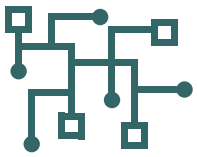

ISE

Industrial and Systems Engineering

Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting

Jakub Konečný[1], Jie Liu[2], Peter Richtárik[1], and Martin Takáč[2]

[1]School of Mathematics, The University of Edinburgh

[2]Department of Industrial and Systems Engineering, Lehigh University, USA

ISE Technical Report 15T-003

LEHIGH UNIVERSITY

Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting

Jakub Konečný, Jie Liu, Peter Richtárik, Martin Takáč

***Abstract*—We propose mS2GD: a method incorporating a mini-batching scheme for improving the theoretical complexity and practical performance of semi-stochastic gradient descent (S2GD). We consider the problem of minimizing a strongly convex function represented as the sum of an average of a large number of smooth convex functions, and a simple nonsmooth convex regularizer. Our method first performs a deterministic step (computation of the gradient of the objective function at the starting point), followed by a large number of stochastic steps. The process is repeated a few times with the last iterate becoming the new starting point. The novelty of our method is in introduction of mini-batching into the computation of stochastic steps. In each step, instead of choosing a single function, we sample b functions, compute their gradients, and compute the direction based on this. We analyze the complexity of the method and show that it benefits from two speedup effects. First, we prove that as long as b is below a certain threshold, we can reach any predefined accuracy with less overall work than without mini-batching. Second, our mini-batching scheme admits a simple parallel implementation, and hence is suitable for further acceleration by parallelization.**

***Index Terms*—mini-batches, proximal methods, empirical risk minimization, semi-stochastic gradient descent, sparse data, stochastic gradient descent, variance reduction.**

I. Introduction

THE problem we are interested in is to minimize a sum of two convex functions,

$$\min_{x\in\mathbb{R}^d}\{P(x) := F(x) + R(x)\}, \tag{1}$$

where R is a separable, convex, possibly nonsmooth function and F is the average of a large number of smooth convex functions f_i, i.e.,

$$F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x). \tag{2}$$

We further make the following assumptions:

Assumption 1. *The regularizer $R : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ is convex and closed. The functions $f_i : \mathbb{R}^d \to \mathbb{R}$ are differentiable and have Lipschitz continuous gradients with constant $L > 0$. That is,*

$$\|\nabla f_i(x) - \nabla f_i(y)\| \le L\|x - y\|,$$

for all $x, y \in \mathbb{R}^d$, where $\|\cdot\|$ is L2 norm.

Jakub Konečný and Peter Richtárik are with the School of Mathematics, University of Edinburgh, United Kingdom, EH9 3FD.

Jie Liu and Martin Takáč are with the Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, USA.

Manuscript received April 15, 2015; revised —.

Hence, the gradient of F is also Lipschitz continuous with the same constant L.

Assumption 2. *P is strongly convex with parameter $\mu > 0$, i.e., $\forall x, y \in \mathrm{dom}(R)$,*

$$P(y) \ge P(x) + \xi^T(y - x) + \frac{\mu}{2}\|y - x\|^2, \quad \forall \xi \in \partial P(x), \tag{3}$$

where $\partial P(x)$ is the subdifferential of P at x.

Let $\mu_F \ge 0$ and $\mu_R \ge 0$ be the strong convexity parameters of F and R, respectively (we allow both of them to be equal to 0, so this is not an additional assumption). We assume that we have lower bounds available ($\nu_F \in [0, \mu_F]$ and $\nu_R \in [0, \mu_R]$).

A. Our contributions

In this work, we combine the variance reduction ideas for stochastic gradient methods with mini-batching. In particular, we develop and analyze he complexity of mS2GD (Algorithm 1) – a mini-batch proximal variant of S2GD [1]. We show that the method enjoys twofold benefit compared to previous methods. Apart from admitting a parallel implementation (and hence speedup in clocktime in an HPC environment), our results show that in order attain a specified accuracy ϵ, our mini-batching scheme can get by with fewer gradient evaluations. This is formalized in Theorem 2, which predicts more than linear speedup up to some b — the size of the mini-batches. Another advantage, compared to [2], is that we do not need to average the t_k points x in each loop, but we instead simply continue from the last one (this is the approach employed in S2GD [1]).

B. Related work

There has been intensive interest and activity in solving problems of the structure (1) in the past years. One of the most practical methods for this problem is accelerated proximal gradient descent of Nesterov, with its most successful variant being FISTA [3]. However, this method is not very efficient in large-scale settings (big n) as it needs to process all n functions in each iteration. Two classes of methods address this issue – randomized coordinate descent methods [4], [5], [6], [7], [8], [9], [10], [11], [12], [13] and stochastic gradient methods [14], [15], [16], [17], [18]. This paper is closely related to the works on stochastic gradient methods with a technique of explicit variance reduction of stochastic approximation of the gradient. In particular, our method is a mini-batch variant of S2GD [1]; the proximal setting was motivated by SVRG [19], [2]. Moreover, an accelerated version of proximal SVRG

with mini-batches has been proposed by Nitanda as Acc-Prox-SVRG [20]; however, the acceleration of Acc-Prox-SVRG depends largely on the mini-batch size.

A typical stochastic gradient descent (SGD) method will randomly sample i^{th} function and then update the variable x using $\nabla f_i(x)$ — a stochastic estimate of $\nabla F(x)$. Important limitation of SGD is that it is inherently sequential. In order to enable parallelism, the idea of mini-batching—using estimating the gradient using multiple random i in each iteration—is often employed [21], [22], [23], [24], [18], [25].

II. Proximal Algorithms

A popular *proximal gradient* approach to solving (1) is to form a sequence $\{y_k\}$ via

$$y_{t+1} = \arg\min_{x\in\mathbb{R}^d} \Big[U_k(x) \stackrel{\text{def}}{=} F(y_t) + \nabla F(y_t)^T(x-y_t) + \frac{1}{2h}\|x-y_t\|^2 + R(x)\Big].$$

Note that in view of Assumption 1, U_t is an upper bound on P whenever $h>0$ is a stepsize parameter satisfying $1/h \geq L$. This procedure can be equivalently written using the *proximal operator* as follows:

$$y_{t+1} = \text{prox}_{hR}(y_t - h\nabla F(y_t)),$$

where

$$\text{prox}_R(z) \stackrel{\text{def}}{=} \arg\min_{x\in\mathbb{R}^d}\{\tfrac{1}{2}\|x-z\|^2 + R(x)\}.$$

In a large-scale setting it is more efficient to instead consider the *stochastic proximal gradient* approach, in which the proximal operator is applied to a stochastic gradient step:

$$y_{t+1} = \text{prox}_{hR}(y_t - hG_t), \tag{4}$$

where G_t is a stochastic estimate of the gradient $\nabla f(y_t)$. Of particular relevance to our work are the the SVRG [19], S2GD [1] and Prox-SVRG [2] methods where the stochastic estimate of $\nabla F(y_t)$ is of the form

$$G_t = \nabla F(x) + \tfrac{1}{nq_{i_t}}(\nabla f_{i_t}(y_t) - \nabla f_{i_t}(x)), \tag{5}$$

where x is an "old" reference point for which the gradient $\nabla f(x)$ was already computed in the past, and $i_t \in [n] \stackrel{\text{def}}{=} \{1,2,\ldots,n\}$ is a random index equal to i with probability $q_i > 0$. Notice that G_t is an unbiased estimate of the gradient of F at y_t:

$$\mathbf{E}_i[G_t] \stackrel{(5)}{=} \nabla F(x) + \sum_{i=1}^n q_i \tfrac{1}{nq_i}(\nabla f_i(y_t) - \nabla f_i(x)) \stackrel{(2)}{=} \nabla F(y_t).$$

Methods such as S2GD [1], SVRG [19] and Prox-SVRG [2] update the points y_t in an inner loop, and the reference point x in an outer loop indexed by k. With this new out iteration counter, we will have x_k instead of x, $y_{k,t}$ instead of y_t and $G_{k,t}$ instead of G_t. This is the notation we will use in the description of our algorithm in the next section. The outer loop ensures that the squared norm of $G_{k,t}$ approaches zero as $k,t\to\infty$ (it is easy to see that this is equivalent to saying that the stochastic estimate $G_{k,t}$ has a diminishing variance), which ultimately leads to extremely fast convergence. Indeed, semi-stochastic methods enjoy the complexity bound $O((n+\kappa)\log(1/\epsilon))$, where $\kappa = L/\mu$ is a condition number. This should be contrasted with proximal gradient descent, with complexity $O(n\kappa\log(1/\epsilon))$ or FISTA, with complexity $O(\sqrt{n\kappa}\log(1/\epsilon))$. It is clear that semi-stochastic methods always beat gradient descent, and even outperform FISTA in specific regimes for κ and n. However, unlike FISTA, this is achieved *without* the use of momentum.

III. Mini-batch S2GD

We now describe the mS2GD method (Algorithm 1).

Algorithm 1 mS2GD

1: **Input:** m (max # of stochastic steps per epoch); $h>0$ (stepsize); $x_0\in\mathbb{R}^d$ (starting point); mini-batch size $b\in[n]$
2: **for** $k=0,1,2,\ldots$ **do**
3: Compute and store $g_k \leftarrow \nabla F(x_k) = \frac{1}{n}\sum_i \nabla f_i(x_k)$
4: Initialize the inner loop: $y_{k,0}\leftarrow x_k$
5: Let $t_k \leftarrow t \in \{1,2,\ldots,m\}$ with probability q_t given by (6)
6: **for** $t=0$ to t_k-1 **do**
7: Choose mini-batch $A_{kt}\subset[n]$ of size b, uniformly at random
8: Compute a stochastic estimate of $\nabla F(y_{k,t})$: $G_{k,t} \leftarrow g_k + \frac{1}{b}\sum_{i\in A_{kt}}(\nabla f_i(y_{k,t}) - \nabla f_i(x_k))$
9: $y_{k,t+1}\leftarrow \text{prox}_{hR}(y_{k,t} - hG_{k,t})$
10: **end for**
11: Set $x_{k+1}\leftarrow y_{k,t_k}$
12: **end for**

The algorithm includes an outer loop, indexed by epoch counter k, and an inner loop, indexed by t. Each epoch is started by computing g_k, which is the (full) gradient of F at x_k. It then immediately proceeds to the inner loop. The inner loop is run for t_k iterations, where t_k is chosen randomly by setting $t_k = t \in \{1,2,\ldots,m\}$ with probability q_t, where

$$q_t \stackrel{\text{def}}{=} \frac{1}{\gamma}\left(\tfrac{1-h\nu_F}{1+h\nu_R}\right)^{m-t}, \quad \text{where} \quad \gamma \stackrel{\text{def}}{=} \sum_{t=1}^m \left(\tfrac{1-h\nu_F}{1+h\nu_R}\right)^{m-t}. \tag{6}$$

Subsequently, we run t_k iterations in the inner loop — the main step of our method (Step 8). Each new iterate is given by the proximal update (4), however with the stochastic estimate of the gradient $G_{k,t}$ in (5), which is formed by using a *mini-batch* of examples $A_{kt}\subset[n]$ of size $|A_{kt}| = b$. Each inner iteration takes *2b component gradient evaluations*[1].

