On Convergence of Model Parallel Proximal Gradient Algorithm for Stale Synchronous Parallel System

Yi Zhou* **Yaoliang Yu†** **Wei Dai†** **Yingbin Liang*** **Eric P. Xing†**
*Syracuse University †Carnegie Mellon University

Abstract

With ever growing data volume and model size, an error-tolerant, communication efficient, yet versatile parallel algorithm has become a vital part for the success of many large-scale applications. In this work we propose msPG, an extension of the flexible proximal gradient algorithm to the model parallel and stale synchronous setting. The worker machines of msPG operate asynchronously as long as they are not too far apart, and they communicate efficiently through a dedicated parameter server. Theoretically, we provide a rigorous analysis of the various convergence properties of msPG, and a salient feature of our analysis is its seamless generality that allows both nonsmooth and nonconvex functions. Under mild conditions, we prove the *whole* iterate sequence of msPG converges to a critical point (which is optimal under convexity assumptions). We further provide an economical implementation of msPG, completely bypassing the need of keeping a local full model. We confirm our theoretical findings through numerical experiments.

1 Introduction

Many machine learning and statistics problems fit into the general composite minimization framework:

$$\min_{\mathbf{x}\in\mathbb{R}^d} \ \frac{1}{n}\sum_{i=1}^n f_i(\mathbf{x}) + g(\mathbf{x}), \tag{1}$$

where the first term is typically a smooth empirical risk over n training samples and the second term g is a nonsmooth regularizer that promotes structures. Popular examples under this framework include the Lasso (least squares loss f_i and various sparse regularizers g), logistic regression (logistic loss f_i), boosting (exponential loss f_i), support vector machines (hinge loss f_i and RKHS norm regularizer g), matrix completion (least squares loss f and trace norm regularizer g), etc. There is also rising interest in using nonconvex losses (mainly for robustness to outliers) and nonconvex regularizers (mainly for smaller bias in feature selection and support estimation), see e.g. [15, 26, 31, 35, 36, 38–40].

Due to the apparent importance of the composite minimization framework and the rapidly growing size in both model dimension and sample volume, there is a strong need to develop a practical *parallel* system that can solve the problem in (1) efficiently and in a scale that is impossible for a single machine [2, 8, 14, 16, 19, 21, 25, 37]. Existing systems can roughly be divided into three categories: bulk synchronous [14, 34, 37], (totally) asynchronous [8, 25], and partially asynchronous (also called stale synchronous in this work) [2, 8, 16, 19, 21, 32]. The bulk synchronous parallel mechanism (BSP) forces synchronization barriers so that the worker machines can stay on the same page to ensure correctness. However, in a real deployed parallel system BSP usually suffers from the straggler problem, that is, the performance of the whole system is bottlenecked by the *slowest* worker machine. On the other hand, asynchronous systems achieve much greater throughputs, although at the expense of potentially losing the correctness of the algorithm. The stale synchronous parallel (SSP) mechanism is a compromise between the previous two mechanisms: it allows the worker machines to operate asynchronously, as long as they are not too far apart. SSP is particularly suitable for machine learning applications, where iterative algorithms robust to small errors are usually used to find an appropriate model. This view is also practiced by many recent works building on the SSP mechanism [2, 16, 19, 21, 24, 27].

Existing parallel systems can also be divided into data parallel and model parallel. In the former case,

Appearing in Proceedings of the 19^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51.

one usually distributes the computation involving each component function f_i in (1) into different worker machines. This is suitable when $n \gg d$, i.e. large data volume but moderate model size. A popular algorithm for this case is the stochastic gradient algorithm and its proximal versions [2, 16, 19, 21], under the SSP mechanism. In contrast, model parallel refers to the regime where $d \gg n$, i.e. large model size but moderate data volume. This is the case for many computational biology and health care problems, where collecting many samples can be very expensive but for each sample we can relatively cheaply take a large number of measurements (features). As a result we need to partition the model $\mathbf{x}$ into different (disjoint) blocks and distribute them among many worker machines. The proximal gradient [11, 18] or its accelerated version [6] is again a natural candidate algorithm due to its nice ability of handling nonsmooth regularizers. However, such proximal gradient algorithm has not been investigated under SSP mechanism for model parallelism, although some other types of asynchronous algorithms are studied before (see Section 7). The main goal of this work is to fill in this important gap.

