1612.04003* (2016).

[14] Aditya Devarakonda, Kimon Fountoulakis, James Demmel, and Michael W Mahoney. 2017. Avoiding Synchronization in First-Order Methods for Sparse Convex Optimization. *arXiv preprint arXiv:1712.06047* (2017).

[15] Jack Dongarra, Jeffrey Hittinger, John Bell, Luis Chacon, Robert Falgout, Michael Heroux, Paul Hovland, Esmond Ng, Clayton Webster, and Stefan Wild. 2014. *Applied mathematics research for exascale computing.* Technical Report. Lawrence Livermore National Laboratory (LLNL), Livermore, CA.

[16] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. 2010. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software* 33, 1 (2010), 1.

[17] Samuel H Fuller and Lynette I Millett. 2011. Computing performance: Game over or next level? *Computer* 44, 1 (2011), 31–38.

[18] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. 2015. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*. 797–842.

[19] Mingyi Hong and Tsung-Hui Chang. 2017. Stochastic proximal gradient consensus over random networks. *IEEE Transactions on Signal Processing* 65, 11 (2017), 2933–2948.

[20] Tyler Johnson and Carlos Guestrin. 2015. Blitz: A principled meta-algorithm for scaling sparse optimization. In *International Conference on Machine Learning*. 1171–1179.

[21] Jakub Konečnỳ, Jie Liu, Peter Richtárik, and Martin Takáč. 2016. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing* 10, 2 (2016), 242–255.

[22] Jason D Lee, Yuekai Sun, and Michael A Saunders. 2014. Proximal Newton-type methods for minimizing composite functions. *SIAM Journal on Optimization* 24, 3 (2014), 1420–1443.

[23] Dhruv Mahajan, S Sathiya Keerthi, and S Sundararajan. 2017. A distributed block coordinate descent method for training l 1 regularized linear classifiers. *Journal of Machine Learning Research* 18, 91 (2017), 1–35.

[24] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. 2009. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 689–696.

[25] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, and others. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[26] Atsushi Nitanda. 2014. Stochastic Proximal Gradient Descent with Acceleration Techniques. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1574–1582. http://papers.nips.cc/paper/5610-stochastic-proximal-gradient-descent-with-acceleration-techniques.pdf

[27] Atsushi Nitanda. 2014. Stochastic proximal gradient descent with acceleration techniques. In *Advances in Neural Information Processing Systems*. 1574–1582.

[28] Neal Parikh, Stephen Boyd, and others. 2014. Proximal algorithms. *Foundations and Trends® in Optimization* 1, 3 (2014), 127–239.

[29] Mark Schmidt, Alexandru Niculescu-Mizil, Kevin Murphy, and others. 2007. Learning graphical model structure using L1-regularization paths. In *AAAI*, Vol. 7. 1278–1283.

[30] Virginia Smith, Simone Forte, Michael I Jordan, and Martin Jaggi. 2015. L1-regularized distributed optimization: A communication-efficient primal-dual framework. *arXiv preprint arXiv:1512.04011* (2015).

[31] Virginia Smith, Simone Forte, Chenxin Ma, Martin Takac, Michael I Jordan, and Martin Jaggi. 2016. CoCoA: A General Framework for Communication-Efficient Distributed Optimization. *arXiv preprint arXiv:1611.02189* (2016).

[32] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, and others. 2014. XSEDE: accelerating scientific discovery. *Computing in Science & Engineering* 16, 5 (2014), 62–74.

[33] Tong Tong Wu, Kenneth Lange, and others. 2008. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics* 2, 1 (2008), 224–244.

[34] Lin Xiao and Tong Zhang. 2014. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization* 24, 4 (2014), 2057–2075.

[35] Zeyuan Allen Zhu, Weizhu Chen, Gang Wang, Chenguang Zhu, and Zheng Chen. 2009. P-packSVM: Parallel Primal grAdient desCent Kernel SVM. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining (ICDM '09)*. IEEE Computer Society, Washington, DC, USA, 677–686. DOI: http://dx.doi.org/10.1109/ICDM.2009.29