IV. Complexity Result

In this section, we state our main complexity result and comment on how to optimally choose the parameters of the method.

[1]It is possible to finish each iteration with only b evaluations for component gradients, namely $\{\nabla f_i(y_{k,t})\}_{i\in A_{kt}}$, at the cost of having to store $\{\nabla f_i(x_k)\}_{i\in[n]}$, which is exactly the way that SAG [26] works. This speeds up the algorithm; nevertheless, it is impractical for big n.

Theorem 1. *Let Assumptions 1 and 2 be satisfied and let* $x_* \overset{def}{=} \arg\min_x P(x)$. *In addition, assume that the stepsize satisfies* $0 < h \leq \min\left\{\frac{1-h\nu_F}{1+h\nu_R}\frac{1}{4L\alpha(b)}, \frac{1}{L}\right\}$ *and that* m *is sufficiently large so that*

$$\rho \overset{def}{=} \frac{\left(\frac{1-h\nu_F}{1+h\nu_R}\right)^m \frac{1}{\mu} + \frac{4h^2L\alpha(b)}{1+h\nu_R}\left(\gamma + \left(\frac{1-h\nu_F}{1+h\nu_R}\right)^{m-1}\right)}{\gamma h\left\{\frac{1}{1+h\nu_R} - \frac{4hL\alpha(b)}{1-h\nu_F}\right\}} < 1, \tag{7}$$

where $\alpha(b) = \frac{n-b}{b(n-1)}$. *Then mS2GD has linear convergence in expectation:*

$$\mathbf{E}(P(x_k) - P(x_*)) \leq \rho^k(P(x_0) - P(x_*)).$$

Remark 1. *If we consider the special case* $\nu_F = 0$, $\nu_R = 0$ *(i.e., the case that we do not have any good estimate for* μ_F *and* μ_R*), we obtain*

$$\rho = \frac{1}{mh\mu(1 - 4hL\alpha(b))} + \frac{4hL\alpha(b)\,(m+1)}{m(1 - 4hL\alpha(b))}. \tag{8}$$

In the special case when $b = 1$ *we get* $\alpha(b) = 1$, *and the rate given by* (8) *exactly recovers the rate achieved by Prox-SVRG [2] (in the case when the Lipschitz constants of* ∇f_i *are all equal).*

The rest of this section focuses on post-analysis of the above result. In particular, we show what the optimal choice of parameters is, and how it translates to the overall complexity result.

A. Mini-batch Speedup

To explore the speedup from applying mini-batch strategy, we need to fix some parameters to avoid too many parameters in the complexity result (Theorem 1). For simplicity, we are using $\nu_F = 0$ and $\nu_R = 0$ in (7) so that we can analyze (8), in which case we can still analyze a strongly convex problem even without any explicit knowledge of its modulus.

Moreover, due to the fact that mini-batching is only employed in inner loops, it would be reasonable for us to fix the target decrease ρ for each epoch. Consequently, to achieve ϵ-accuracy, which should guarantee

$$\mathbf{E}[P(x_k) - P(x_*)] \leq \epsilon[P(x_0) - P(x_*)], \tag{9}$$

the number of epochs should be at least $\lceil \log(1/\epsilon) \rceil$.

Now let us focus only on inner loops and fix target decrease as ρ_* in a single epoch. For any $1 \leq b \leq n$, define (h_*^b, m_*^b) to be the optimal pair in the sense that the stepsize $h = h_*^b$ minimizes the computational effort — m is minimized subject to $\rho \leq \rho_*$ with ρ defined in (8). Under these definitions, $b = 1$ recovers the optimal choice of parameters without mini-batching. If $m_*^b < m_*^1/b$ for some $b > 1$, then mini-batching can help us reach the target decrease ρ_* with fewer component gradient evaluations.

The following theorem presents the formulas of h_*^b and m_*^b. Equation (10) suggests that as long as the condition $\tilde{h}^b \leq \frac{1}{L}$ holds, m_*^b is decreasing at a rate roughly faster than $1/b$. Hence, we can attain the same decrease with less work, compared to the case when $b = 1$.

Theorem 2. *Fix target decrease* $\rho = \rho_*$, *where* ρ *is given by* (8) *and* $\rho_* \in (0, 1)$. *If we consider the mini-batch size* b *to be fixed and define the following quantity,*

$$\tilde{h}^b \overset{def}{=} \sqrt{\left(\frac{1+\rho}{\rho\mu}\right)^2 + \frac{1}{4\mu\alpha(b)L}} - \frac{1+\rho}{\rho\mu},$$

then the choice of stepsize h_*^b *and size of inner loops* m_*^b, *which minimizes the work done — the number of gradients evaluated — while having* $\rho \leq \rho_*$, *is given by the following statements.*

If $\tilde{h}^b \leq \frac{1}{L}$, *then* $h_*^b = \tilde{h}^b$ *and*

$$m_*^b = \frac{2\kappa}{\rho}\left\{\left(1 + \frac{1}{\rho}\right)4\alpha(b) + \sqrt{\frac{4\alpha(b)}{\kappa} + \left(1 + \frac{1}{\rho}\right)^2[4\alpha(b)]^2}\right\}, \tag{10}$$

where $\kappa \overset{def}{=} \frac{L}{\mu}$ *is the condition number; otherwise,* $h_*^b = \frac{1}{L}$ *and*

$$m_*^b = \frac{\kappa + 4\alpha(b)}{\rho - 4\alpha(b)(1+\rho)}. \tag{11}$$

B. Convergence Rate

In this section we provide a practical bound on number of component gradient evaluations $\nabla f_i(x)$ needed to achieve a predefined ϵ-accuracy in k iterations (9). In particular, we show that the efficient speedup from mini-batching is obtainable only for b roughly up to 29.

We set the number of outer iterations to be $k = \lceil \log(1/\epsilon) \rceil$. Fix the target decrease in Theorem 2 to satisfy $\rho \leq \epsilon^{1/k}$, which gives the corresponding optimal choice of parameters h^b and m^b, then this yields the total complexity of

$$\mathcal{O}\left((n + bm^b)\log(1/\epsilon)\right)$$

gradient evaluaitons to get (9).

From Theorem 2, the following equivalence holds,

$$\tilde{h}^b < \frac{1}{L} \quad \Longleftrightarrow \quad b < b_0 \overset{\text{def}}{=} \frac{8\rho n\kappa + 8n\kappa + 4\rho n}{\rho n\kappa + (7\rho + 8)\kappa + 4\rho}.$$

Hence, it follows that if $b < \lceil b_0 \rceil$, then $h^b = \tilde{h}^b$ and m^b is defined in (10); otherwise, $h^b = \frac{1}{L}$ and m^b is defined in (11). Obviously, with n large, we have $b_0 \geq 8$, which, together with the above two cases, is able to demonstrate a total complexity of

$$\mathcal{O}\left((n + b\kappa)\log(1/\epsilon)\right).$$

Denote e as the base of natural logarithm. By selecting $b_0 = \frac{8n\kappa + 8en\kappa + 4n}{n\kappa + (7+8e)\kappa + 4}$, choosing mini-batch size $b < \lceil b_0 \rceil$, and running the inner loop of mS2GD for

$$m_b = \left\lceil 8e\alpha(b)\kappa\left(e + 1 + \sqrt{\frac{1}{4\alpha(b)\kappa} + (1+e)^2}\right)\right\rceil \tag{12}$$

iterations with constant stepsize

$$h^b = \sqrt{\left(\frac{1+e}{\mu}\right)^2 + \frac{1}{4\mu\alpha(b)L}} - \frac{1+e}{\mu}, \tag{13}$$

we can achieve a convergence rate $\rho \leq e^{-1} < \epsilon^{1/k}$ with $k = \lceil \log(1/\epsilon) \rceil$. Therefore, running mS2GD for k outer iterations achieves ϵ-accuracy solution defined in (9). Moreover, the total complexity can be translated to

$$\mathcal{O}\left((n+\kappa)\log(1/\epsilon)\right).$$

This result shows that we can reach efficient speedup by mini-batching as long as the mini-batch size is smaller than some threshold $b_0 = \frac{8(1+e)n\kappa+4n}{n\kappa+(7+8e)\kappa+4}$. Since in general $\kappa \gg e, n \gg e$, it can be concluded that

$$b_0 \approx 8\,(e+1) \approx 29.75,$$

which proves the following corollary.