More specifically, we make the following contributions: 1). We propose msPG, an extension of the proximal gradient algorithm to the new model parallel and stale synchronous setting. 2). We provide a rigorous analysis of the convergence properties of msPG. Under a very general condition that allows both *nonsmooth* and *nonconvex* functions we prove in Theorem 1 that any limit point of the sequence generated by msPG is a critical point. Then, inspired by the recent Kurdyka-Łojasiewicz (KŁ) inequality [3, 5, 9, 11, 20], we further prove in Theorem 2 that the whole sequence of msPG in fact converges to a critical point, under mild technical assumptions that we verify for many familiar examples. Lastly, relating msPG to recent works on inexact proximal gradient (on a single machine), we provide, under the new model parallel and SSP setting, a simple proof of the usual sublinear $O(1/t)$ rate of convergence (assuming convexity). We remark our technical contributions with comparison to related work after each main results. 3). Building on the recent parameter server framework [19, 21], we give an economical implementation of msPG that completely avoids storing local full models in each worker machine. The resulting implementation only requires storing the partitioned data (with size $O(nd_i)$ for d_i assigned parameters) and communicating a vector of length n in each iteration. 4). We corroborate our theoretical findings with controlled numerical experiments.

This paper proceeds as follows: We first recall some definitions in §2, followed by the proposed algorithm msPG in §3. Theoretical findings are reported in §4 (with all proofs deferred to the appendix). An economical implementation of msPG is detailed in §5 and experimentally verified in §6. Finally, we discuss some related works in §7 and conclude in §8.

2 Preliminaries

We collect here some useful definitions that will be needed in our later analysis.

Since we consider a proper and closed[1] function $h : \mathbb{R}^d \to (-\infty, +\infty]$ that may *not* be smooth or convex, we need a generalized notion of "derivative".

Definition 1 (Subdifferential and critical point, [29])**.** *The Frechét subdifferential $\hat{\partial} h$ of h at $\mathbf{x} \in \operatorname{dom} h$ is the set of $\mathbf{u}$ such that*

$$\liminf_{\mathbf{z} \neq \mathbf{x}, \mathbf{z} \to \mathbf{x}} \frac{h(\mathbf{z}) - h(\mathbf{x}) - \mathbf{u}^\top(\mathbf{z}-\mathbf{x})}{\|\mathbf{z}-\mathbf{x}\|} \geq 0, \tag{2}$$

while the (limiting) subdifferential ∂h at $\mathbf{x} \in \operatorname{dom} h$ is the graphical closure of $\hat{\partial} h$:

$$\{\mathbf{u} : \exists \mathbf{x}^k \to \mathbf{x}, h(\mathbf{x}^k) \to h(\mathbf{x}), \mathbf{u}^k \in \hat{\partial} h(\mathbf{x}^k) \to \mathbf{u}\}. \tag{3}$$

The critical points of h are $\operatorname{crit} h := \{\mathbf{x} : \mathbf{0} \in \partial h(\mathbf{x})\}$.

Pleasantly, when h is continuously differentiable or convex, the subdifferential ∂h and critical points $\operatorname{crit} h$ coincide with the usual notions.

Definition 2 (Distance and projection)**.** *The distance function to a closed set $\Omega \subseteq \mathbb{R}^d$ is defined as:*

$$\operatorname{dist}_\Omega(\mathbf{x}) := \min_{\mathbf{y} \in \Omega} \|\mathbf{y} - \mathbf{x}\|, \tag{4}$$

and the metric projection onto Ω is:

$$\operatorname{proj}_\Omega(\mathbf{x}) := \operatorname{argmin}_{\mathbf{y} \in \Omega} \|\mathbf{y} - \mathbf{x}\|. \tag{5}$$

Note that $\operatorname{proj}_\Omega$ is always a singleton iff Ω is convex.

Definition 3 (Proximal map, e.g. [29])**.** *A natural generalization of the metric projection using a closed and proper function h is (with parameter $\eta > 0$):*

$$\operatorname{prox}_h^\eta(\mathbf{x}) := \operatorname{argmin}_{\mathbf{z}} h(\mathbf{z}) + \tfrac{1}{2\eta}\|\mathbf{z} - \mathbf{x}\|^2, \tag{6}$$

where $\|\cdot\|$ is the usual Euclidean norm.

If h decreases slower than a quadratic function (in particular, when h is bounded below), its proximal map is well-defined for all (small) η. For convex h, the proximal map is always a singleton while for nonconvex h, the proximal map can be set-valued. In the latter case we also abuse the notation $\operatorname{prox}_h^\eta(\mathbf{x})$ for an arbitrary element from that set. The proximal map is the key component of the popular proximal gradient algorithms [6, 11, 18].