Corollary 3. *By setting the number of outer iterations*

$$k = \lceil \log(1/\epsilon) \rceil,$$

minibatch size to $1 \leq b \leq 29$*, and* h *as in* (13) *and* m *as in* (12)*, the total complexity of mS2GD is*

$$\mathcal{O}\left((n+\kappa)\log(1/\epsilon)\right).$$

The complexity is measured in terms of component gradient evaluations, with the goal to achieve target accuracy ϵ *in* (9)*.*

C. Comparison with Acc-Prox-SVRG

One of the most related method, which applies mini-batch scheme to stochastic gradient variance-reduced methods, is the Acc-Prox-SVRG [20]. Acc-Prox-SVRG incorporates both mini-batch scheme and Nesterov's acceleration method [27], [28]; however, the author claims that when $b < \lceil b_0 \rceil$, with the threshold b_0 defined as $\frac{8\sqrt{\kappa}n}{\sqrt{2}p(n-1)+8\sqrt{\kappa}}$, the overall complexity is

$$\mathcal{O}\left(\left(n + \frac{n-b}{n-1}\kappa\right)\log(1/\epsilon)\right);$$

otherwise, it is

$$\mathcal{O}\left(\left(n + b\sqrt{\kappa}\right)\log(1/\epsilon)\right).$$

This suggests that acceleration will only be realized when the mini-batch size is large, while for small b, Acc-Prox-SVRG achieves the same overall complexity of

$$\mathcal{O}\left((n+\kappa)\log(1/\epsilon)\right)$$

as mS2GD.

Taking a close look into the theoretical results given by Acc-Prox-SVRG and mS2GD, for each $\epsilon \in (0,1)$, we are numerically minimizing the total work done — total number of component gradient evaluations given by

$$\left(n + 2b\lceil m^b \rceil\right)\left\lceil \frac{\log(1/\epsilon)}{\log(1/\rho)} \right\rceil,$$

over $\rho \in (0,1)$ and h, to compare their complexity.[2]

Fig. 1 illustrates situations for both ill-conditioned and well-conditioned data in theory. With a small mini-batch size b, mS2GD is advantageous over Acc-Prox-SVRG; however, for a large b, the situation reverses because of the acceleration in Acc-Prox-SVRG.[3] Plots with $b = 64$ illustrate the cases where we cannot observe any differences in both methods.

V. Experiments

In this section we perform numerical experiments to illustrate the properties and performance of our algorithm. In Section V-A, we impose an efficient way to apply mS2GD algorithm to sparse datasets. In Section V-B, we introduce numerical characteristics of mS2GD. In the last section, we compare our mS2GD with some relevant algorithms.

Although our mS2GD in proximal setting applies to both the popular regularizers with L1 norm and L2 norm. We focus our experiments in Sections V-B and V-C on the problem with L2-regularizer, i.e.,

$$P(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) + \frac{\lambda}{2}\|x\|^2, \tag{14}$$

and in Sections V-B and V-C, we conduct experiments on logistic regression for which we have

$$f_i(x) = \log[1 + \exp(-b_i a_i^T x)] \tag{15}$$

in problem (14), with a set of training data points $\{(a_i, b_i)\}_{i=1}^n$, where $a_i \in \mathbb{R}^d$ and $b_i \in \{+1, -1\}$ for binary classification problems.

We performed experiments on four publicly available binary classification datasets, namely *rcv1, news20, covtype* [4] and *astro-ph* [5]. In Section V-B, we investigate speedup from mini-batches and parallelism in practice on the *rcv1* and *astro-ph* datasets. Section V-C, focuses on comparison of the performances of other relevant algorithms on all four datasets.

In the logistic regression problem, the Lipschitz constant of function f_i can be derived as $L_i = \|a_i\|^2/4$. Our analysis assumes (Assumption 1) the same constant L for all functions. Hence, we get the constant as $L = \max_{i\in[n]} L_i$. We set the regularization parameter $\lambda = \frac{1}{n}$ in our experiments, resulting in the problem having the condition number, evaluated as $\frac{L}{\mu} = \mathcal{O}(n)$, which is typically the most ill-conditioned problem considered in practice.

TABLE I gives a summary of the four datasets, including the sizes n, dimensions d, their sparsity as proportion of nonzero elements and Lipschitz constants L.

Dataset	n	d	Sparsity	L
rcv1	20,242	47,236	0.1568%	0.2500
news20	19,996	1,355,191	0.0336%	0.2500
covtype	581,012	54	22.1212%	1.9040
astro-ph	62,369	99,757	0.0767%	0.2500

TABLE I: Summary of datasets used for experiments.

[2] m^bs are the best choices of m claimed by Acc-Prox-SVRG and mS2GD, respectively; meanwhile, hs are within the safe upper bounds for Acc-Prox-SVRG and mS2GD, respectively.

[3] We have experimented with different values for n, b and κ, and this result always holds.

[4] *rcv1, covtype* and *news20* are available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

[5] Available at http://users.cecs.anu.edu.au/~xzhang/data/.

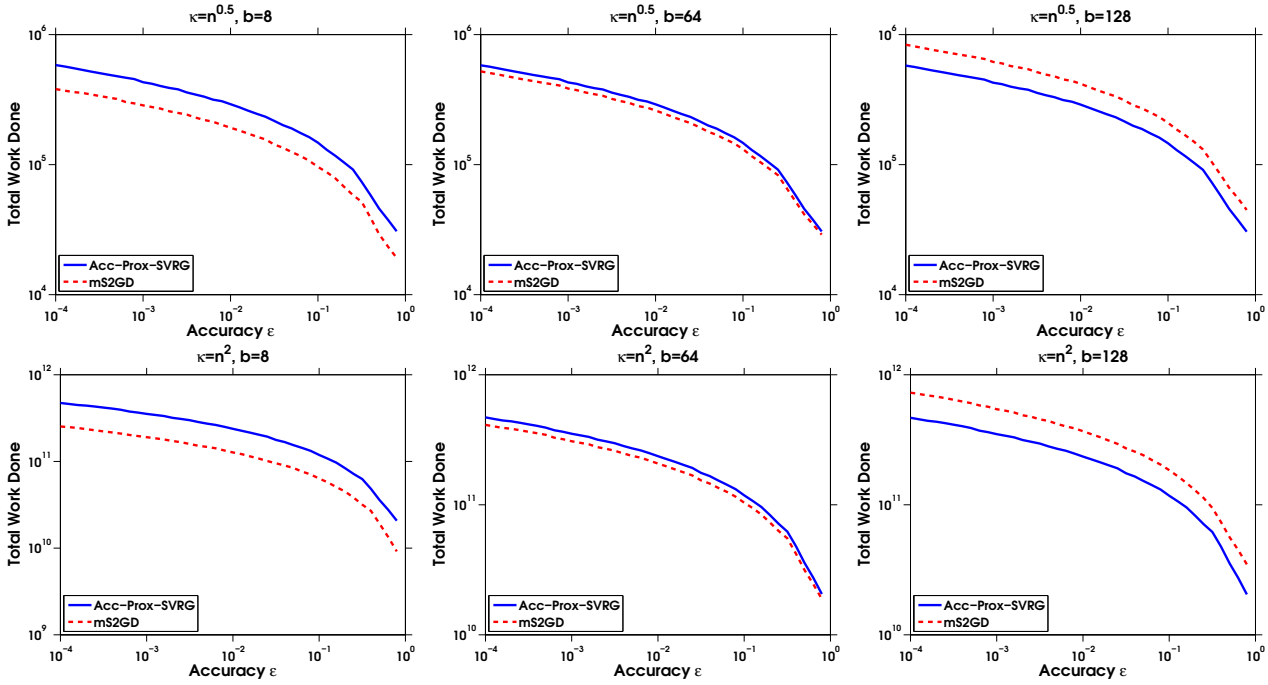

Fig. 1: Complexity of Acc-Prox-SVRG and mS2GD in terms of total work done: $n = 10000$; $\kappa = \sqrt{n}$ (top row), n^2(bottom row).

A. Implementation for Data with Sparse Structure

A natural question one might want to ask is whether the mS2GD can take advantage of sparse data. In the case of SGD, if the i^{th} data point depends only on few coordinates, say ω, computing the gradient of the i^{th} function can cost [6] $\mathcal{O}(\omega)$ operations, and the resulting gradient will have just ω nonzero elements. Thus, updating the test point will require $\mathcal{O}(\omega)$ operations.

This is not the case for mS2GD, since the full gradient in the update (5) is, in general, fully dense even for sparse data. However, taking in to account that we do not need all coordinates to compute the stochastic estimate of the gradient due to sparse data, we can adapt the algorithm, to make update in respective coordinates only before they are needed to evaluate a stochastic gradient, which induces our application of proximal "lazy" updates into Algorithm 1. For example, in Algorithm 1, for inner iteration t at epoch k, we would only update $G_{k,t}$ with coordinates in A_{kt} for the full gradient g_k.

In order to illustrate our mS2GD algorithm with sparse data in Algorithm 2, we define the "lazy" update operator as "prox$^l[\cdot]$", which is a lazy update for the i^{th} coordinate. This operator is able to fulfill the cases for both L1 and L2 regularizers, and details about this operator will be explained after introducing the algorithm.

In Algorithm 2, we assume the functions f_i have the form

$$f_i(x) = \phi(a_i^T x)$$

for all $i = 1, \ldots, n$, which covers the cases of linear and logistic regression. We denote the s^{th} coordinate of a vector y by $(y)_s$.

[6]For example, this is the case for linear/logistic regression.

Algorithm 2 mS2GD in proximal setting, using "lazy" updates

Input: m (max # of stochastic steps per epoch); $h > 0$ (stepsize); $x_0 \in \mathbb{R}^d$ (starting point); mini-batch size $b \in [n]$.
for $k = 0, 1, 2, \ldots$ **do**
  Compute and store $g_k \leftarrow \nabla F(x_k) = \frac{1}{n}\sum_i \nabla f_i(x_k)$
  Initialize the inner loop: $y_{k,0} \leftarrow x_k$
  $\chi_j \leftarrow 0$ for $j = 1, 2, \ldots, n$
  Let $t_k \leftarrow t \in \{1, 2, \ldots, m\}$ with probability q_t given by (6)
  for $t = 0$ to $t_k - 1$ **do**
    Choose mini-batch $A_{kt} \subset [n]$ of size b, uniformly at random
    for $i \in A_{kt}$ **do**
      for $s \in \text{nnz}(a_i)$ **do**
        $(y_{k,t})_s \leftarrow \text{prox}^l[(y_{k,t})_s, (g_k)_s, t - \chi_s, \lambda, h]$
        $\chi_s \leftarrow t$
      end for
    end for
    $y_{k,t+1} \leftarrow y_{k,t} - \frac{h}{b}\sum_{i \in A_{kt}}(\nabla f_i(y_{k,t}) - \nabla f_i(x_k))$
  end for
  for $s = 1$ to d **do**
    $(y_{k,t})_s \leftarrow \text{prox}^l[(y_{k,t})_s, (g_k)_s, t_k - \chi_s, \lambda, h]$
  end for
  Set $x_{k+1} \leftarrow y_{k,t_k}$
end for

This algorithm follows the scheme of Algorithm 1 while the only difference is the application of lazy updates for proximal operators. Problems with various regularizers can be performed with efficient updates by using "prox^l" operator for sparse data. The most popular regularizers in machine learning

research are L1 and L2 regularizers. The following lemmas include details about proximal lazy updates with L2 and L1 regularizers, respectively.

Lemma 1 (Proximal Lazy Updates with L2 Regularizer)**.** *For the problem* (14)*, which has L2 regularizer in* (1) *($\lambda \neq 0$) , our mS2GD algorithm can efficiently perform proximal lazy updates for sparse data by using the following operator in Algorithm 2.*

$$prox^l[(y_{k,t})_s,(g_k)_s,\tau,\lambda,h] = \beta^\tau (y_{k,t})_s - \tfrac{h\beta}{1-\beta}\left[1-\beta^\tau\right](g_k)_s,$$

where s corresponds to the s^{th} coordinate and $\beta \overset{def}{=} 1/(1+\lambda h)$ is predefined.

Lemma 2 (Proximal Lazy Updates with L1 Regularizer)**.** *Consider problem* (1) *with L1 regularizer $R(x) = \|x\|_1$. Our mS2GD algorithm can efficiently update the iterates for sparse data by using the proximal lazy update operator in Algorithm 2. Define M and m as follows,*

$$M = [\lambda + (g_k)_s]h, \qquad m = -[\lambda - (g_k)_s]h,$$

then the forms of the operator are distinct in the following three situations, according to the value of $(g_k)_s$.

1) *If $(g_k)_s \geq \lambda$, then by letting $p \overset{def}{=} \left\lceil \frac{(y_k)_s}{M} \right\rceil - 1$, the operator can be defined as*

$$prox^l[(y_{k,t})_s,(g_k)_s,\tau,\lambda,h] = \begin{cases} (y_{k,t})_s - \tau M, & \text{if } p \geq \tau, \\ \min\{(y_{k,t})_s, m\} + (\max\{p,0\} - \tau)m, & \text{if } p < \tau. \end{cases}$$

2) *If $-\lambda < (g_k)_s < \lambda$, then the operator can be defined as*

$$prox^l[(y_{k,t})_s,(g_k)_s,\tau,\lambda,h] = \begin{cases} \max\{(y_{k,t})_s - \tau M, 0\}, & \text{if } (y_k)_s \geq 0, \\ \min\{(y_{k,t})_s - \tau m, 0\}, & \text{if } (y_k)_s < 0. \end{cases}$$

3) *If $(g_k)_s \leq -\lambda$, then by letting $q \overset{def}{=} \left\lceil \frac{(y_{k,t})_s}{m} \right\rceil - 1$, the operator can be defined as*

$$prox^l[(y_{k,t})_s,(g_k)_s,\tau,\lambda,h] = \begin{cases} (y_{k,t})_s - \tau m, & \text{if } q \geq \tau, \\ \max\{(y_{k,t})_s, M\} + (\max\{q,0\} - \tau)M, & \text{if } q < \tau. \end{cases}$$

The proofs of Lemmas 1 and 2 are available in APPENDIX C.

B. Speedup of mS2GD

With mini-batches, mS2GD can be accelerated in the benefit of simple parallelism. In Section IV-A, we have shown in theory that up to some threshold of mini-batch size, increasing mini-batch size does not hurt the performance of mS2GD.

Fig. 2 compares the best performances of mS2GD with different mini-batch sizes on datasets *rcv1* and *astro-ph*. Each effective pass is considered as n evaluations in component gradients and each full gradient evaluation counts as one effective pass. In both cases, by increasing the mini-batch size, mS2GD with $b = 2, 4, 8$ are comparable or sometimes even better than S2GD ($b = 1$) without any parallelism.

Although for larger mini-batch sizes, mS2GD would be obviously worse, the results are still promising with parallelism. In Fig. 3, we show the ideal speedup by parallelism, which would be achievable if and only if we could always evaluate the b gradients efficiently in parallel [7].

C. Comparison with Related Algorithms

In this part, we implemented the following algorithms to conduct a comparison.

1) **SGDcon**: the proximal stochastic gradient descent method with the constant step-size which gave the best performance in hindsight.
2) **SGD+**: the proximal stochastic gradient descent with adaptive step-size $h = h_0/(k+1)$, where k is the number of effective passes and h_0 is some initial constant step-size. We used h_0 which gave the best performance in hindsight.
3) **FISTA**: fast iterative shrinkage-thresholding algorithm proposed in [3]. This is considered as the full gradient descent method in our experiments.
4) **SAG**: a proximal version of the stochastic average gradient algorithm [26]. Instead of using $h = 1/16L$, which is analyzed in the reference, we used the constant step-size, which provided the best performance in practice, instead of using $h = 1/16L$, which is analyzed in the reference.
5) **S2GD**: semi-stochastic gradient descent method proposed in [1]. We applied proximal setting to the algorithm and used constant step-size, which gave the best performance in hindsight.
6) **mS2GD**: the mS2GD algorithm with mini-batch size $b = 8$. Although a safe step-size is given in our theoretical analyses in Theorem 1, we ignored the bound, experimented with various step-sizes and used the constant step-size that gave the best performance.

Fig. 4 demonstrates superiority of mS2GD over state-of-the-art algorithms on the four datasets. For mS2GD, the best choices of parameters with $b = 8$ are given in TABLE II.

Parameter	*rcv1*	*news20*	*covtype*	*astro-ph*
m	0.11n	0.10n	0.07n	0.08n
h	5.5/L	6/L	350/L	6.5/L

TABLE II: Best choices of parameters in mS2GD.

VI. Conclusion

In this paper, we have proposed a mini-batch proximal algorithm, with variance reduction technique on stochastic gradients, for minimizing a strongly convex composite function, which is the sum of a smooth convex function and a possibly nonsmooth regularizer. This kind of unconstrained optimization problems arise in inverse problems in signal processing

[7] In practice, it is impossible to ensure that the times of evaluating different component gradients are the same.

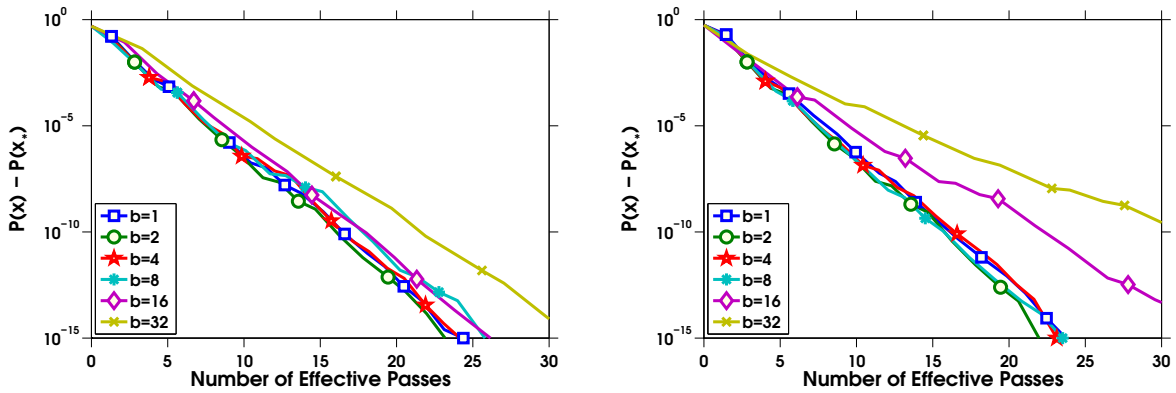

Fig. 2: Comparison of mS2GD with different mini-batch sizes on *rcv1* (left) and *astro-ph* (right).

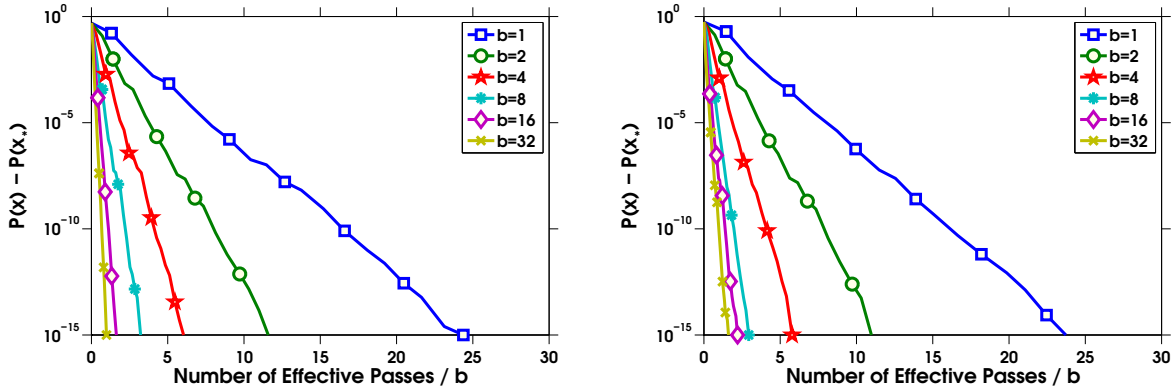

Fig. 3: Speedup from parallelism for *rcv1* (left) and *astro-ph* (right) in theory (unachievable in practice).

and statistics. Our mS2GD algorithm enjoys speedup from both mini-batch and variance reduction of stochastic gradients, where the former admits a simple parallelism. Comparisons to state-of-the-art algorithms suggest mS2GD, with a small mini-batch size, is competitive in theory and faster in practice even without parallelism. Possible implementation in parallel and adaptiveness for sparse data imply its potential in industry.