[1] An extended real-valued function h is proper if its domain $\operatorname{dom} h := \{\mathbf{x} : h(\mathbf{x}) < \infty\}$ is nonempty; it is closed iff its sublevel sets $\{\mathbf{x} : h(\mathbf{x}) \leq \alpha\}$ is closed for all $\alpha \in \mathbb{R}$.

Definition 4 (KŁ function, [10, 20]). *A function h is called KŁ if for all $\bar{\mathbf{x}} \in \operatorname{dom} \partial h$ there exist $\lambda > 0$ and a neighborhood X of $\bar{\mathbf{x}}$ such that for all $x \in X \cap [\mathbf{x} : h(\bar{\mathbf{x}}) < h(\mathbf{x}) < h(\bar{\mathbf{x}}) + \lambda]$ the following inequality holds*

$$\varphi'\left(h(\mathbf{x}) - h(\bar{\mathbf{x}})\right) \cdot \operatorname{dist}_{\partial h(\mathbf{x})}(\mathbf{0}) \geq 1, \tag{7}$$

where the function $\varphi : [0, \lambda) \to \mathbb{R}_+, 0 \mapsto 0$, is continuous, concave, and has continuous and positive derivative φ' on $(0, \lambda)$.

The KL inequality (7) is an important tool to bound the trajectory length of a dynamical system (see [10, 20] and the references therein for some historic developments). It has recently been used to analyze discrete-time algorithms in [1] and proximal algorithms in [3, 4, 11]. As we shall see, the function φ will serve as a Lyapunov function. Quite conveniently, most practical functions, in particular, "definable" functions and convex functions under certain growth condition, are KŁ. For a more detailed discussion of KŁ functions, including many familiar examples, see [11, Section 5] and [4, Section 4].

3 Problem Formulation

We consider the composite minimization problem:

$$\min_{\mathbf{x} \in \mathbb{R}^d} F(\mathbf{x}), \quad \text{where} \quad F(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x}). \tag{P}$$

Usually f is a smooth loss function and g is a regularizer that promotes structure. We consider the **model parallel** scenario, that is, we decompose the d model parameters into p disjoint groups, and designate one worker machine for each group. Formally, consider the decomposition $\mathbb{R}^d = \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \times \cdots \times \mathbb{R}^{d_p}$, $\mathbf{x} = (x_1, x_2, \ldots, x_p)$, and let $\nabla_i f : \mathbb{R}^d \to \mathbb{R}^{d_i}$ be the partial gradient of f on the i-th factor space (machine). Clearly, $x_i, \nabla_i f(\mathbf{x}) \in \mathbb{R}^{d_i}$ and $\sum_{i=1}^p d_i = d$. The i-th machine is responsible for the i-th factor $x_i \in \mathbb{R}^{d_i}$, however, we also allow machine i to keep a local copy $\mathbf{x}^i \in \mathbb{R}^d$ of the *full* model parameter. This is for the convenience of evaluating the partial gradient $\nabla_i f : \mathbb{R}^d \to \mathbb{R}^{d_i}$, and we will discuss in Section 5 how to implement this in an economical way. Note that unlike the **data parallel** setting, we do *not* consider explicitly distributing the computation of the gradient $\nabla_i f$.

We extend the proximal gradient algorithm [11, 18] to solve the composite problem (P) under the new model parallel setting, and we require the following standard assumptions for our convergence analysis.

Assumption 1. *Regarding the functions f, g in* (P)*:*

1. *They are bounded from below;*
2. *The function f is differentiable and the gradients ∇f, $\nabla_i f$ are Lipschitz continuous with constant L_f and L_i, respectively. Set $L = \sum_{i=1}^p L_i$;*
3. *The function g is closed, and separable, i.e., $g(\mathbf{x}) = \sum_{i=1}^p g_i(x_i)$.*

The first two assumptions are needed to analyze the proximal gradient algorithm even in the convex and non-distributed setting, and the third assumption is what makes model parallelism interesting (and feasible). We remark that the differentiability assumption on f can be easily relaxed by smoothing, and the separability assumption on g can also be relaxed using the proximal average idea in [36]. For brevity we do not pursue these extensions here. One salient feature of our analysis is that we do *not* assume convexity on either f or g (although our conclusions are considerably stronger under convexity).