Appendix A
Technical Results

Lemma 3 (Lemma 2 in [2]). *Let R be a closed convex function on $\mathbb{R}^d$ and $x, y \in \mathrm{dom}(R)$, then*

$$\|\operatorname{prox}_R(x) - \operatorname{prox}_R(y)\| \leq \|x - y\|.$$

Note that the above contractiveness of proximal operator is a standard result in optimization literature.

Lemma 4. *Let $\{\xi_i\}_{i=1}^n$ is a collection of vectors in $\mathbb{R}^d$ and $\mu \overset{def}{=} \frac{1}{n}\sum_{i=1}^n \xi_i \in \mathbb{R}^d$. Let $\hat{S}$ be a τ-nice sampling. Then*

$$\mathbf{E}\left[\left\|\frac{1}{\tau}\sum_{i\in\hat{S}}\xi_i - \mu\right\|^2\right] = \frac{1}{n\tau}\frac{n-\tau}{(n-1)}\sum_{i=1}^n \|\xi_i\|^2. \qquad (16)$$

Following from the proof of Corollary 3 in [2], by applying Lemma 4 with $\xi_i := \nabla f_i(y_{k,t}) - \nabla f_i(x_k)$, we have the bound for variance as follows.

Theorem 4 (Bounding Variance). *Considering the definition of $G_{k,t}$ in Algorithm 1, conditioned on $y_{k,t}$, we have $\mathbf{E}[G_{k,t}] = \nabla F(y_{k,t})$ and the variance satisfies,*

$$\mathbf{E}\left[\|G_{k,t} - \nabla F(y_{k,t})\|^2\right] \leq \underbrace{\frac{n-b}{b(n-1)}}_{\alpha(b)} 4L(P(y_{k,t}) - P(x_*) + P(x_k) - P(x_*)). \qquad (17)$$

Appendix B
Proofs

A. Proof of Lemma 4

Note that

$$\eta \overset{\text{def}}{=} \mathbf{E}\left[\|\frac{1}{\tau}\sum_{i\in\hat{S}}\xi_i - \mu\|^2\right] = \mathbf{E}\left[\frac{1}{\tau^2}\|\sum_{i\in\hat{S}}\xi_i\|^2\right] - \|\mu\|^2$$
$$= \frac{1}{\tau^2}\mathbf{E}\left[\sum_{i\in\hat{S}}\sum_{j\in\hat{S}}\xi_i^T\xi_j\right] - \|\mu\|^2.$$

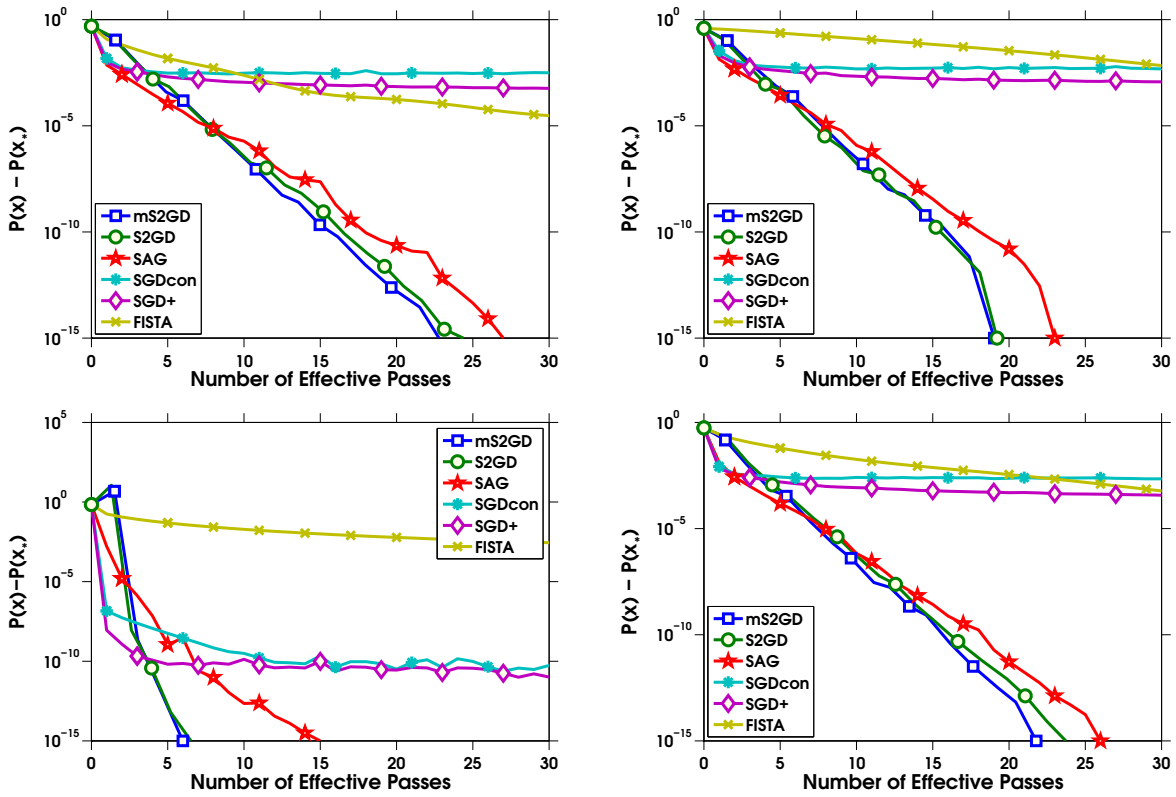

Fig. 4: Comparison of different algorithms on four datasets: *rcv1* (top left), *news20* (top right), *covtype* (bottom left) and *astro-ph* (bottom right); $b = 8$ in mS2GD.

If we let $C \overset{\text{def}}{=} \frac{1}{n^2}\left(\sum_{i,j}\xi_i^T\xi_j\right)$, we can thus write:

$$\eta = \frac{1}{\tau^2}\left(\frac{\tau(\tau-1)}{n(n-1)}\sum_{i\neq j}\xi_i^T\xi_j + \frac{\tau}{n}\sum_{i=1}^n \xi_i^T\xi_i\right) - C$$

$$= \frac{1}{\tau^2}\left(\frac{\tau(\tau-1)}{n(n-1)}\sum_{i,j}\xi_i^T\xi_j + \left(\frac{\tau}{n} - \frac{\tau(\tau-1)}{n(n-1)}\right)\sum_{i=1}^n \xi_i^T\xi_i\right) - C$$

$$= \frac{1}{n\tau}\left[-\left(-\frac{(\tau-1)}{(n-1)} + \frac{\tau}{n}\right)\sum_{i,j}\xi_i^T\xi_j + \frac{n-\tau}{n-1}\sum_{i=1}^n \xi_i^T\xi_i\right]$$

$$= \frac{1}{n\tau}\frac{n-\tau}{(n-1)}\left[\sum_{i=1}^n \xi_i^T\xi_i - \frac{1}{n}\sum_{i,j}\xi_i^T\xi_j\right] \leq \underbrace{\frac{n-\tau}{\tau(n-1)}}_{\alpha(\tau)}\frac{1}{n}\sum_{i=1}^n \|\xi_i\|^2,$$

where in the last step we have used the bound

$$\frac{1}{n}\sum_{i,j}\xi_i^T\xi_j = n\left\|\sum_{i=1}^n \frac{1}{n}\xi_i\right\|^2 \geq 0.$$

B. Proof of Theorem 1

The proof is following the steps in [2]. For convenience, let us define the stochastic gradient mapping

$$d_{k,t} = \frac{1}{h}(y_{k,t} - y_{k,t+1}) = \frac{1}{h}(y_{k,t} - \text{prox}_{hR}(y_{k,t} - hG_{k,t})),$$

then the iterate update can be written as

$$y_{k,t+1} = y_{k,t} - hd_{k,t}.$$

Let us estimate the change of $\|y_{k,t+1} - x_*\|$. It holds that

$$\begin{aligned}\|y_{k,t+1} - x_*\|^2 &= \|y_{k,t} - hd_{k,t} - x_*\|^2 \\ &= \|y_{k,t} - x_*\|^2 - 2hd_{k,t}^T(y_{k,t-1} - x_*) + h^2\|d_{k,t}\|^2. \quad (18)\end{aligned}$$

Apply Lemma 3 in [2] with $x = y_{k,t}, v = G_{k,t}, x^+ = y_{k,t+1}, g = d_{k,t}$, $y = x_*$ and $\Delta = \Delta_{k,t} = G_{k,t} - \nabla F(y_{k,t})$, we have

$$\begin{aligned}-d_{k,t}^T(y_{k,t} - x_*) + \frac{h}{2}\|d_{k,t}\|^2 \leq & P(x_*) - P(y_{k,t+1}) \\ & - \frac{\mu_F}{2}\|y_{k,t} - x_*\|^2 - \frac{\mu_R}{2}\|y_{k,t+1} - x_*\|^2 \\ & - \Delta_{k,t}^T(y_{k,t+1} - x_*). \quad (19)\end{aligned}$$

Therefore,

$$\begin{aligned}\|y_{k,t+1} - x_*\|^2 &\overset{(18),(19)}{\leq} 2h\left(P(x_*) - P(y_{k,t+1}) - \frac{\mu_F}{2}\|y_{k,t} - x_*\|^2\right. \\ &\left. - \frac{\mu_R}{2}\|y_{k,t+1} - x_*\|^2 - \Delta_{k,t}^T(y_{k,t+1} - x_*)\right) + \|y_{k,t} - x_*\|^2 \\ &= \|y_{k,t} - x_*\|^2 - h\mu_F\|y_{k,t} - x_*\|^2 - h\mu_R\|y_{k,t+1} - x_*\|^2 \\ &- 2h\Delta_{k,t}^T(y_{k,t+1} - x_*) - 2h[P(y_{k,t+1}) - P(x_*)].\end{aligned}$$

After moving $h\mu_R\|y_{k,t+1} - x_*\|^2$ to the LHS, we obtain

$$\begin{aligned}\|y_{k,t+1} - x_*\|^2(1 + h\mu_R) \leq & (1 - h\mu_F)\|y_{k,t} - x_*\|^2 \\ & - 2h\Delta_{k,t}^T(y_{k,t+1} - x_*) - 2h[P(y_{k,t+1}) - P(x_*)],\end{aligned}$$

and dividing both sides by $(1+h\mu_F)$ gives us

$$\|y_{k,t+1}-x_*\|^2 \le \alpha\|y_{k,t}-x_*\|^2 + \frac{2h}{1+h\mu_R}\left(-\Delta_{k,t}^T(y_{k,t+1}-x_*) - [P(y_{k,t+1})-P(x_*)]\right). \tag{20}$$

In order to bound $-\Delta_{k,t}^T(y_{k,t+1}-x_*)$, let us define the proximal full gradient update as[8]

$$\bar{y}_{k,t+1} = \mathrm{prox}_{hR}(y_{k,t}-h\nabla F(y_{k,t})),$$

with which, by using Cauchy-Schwartz inequality and Lemma 3, we can conclude that

$$\begin{aligned}
&-\Delta_{k,t}^T(y_{k,t+1}-x_*)\\
&= -\Delta_{k,t}^T(y_{k,t+1}-\bar{y}_{k,t+1}) - \Delta_{k,t}^T(\bar{y}_{k,t+1}-x_*)\\
&= -\Delta_{k,t}^T(\bar{y}_{k,t+1}-x_*) - \Delta_{k,t}^T[\mathrm{prox}_{hR}(y_{k,t}-hG_{k,t})\\
&\quad - \mathrm{prox}_{hR}(y_{k,t-1}-h\nabla F(y_{k,t-1}))]\\
&\le \|\Delta_{k,t}\|\|(y_{k,t}-hG_{k,t}) - (y_{k,t}-h\nabla F(y_{k,t}))\|\\
&\quad - \Delta_{k,t}^T(\bar{y}_{k,t+1}-x_*),\\
&= h\|\Delta_{k,t}\|^2 - \Delta_{k,t}^T(\bar{y}_{k,t+1}-x_*).
\end{aligned} \tag{21}$$

Letting $\delta \overset{\text{def}}{=} \frac{1-h\mu_F}{1+h\mu_R}$, we thus get

$$\begin{aligned}
\|y_{k,t+1}-x_*\|^2 &\overset{(21),(20)}{\le} \delta\|y_{k,t}-x_*\|^2\\
&+\tfrac{2h}{1+h\mu_R}\left(h\|\Delta_{k,t}\|^2 - \Delta_{k,t}^T(\bar{y}_{k,t+1}-x_*)\right.\\
&\left.-[P(y_{k,t+1})-P(x_*)]\right).
\end{aligned}$$

By taking expectation, conditioned on $y_{k,t}$[9] we obtain

$$\begin{aligned}
\mathbf{E}[\|y_{k,t+1}-x_*\|^2] &\overset{(21),(20)}{\le} \delta\|y_{k,t}-x_*\|^2\\
+\tfrac{2h}{1+h\mu_R}&\left(h\,\mathbf{E}[\|\Delta_{k,t}\|^2] - \mathbf{E}[P(y_{k,t+1})-P(x_*)]\right),
\end{aligned} \tag{22}$$

where we have used that $\mathbf{E}[\Delta_{k,t}] = \mathbf{E}[G_{k,t}] - \nabla F(y_{k,t}) = 0$ and hence $\mathbf{E}[-\Delta_{k,t}^T(\bar{y}_{k,t+1}-x_*)] = 0$[10]. Now, if we put (17) into (22) and decrease index t by 1, we obtain

$$\begin{aligned}
\mathbf{E}[\|y_{k,t}-x_*\|^2] &\overset{(21),(20)}{\le} \delta\|y_{k,t-1}-x_*\|^2\\
&+\tfrac{2h}{1+h\mu_R}\{4Lh\alpha(b)[P(y_{k,t-1})-P(x_*)\\
&+P(x_k)-P(x_*)] - \mathbf{E}[P(y_{k,t})-P(x_*)]\},
\end{aligned} \tag{23}$$

where $\alpha(b) = \frac{n-b}{b(n-1)}$.

Now recall that we assume that we have lower-bounds $\nu_F \ge 0$ and $\nu_R \ge 0$ on the true strong convexity parameter μ_F and μ_R available. Letting

$$\delta' \overset{\text{def}}{=} \tfrac{1-h\nu_F}{1+h\nu_R},$$

we obtain from (23) that

$$\begin{aligned}
\mathbf{E}[\|y_{k,t}-x_*\|^2] &\overset{(23)}{\le} \delta'\|y_{k,t-1}-x_*\|^2\\
&+\tfrac{2h}{1+h\nu_R}\{4Lh\alpha(b)(P(y_{k,t-1})-P(x_*)\\
&+P(x_k)-P(x_*)) - \mathbf{E}[P(y_{k,t})-P(x_*)]\},
\end{aligned}$$

which is equivalent to

$$\begin{aligned}
&\mathbf{E}[\|y_{k,t}-x_*\|^2] + \tfrac{2h}{1+h\nu_R}(\mathbf{E}[P(y_{k,t})-P(x_*)])\\
&\le \delta'\|y_{k,t-1}-x_*\|^2\\
&\quad + \tfrac{8h^2L\alpha(b)}{1+h\nu_R}(P(y_{k,t-1})-P(x_*)+P(x_k)-P(x_*)).
\end{aligned} \tag{24}$$

Now, by the definition of x_k we have that

$$\mathbf{E}[P(x_{k+1})] = \tfrac{1}{\gamma}\sum_{t=1}^{m}(\delta')^{m-t}\,\mathbf{E}[P(y_{k,t})], \tag{25}$$

where $\gamma = \sum_{t=1}^{m}(\delta')^{m-t}$. By summing (24), multiplied by $(\delta')^{m-t}$ for $t = 1,\dots,m$, we obtain on the left hand side

$$\begin{aligned}
LHS = &\sum_{t=1}^{m}(\delta')^{m-t}\,\mathbf{E}[\|y_{k,t}-x_*\|^2]\\
&+\tfrac{2h}{1+h\nu_R}\sum_{t=1}^{m}(\delta')^{m-t}\,\mathbf{E}[P(y_{k,t})-P(x_*)]
\end{aligned} \tag{26}$$

and for the right hands side we have:

$$\begin{aligned}
RHS = &\sum_{t=1}^{m}(\delta')^{m-t+1}\,\mathbf{E}\,\|y_{k,t-1}-x_*\|^2\\
&+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\sum_{t=1}^{m}(\delta')^{m-t}\,\mathbf{E}[P(y_{k,t-1})\\
&-P(x_*)+P(x_k)-P(x_*)]\\
= &\sum_{t=0}^{m-1}(\delta')^{m-t}\,\mathbf{E}\,\|y_{k,t}-x_*\|^2\\
&+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\tfrac{1}{\delta'}\cdot\left(\sum_{t=0}^{m-1}(\delta')^{m-t}\,\mathbf{E}[P(y_{k,t})-P(x_*)]\right)\\
&+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\,\mathbf{E}[P(x_k)-P(x_*)]\gamma.
\end{aligned}$$

Therefore,

$$\begin{aligned}
RHS \le &\sum_{t=0}^{m-1}(\delta')^{m-t}\,\mathbf{E}\,\|y_{k,t}-x_*\|^2\\
&+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\tfrac{1+h\nu_R}{1-h\nu_F}\cdot\left(\sum_{t=0}^{m}(\delta')^{m-t}\,\mathbf{E}[P(y_{k,t})-P(x_*)]\right)\\
&+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\,\mathbf{E}[P(x_k)-P(x_*)]\gamma.
\end{aligned} \tag{27}$$

Combining (26) and (27) and using the fact that $LHS \le RHS$, we have

$$\begin{aligned}
&\mathbf{E}[\|y_{k,m}-x_*\|^2] + \tfrac{2h}{1+h\nu_R}\sum_{t=1}^{m}(\delta')^{m-t}\,\mathbf{E}[P(y_{k,t})-P(x_*)]\\
&\le (\delta')^m\,\mathbf{E}\,\|y_{k,0}-x_*\|^2\\
&\quad + \tfrac{8h^2L\alpha(b)}{1+h\nu_R}\,\mathbf{E}[P(x_k)-P(x_*)]\gamma + \tfrac{8h^2L\alpha(b)}{1+h\nu_R}\tfrac{1+h\nu_R}{1-h\nu_F}\\
&\quad\cdot\left(\sum_{t=1}^{m}(\delta')^{m-t}\,\mathbf{E}[P(y_{k,t})-P(x_*)]\right)\\
&+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\tfrac{1+h\nu_R}{1-h\nu_F}\left((\delta')^m\,\mathbf{E}[P(y_{k,0})-P(x_*)]\right).
\end{aligned}$$

[8]Note that this quantity is never computed during the algorithm. We can use it in the analysis nevertheless.