The separability assumption above on g implies that

$$\operatorname{prox}_g^\eta(\mathbf{x}) = \left(\operatorname{prox}_{g_1}^\eta(x_1), \ \ldots \ , \ \operatorname{prox}_{g_p}^\eta(x_p)\right). \tag{8}$$

Let us introduce the update operator (on machine i):

$$U_i(\mathbf{x}^i) = U_i(\mathbf{x}^i, x_i) := \operatorname{prox}_{g_i}^\eta(x_i - \eta \nabla_i f(\mathbf{x}^i)) - x_i, \tag{9}$$

i.e. machine i computes the i-th part of the gradient using its local model $\mathbf{x}^i$, updates its parameter x_i in charge using step size η, and finally applies the proximal map of the component function g_i. In a real large scale parallel system, the communication delay among machines and the unexpected shut down of machines are practical issues that bottlenecks the performance of the system, and hence a more relaxed synchronization protocol than full synchronization is needed. Consider a global clock shared by all machines and denote T_i the set of active clocks when machine i takes an update, and $\mathbb{I}_{\{t \in T_i\}}$ as the indicator function of the event $t \in T_i$. Formally, the t-th iteration on machine i can be written as:

$$\mathsf{msPG} \begin{cases} \forall i, \ x_i(t+1) = x_i(t) + \mathbb{I}_{\{t \in T_i\}} U_i(\mathbf{x}^i(t)), \\ \text{(local)} \quad \mathbf{x}^i(t) = \left(x_1(\tau_1^i(t)), \ \ldots \ , \ x_p(\tau_p^i(t))\right), \\ \text{(global)} \quad \mathbf{x}(t) = \left(x_1(t), \ \ldots \ , \ x_p(t)\right), \end{cases}$$

where $0 \leq \tau_j^i(t) \leq t$ models the delay among machines: when machine i conducts its t-th update it only has access to $x_j(\tau_j^i(t))$, a delayed version of the factor $x_j(t)$ on the j-th machine. In the above, we collect the fresh parameters of all machines to form a global model $\mathbf{x}(t)$, which brings convenience for our analysis. We will refer to the above updates as msPG (for **m**odel parallel, **s**tale synchronous, **P**roximal **G**radient). Figure 1 illustrates the main idea of msPG.

Obviously, to establish convergence for msPG we need some control over the delay $\tau_j^i(t)$ and the active

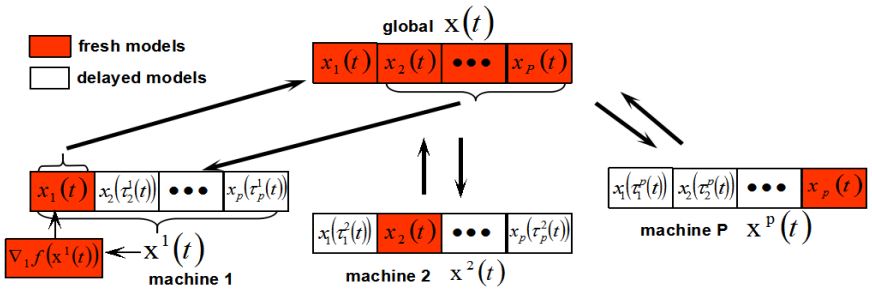

Figure 1: The algorithm msPG under model parallelism and stale synchronism. Machine i keeps a local model $\mathbf{x}^i(t)$ that contains stale parameters of other machines (due to communication delay and network latency). These local models are used to compute the partial gradient $\nabla_i f(\mathbf{x}^i)$ which is then used to update the parameters $x_i(t)$ in each machine. See Section 5 for an economical implementation of msPG.

clocks T_i, for otherwise some machines may not make progress at all. We first impose the following assumptions on the delay and active clocks, and follow by some explanations.

Assumption 2. *The delay and skip frequency satisfy:*

1. $\forall i, \forall j, \forall t,\ 0 \le t - \tau_j^i(t) \le s$;
2. $\forall i, \forall t,\ \tau_i^i(t) = t$;
3. $\forall i, \forall t,\ T_i \cap \{t, t+1, \cdots, t+s\} \neq \emptyset$.

The first assumption basically says that if machine i conducts its t-th update, then the information it gathered from other machines cannot be too obsolete (bounded by s iterations). The second assumption simply says that machine i always has the latest information on itself. The third assumption requires each machine to update at least once in every s iterations. These assumptions are very natural and have been widely adopted in previous works [2, 8, 16, 19, 22, 24, 32]. They are also in some sense unavoidable: one can construct instances such that msPG do not converge if these assumptions are violated. Clearly, when $s = 0$ (no delay) our framework reduces to the bulk synchronous proximal gradient algorithm.