[9]For simplicity, we omit the $\mathbf{E}[\cdot\,|\,y_{k,t}]$ notation in further analysis

[10]$\bar{y}_{k,t+1}$ is constant, conditioned on $y_{k,t}$

Now, using (25), we obtain

$$\begin{aligned}&\mathbf{E}[\|y_{k,m}-x_*\|^2]+\tfrac{2h}{1+h\nu_R}\gamma(\mathbf{E}[P(x_{k+1})]-P(x_*))\\&\le(\delta')^m\,\mathbf{E}\,\|y_{k,0}-x_*\|^2+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\gamma\,\mathbf{E}[P(x_k)-P(x_*)]\\&\quad+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\tfrac{1}{\delta'}\gamma\left(\mathbf{E}[P(x_{k+1})]-P(x_*)\right)\\&\quad+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\tfrac{1}{\delta}\left(\delta'\right)^m\mathbf{E}[P(y_{k,0})-P(x_*)].\end{aligned}\tag{28}$$

Strong convexity (3) and optimality of x_* imply that $0\in\partial P(x_*)$, and hence for all $x\in\mathbb{R}^d$ we have

$$\|x-x_*\|^2\le\tfrac{2}{\mu}[P(x)-P(x_*)].\tag{29}$$

Since $\mathbf{E}\,\|y_{k,m}-x_*\|^2\ge 0$ and $y_{k,0}=x_k$, by combining (29) and (28) we get

$$\begin{aligned}&2\gamma h\left\{\tfrac{1}{1+h\nu_R}-\tfrac{4hL\alpha(b)}{1-h\nu_F}\right\}\left(\mathbf{E}[P(x_{k+1})]-P(x_*)\right)\\&\le\left(P(x_k)-P(x_*)\right)\left\{\left(\delta'\right)^m\tfrac{2}{\mu}+\tfrac{8h^2L\alpha(b)}{1+h\nu_R}\left(\gamma+\left(\delta'\right)^{m-1}\right)\right\}.\end{aligned}$$

This is equivalent to

$$\mathbf{E}[P(x_{k+1})-P(x_*)]\le\rho[P(x_k)-P(x_*)],$$

when $\frac{1}{1+h\nu_R}-\frac{4hL\alpha(b)}{1-h\nu_F}>0$ (which is equivalent to $\frac{1-h\nu_F}{1+h\nu_R}\frac{1}{4L\alpha(b)}\ge h$), and when ρ is defined as

$$\rho=\frac{\left(\frac{1-h\nu_F}{1+h\nu_R}\right)^m\frac{1}{\mu}+\frac{4h^2L\alpha(b)}{1+h\nu_R}\left(\gamma+\left(\frac{1-h\nu_F}{1+h\nu_R}\right)^{m-1}\right)}{\gamma h\left\{\frac{1}{1+h\nu_R}-\frac{4hL\alpha(b)}{1-h\nu_F}\right\}}.$$

The above statement, together with assumptions of Lemma 3 in [2], implies

$$0<h<\min\left\{\tfrac{1-h\nu_F}{1+h\nu_R}\tfrac{1}{4L\alpha(b)},\tfrac{1}{L}\right\}.$$

Applying the above linear convergence relation recursively with chained expectations, we have

$$\mathbf{E}[P(x_k)-P(x_*)]\le\rho^k[P(x_0)-P(x_*)].$$

C. Proof of Theorem 2

Clearly, if we choose some value of h then the value of m will be determined from (8) (i.e. we need to choose m such that we will get desired rate). Therefore, m as a function of h obtained from (8) is

$$m(h)=\frac{1+4\alpha(b)h^2L\mu}{h\mu(\rho-4\alpha(b)hL(\rho+1))}.\tag{30}$$

Now, we can observe that the nominator is always positive and the denominator is positive only if

$$\rho>4\alpha(b)hL(\rho+1)\Rightarrow\frac{1}{4\alpha(b)L}\cdot\underbrace{\frac{\rho}{\rho+1}}_{\in[0,\frac{1}{2}]}>h.$$

Note that this condition is stronger than the one in assumption of Theorem 1. It is easy to verify that

$$\lim_{h\searrow 0}m(h)=+\infty,\qquad\lim_{h\nearrow\frac{1}{4\alpha(b)L}\cdot\frac{\rho}{\rho+1}}m(h)=+\infty.$$

Also note that $m(h)$ is differentiable (and continuous) at any $h\in(0,\frac{1}{4\alpha(b)L}\cdot\frac{\rho}{\rho+1})=:I_h$. Let us look at the $m'(h)$. We have

$$m'(h)=\frac{-\rho+4\alpha(b)hL(2+(2+h\mu)\rho)}{h^2\mu(\rho-4\alpha(b)hL(1+\rho))^2}.$$

Observe that $m'(h)$ is defined and continuous for any $h\in I_h$. Therefore there have to be some stationary points (and in case that there is just on I_h) it will be the global minimum on I_h. The FOC gave us that

$$\begin{aligned}\tilde{h}^b&=\frac{-2\alpha(b)L(1+\rho)+\sqrt{\alpha(b)L(\mu\rho^2+4\alpha(b)L(1+\rho)^2)}}{2\alpha(b)L\mu\rho}\\&=\sqrt{\frac{1}{4\alpha(b)L\mu}+\frac{(1+\rho)^2}{\mu^2\rho^2}}-\frac{1+\rho}{\mu\rho}.\end{aligned}\tag{31}$$

If this $\tilde{h}^b\in I_h$ and also $\tilde{h}^b\le\frac{1}{L}$ then this is the optimal choice and plugging (31) into (30) gives us (10).

a) Claim #1: It always holds that $\tilde{h}^b\in I_h$. We just need to verify that

$$\sqrt{\frac{1}{4\alpha(b)L\mu}+\frac{(1+\rho)^2}{\mu^2\rho^2}}-\frac{1+\rho}{\mu\rho}<\frac{1}{4\alpha(b)L}\cdot\frac{\rho}{\rho+1},$$

which is equivalent to

$$\begin{aligned}&\mu\rho^2+4\alpha(b)L(1+\rho)^2\\&>2(1+\rho)\sqrt{\alpha(b)L(\mu\rho^2+4\alpha(b)L(1+\rho)^2)}.\end{aligned}$$

Because both sides are positive, we can square them to obtain an equivalent condition

$$\begin{aligned}&\left(\mu\rho^2+4\alpha(b)L(1+\rho)^2\right)^2\\&>4(1+\rho)^2\alpha(b)L(\mu\rho^2+4\alpha(b)L(1+\rho)^2),\end{aligned}$$

which is equivalent with

$$\mu\rho^2(\mu\rho^2+4\alpha(b)L(1+\rho)^2)>0.$$

b) Claim #2: If $\tilde{h}^b>\frac{1}{L}$ then $h^b_*=\frac{1}{L}$. I believe that this is trivial.

Only one thing which needs to be verified is that the denominator of (11) is positive (or equivalently we want to show that $\rho>4\alpha(b)(1+\rho)$. To see that we just need to realize that in that case we have

$$\frac{1}{L}\le\tilde{h}^b\le\frac{1}{4\alpha(b)L}\cdot\frac{\rho}{\rho+1}$$

which implies that

$$1\le\frac{1}{4\alpha(b)}\cdot\frac{\rho}{\rho+1}\Rightarrow 4\alpha(b)(1+\rho)<\rho.$$

APPENDIX C
PROXIMAL LAZY UPDATES FOR L1 AND L2 REGULARIZERS

A. Proof of Lemma 1

By applying the updates with proximal operator (4) and L2 regularizer, from Algorithm 1 we have

$$\begin{aligned}y_{k+1,t}&=\mathrm{prox}_{hR}(y_{k,t}-hG_{k,t})\\&=\arg\min_{x\in\mathbb{R}^d}\left\{\tfrac{1}{2}\|x-(y_{k,t}-hG_{k,t})\|^2+\frac{\lambda h}{2}\|x\|^2\right\}\\&=\frac{1}{1+\lambda h}(y_{k,t}-hG_{k,t})=\beta(y_{k,t}-hG_{k,t}),\end{aligned}$$

where $\beta = 1/(1+\lambda h)$ and

$$G_{k,t} = g_k + \frac{1}{b}\sum_{i\in A_{kt}}(\nabla f_i(y_{k,t}) - \nabla f_i(x_k)).$$

In lazy updates, for the s^{th} coordinate, we have $(G_{k,t})_s = (g_k)_s$, which gives the lazy update for 1 iteration as

$$(y_{k+1,t})_s = \beta[(y_{k,t})_s - h(g_k)_s],$$

which by being applied recursively, gives lazy updates for τ iterations for the s^{th} coordinate as

$$\begin{aligned}(y_{k+\tau,t})_s &= \beta^\tau (y_{k,t})_s - h\left(\sum_{j=1}^{\tau}\beta^j\right)(g_k)_s \\ &= \beta^\tau (y_{k,t})_s - \frac{h\beta}{1-\beta}\left[1-\beta^\tau\right](g_k)_s \\ &\overset{\text{def}}{=} \text{prox}^l[(y_{k,t})_s, (g_k)_s, \tau, \lambda, h],\end{aligned}$$

when $\lambda \neq 0$.

B. Proof of Lemma 2

In (1), by letting $R(x) = \lambda\|x\|_1$, which is the L1-regularizer, we have the following update from Algorithm 1,

$$y_{k+1,t} = \text{prox}_{hR}(y_{k,t} - hG_{k,t}), \tag{32}$$

and the proximal operator can be calculated as

$$\text{prox}_{hR}(z) = \arg\min_x \frac{1}{2}\|x-z\|^2 + \lambda h\|x\|_1,$$

where each coordinates are separable, i.e., the s^{th} coordinate for (32) can be updated as

$$\begin{aligned}(y_{k+1,t})_s &= \arg\min_{x\in\mathbb{R}} \frac{1}{2}(x-z_s)^2 + \lambda h|x| \\ &= \begin{cases} z_s - \lambda h, & \text{if } z_s > \lambda h, \\ 0, & \text{if } -\lambda h \le z_s \le \lambda h, \\ z_s + \lambda h, & \text{if } z_s < -\lambda h, \end{cases}\end{aligned}$$

where $z_s = (y_{k,t})_s - h(G_{k,t})_s$ and

$$G_{k,t} = g_k + \frac{1}{b}\sum_{i\in A_{kt}}(\nabla f_i(y_{k,t}) - \nabla f_i(x_k)).$$

The lazy update does not include the mini-batch part, which means the lazy update proceeds with $(G_{k,t})_s = (g_k)_s$ for each coordinate s, therefore,

$$(y_{k+1,t})_s = \begin{cases}(y_{k,t})_s - [\lambda + (g_k)_s]h, & \text{if } (y_{k,t})_s > [\lambda+(g_k)_s]h, \\ 0, & \text{if } -[\lambda-(g_k)_s]h \le (y_{k,t})_s \le [\lambda+(g_k)_s]h, \\ (y_{k,t})_s + [\lambda-(g_k)_s]h, & \text{if } (y_{k,t})_s < -[\lambda-(g_k)_s]h, \end{cases}$$

and notice that $(\lambda+(g_k)_s)h - [-(\lambda-(g_k)_s)] = 2\lambda h > 0$. Depending on the value of $(g_k)_s$, there would be three situations for τ lazy updates listed as follows.

(1) When $-\lambda < (g_k)_s < \lambda$, then $\lambda + (g_k)_s > 0, -[\lambda - (g_k)_s] < 0$, thus we have that

$$(y_{k+\tau,t})_s = \begin{cases}\max\{(y_{k,t})_i - \tau[\lambda+(g_k)_s]h, 0\}, & \text{if } (y_{k,t})_s \ge 0, \\ \min\{(y_{k,t})_s + \tau[\lambda-(g_k)_s]h, 0\}, & \text{if } (y_{k,t})_s < 0. \end{cases}$$

(2) When $(g_k)_s \ge \lambda$, then $-[\lambda-(g_k)_s] \ge 0$, $\lambda + (g_k)_s \ge 2\lambda > 0$, thus by letting $p = \left\lceil \frac{(y_k)_s}{[\lambda+(g_k)_s]h} \right\rceil - 1$, we have that:
if $(y_{k,t})_s > [\lambda+(g_k)_s]h$, then

$$(y_{k+\tau,t})_s = \begin{cases}(y_{k,t})_s - \tau[\lambda+(g_k)_s]h, & \text{if } \tau \le p, \\ (\tau-p-1)[\lambda-(g_k)_s]h, & \text{if } \tau > p; \end{cases}$$

if $(y_{k,t})_s < -[\lambda-(g_k)_s]h$, then

$$(y_{k+\tau,t})_s = (y_{k,t})_s + \tau[\lambda-(g_k)_s]h;$$

if $-[\lambda-(g_k)_s] \le (y_{k,t})_s \le [\lambda+(g_k)_s]$, then

$$(y_{k+\tau,t})_s = (\tau-1)[\lambda-(g_k)_s]h.$$

(3) When $(g_k)_s \le -\lambda$, then $\lambda + (g_k)_s \le 0$, $-[\lambda-(g_k)_s] \le -2\lambda < 0$, thus by letting $q = \left\lceil \frac{(y_{k,t})_s}{-[\lambda-(g_k)_s]h} \right\rceil - 1$, we have that:
if $(y_{k,t})_s \ge [\lambda+(g_k)_s]$, then

$$(y_{k+\tau,t})_s = (y_{k,t})_s - \tau[\lambda+(g_k)_s]h;$$

if $(y_{k,t})_s < -[\lambda-(g_k)_s]$, then

$$(y_{k+\tau,t})_s = \begin{cases}(y_{k,t})_s + \tau[\lambda-(g_k)_s]h, & \text{if } \tau \le q, \\ -(\tau-q-1)[\lambda+(g_k)_s]h, & \text{if } \tau > q; \end{cases}$$

if $-[\lambda-(g_k)_s] \le (y_{k,t})_s \le [\lambda+(g_k)_s]$, then

$$(y_{k+\tau,t})_s = -(\tau-1)[\lambda+(g_k)_s]h.$$

For Case (2), when $(y_{k,t})_s \le [\lambda+(g_k)_s]$, we can conclude that

$$(y_{k+\tau,t})_s = \min\{(y_{k,t})_s, -[\lambda-(g_k)_s]h\} + \tau[\lambda-(g_k)_s]h.$$

Moreover, the following equivalences hold under the condition $(g_k)_s \ge \lambda$,

$$\begin{aligned}(y_{k,t})_s > [\lambda+(g_k)_s]h &\Leftrightarrow \frac{(y_{k,t})_s}{[\lambda+(g_k)_s]h} > 1 \Leftrightarrow p \ge 1, \\ (y_{k,t})_s \le [\lambda+(g_k)_s]h &\Leftrightarrow \frac{(y_{k,t})_s}{[\lambda+(g_k)_s]h} \le 1 \Leftrightarrow p \le 0,\end{aligned}$$

which simplifies the situation to

$$\begin{aligned}&(y_{k+\tau,t})_s \\ &= \begin{cases}(y_{k,t})_s - \tau[\lambda+(g_k)_s]h, & \text{if } p \ge \tau \\ (\tau-p-1)[\lambda-(g_k)_s]h, & \text{if } 1 \le p < \tau \\ \min\{(y_{k,t})_s, -[\lambda-(g_k)_s]h\} & \\ \qquad +\tau[\lambda-(g_k)_s]h, & \text{if } p \le 0 \end{cases} \\ &= \begin{cases}(y_{k,t})_s - \tau[\lambda+(g_k)_s]h, & \text{if } p \ge \tau, \\ \min\{(y_{k,t})_s, -[\lambda-(g_k)_s]h\} & \\ \qquad +(\tau-\max\{p,0\})[\lambda-(g_k)_s]h, & \text{if } p < \tau. \end{cases}\end{aligned}$$

For Case (3), when $(y_{k,t})_s \geq -[\lambda - (g_k)_s]h$, we can conclude that

$$(y_{k+\tau,t})_s = \max\{(y_{k,t})_s, \lambda + (g_k)_s\} - \tau[\lambda + (g_k)_s]h,$$

and in addition, the following equivalences hold when $(g_k)_s \leq -\lambda$,

$$(y_{k,t})_s < -[\lambda - (g_k)_s]h \;\Leftrightarrow\; \frac{(y_{k,t})_s}{-[\lambda - (g_k)_s]h} > 1 \;\Leftrightarrow\; q \geq 1,$$
$$(y_{k,t})_s \geq -[\lambda - (g_k)_s]h \;\Leftrightarrow\; \frac{(y_{k,t})_s}{-[\lambda - (g_k)_s]h} \leq 1 \;\Leftrightarrow\; q \leq 0,$$

which can summarize the situation as

$$\begin{aligned}
&(y_{k+\tau,t})_s \\
&\quad = \begin{cases} (y_{k,t})_s + \tau[\lambda - (g_k)_s]h, & \text{if } q \geq \tau \\ -(\tau - q - 1)[\lambda + (g_k)_s]h, & \text{if } 1 \leq q < \tau \\ \max\{(y_{k,t})_s, [\lambda + (g_k)_s]h\} & \\ \qquad -\tau[\lambda + (g_k)_s]h, & \text{if } q \leq 0 \end{cases} \\
&\quad = \begin{cases} (y_{k,t})_s + \tau[\lambda - (g_k)_s]h, & \text{if } q \geq \tau, \\ \max\{(y_{k,t})_s, [\lambda + (g_k)_s]h\} & \\ \qquad +(\max\{q, 0\} - \tau)[\lambda + (g_k)_s]h, & \text{if } q < \tau. \end{cases}
\end{aligned}$$

The proof can be completed by letting $M = [\lambda + (g_k)_s]h$ and $m = -[\lambda - (g_k)_s]h$.

REFERENCES

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

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

[3] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[4] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM J. Optimization*, vol. 22, pp. 341–362, 2012.

[5] P. Richtárik and M. Takáč, "Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function," *Mathematical Programming*, vol. 144, no. 1-2, pp. 1–38, 2014.

[6] P. Richtárik and M. Takáč, "Parallel coordinate descent methods for big data optimization," *arXiv:1212.0873*, 2012.

[7] I. Necoara and A. Patrascu, "A random coordinate descent algorithm for optimization problems with composite objective function and linear coupled constraints," *Comp. Optimization and Applications*, vol. 57, no. 2, pp. 307–337, 2014.

[8] O. Fercoq and P. Richtárik, "Accelerated, parallel and proximal coordinate descent," *arXiv:1312.5799*, 2013.

[9] S. Shalev-Shwartz and T. Zhang, "Stochastic dual coordinate ascent methods for regularized loss," *JMLR*, vol. 14, no. 1, pp. 567–599, 2013.

[10] J. Mareček, P. Richtárik, and M. Takáč, "Distributed block coordinate descent for minimizing partially separable functions," *arXiv:1406.0238*, 2014.

[11] I. Necoara and D. Clipici, "Distributed coordinate descent methods for composite minimization," *arXiv:1312.5302*, 2013.

[12] P. Richtárik and M. Takáč, "Distributed coordinate descent method for learning with big data," *arXiv:1310.2059*, 2013.

[13] O. Fercoq, Z. Qu, P. Richtárik, and M. Takáč, "Fast distributed coordinate descent for non-strongly convex losses," in *IEEE Workshop on Machine Learning for Signal Processing*, 2014.

[14] T. Zhang, "Solving large scale linear prediction using stochastic gradient descent algorithms," in *ICML*, 2004.

[15] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM J. Optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[16] C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Takáč, "Adding vs. averaging in distributed primal-dual optimization," *arXiv preprint arXiv:1502.03508*, 2015.

[17] M. Jaggi, V. Smith, M. Takáč, J. Terhorst, T. Hofmann, and M. I. Jordan, "Communication-efficient distributed dual coordinate ascent," *NIPS*, 2014.

[18] M. Takáč, A. S. Bijral, P. Richtárik, and N. Srebro, "Mini-batch primal and dual methods for SVMs," *ICML*, 2013.

[19] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," *NIPS*, pp. 315–323, 2013.

[20] A. Nitanda, "Stochastic proximal gradient descent with acceleration techniques," *NIPS*, 2014.

[21] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: Primal estimated sub-gradient solver for svm," *Mathematical Programming: Series A and B- Special Issue on Optimization and Machine Learning*, pp. 3–30, 2011.

[22] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *JMLR*, vol. 13, no. 1, pp. 165–202, 2012.

[23] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan, "Better mini-batch algorithms via accelerated gradient methods," in *NIPS*, 2011, pp. 1647–1655.

[24] P. Zhao and T. Zhang, "Accelerating minibatch stochastic gradient descent using stratified sampling," *arXiv:1405.3080*, 2014.

[25] S. Shalev-Shwartz and T. Zhang, "Accelerated mini-batch stochastic dual coordinate ascent," in *NIPS*, 2013, pp. 378–385.

[26] N. Le Roux, M. Schmidt, and F. Bach, "A stochastic gradient method with an exponential convergence rate for finite training sets," *NIPS*, 2012.

[27] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer, Boston, 2004.

[28] ——, "Gradient methods for minimizing composite objective function," *CORE Discussion Papers*, 2007.