4 Convergence Analysis

In this section, we conduct detailed analysis of the model parallel stale synchronous proximal gradient algorithm msPG. Our first result is as follows:

Theorem 1 (Asymptotic consistency)**.** *Let Assumption 1 and 2 hold, and apply msPG to problem* (P)*. If the step size $\eta < (L_f + 2Ls)^{-1}$, then the global model and local models satisfy:*

1. $\sum_{t=0}^{\infty} \|\mathbf{x}(t+1) - \mathbf{x}(t)\|^2 < \infty$;
2. $\lim_{t\to\infty} \|\mathbf{x}(t+1) - \mathbf{x}(t)\| = 0$, $\lim_{t\to\infty} \|\mathbf{x}(t) - \mathbf{x}^i(t)\| = 0$;
3. *The limit points* $\omega(\{\mathbf{x}(t)\}) = \omega(\{\mathbf{x}^i(t)\}) \subseteq \operatorname{crit} F$.

The proof is a bit involved and can be found in Appendix A. The first assertion says that the global sequence $\mathbf{x}(t)$ defined in msPG has square summable successive differences, which we will significantly strengthen below. The second assertion implies that the successive difference of the global sequence diminishes, and the inconsistency between the local sequences and the global sequence also vanishes. These two conclusions provide some stability guarantee about our algorithm msPG. The third assertion further justifies msPG by showing that any limit point it produces is necessarily a critical point. Of course, when F is convex, any critical point is globally optimal.

The closest result to Theorem 1 we are aware of is [8, Proposition 7.5.3], where essentially the same conclusion was reached but under the much more simplified assumption that g is an indicator function of a *convex* set. Thus, our Theorem 1 is new even when g is a convex function such as the popular ℓ_1 norm. Furthermore, we allow g to be nonconvex, which is particularly interesting due to the rising interest in nonconvex penalties in machine learning and statistics (see e.g. [15, 26, 31, 35, 36, 38–40]). We also note that the proof of Theorem 1 (for nonconvex g) involves significantly new ideas beyond those of [8].

Interesting as it is, Theorem 1 has one significant deficiency, though: it does not tell us when there exists a limit point, and it does not guarantee the whole sequence to converge to the limit point. In fact, in the model parallel setting with delays and skips, it is even a nontrivial task to argue that the objective values $\{F(\mathbf{x}(t))\}$ do not diverge to infinity. This is in sharp contrast with the bulk synchronous setting where it is trivial to guarantee the objective values to decrease (by using a sufficiently small step size). This is where we need some further assumptions.

Assumption 3 (Sufficient Decrease)**.** *There exists $\alpha > 0$ such that the global model $\mathbf{x}(t)$ generated by*

msPG (for problem (P)*) satisfies: for all large t,*

$$F(\mathbf{x}(t+1)) \leq F(\mathbf{x}(t)) - \alpha\|\mathbf{x}(t+1)-\mathbf{x}(t)\|^2. \quad (10)$$

We will provide below a sufficient condition in Lemma 1 and many interesting examples in Example 1 to ensure Assumption 3 to hold.

Assumption 4. *F is a KŁ function.*

As we mentioned in the end of Section 2, most practical functions (including all functions in this work) are KŁ. Hence, this is a very mild assumption.

We are now ready to state the much more strengthened convergence guarantee for msPG:

Theorem 2 (Finite Length)**.** *Let Assumption 1, 2, 3 and 4 hold, and apply msPG to problem* (P)*. If the step size $\eta < (L_f + 2Ls)^{-1}$ and $\{\mathbf{x}(t)\}$ is bounded, then*

$$\sum_{t=0}^{\infty} \|\mathbf{x}(t+1)-\mathbf{x}(t)\| < \infty, \quad (11)$$
$$\forall i = 1,\dots,p,\ \sum_{t=0}^{\infty} \|\mathbf{x}^i(t+1)-\mathbf{x}^i(t)\| < \infty. \quad (12)$$

Furthermore, $\{\mathbf{x}(t)\}$ and $\{\mathbf{x}^i(t)\}, i = 1,...,p$, converge to the same critical point of F.

The proof, in Appendix B, borrowed some ideas from the recent works [3, 4, 11], which, however, did not consider the parallel and asynchronous setting. Compared with the first assertion in Theorem 1, we now have the successive differences to be absolutely summable (instead of square summable). The former property is usually called finite length in dynamical systems. It is a significantly stronger property as it immediately implies that the whole sequence is Cauchy hence convergent, whereas we cannot get the same conclusion from the square summable property in Theorem 1.[2] We note that local maxima are excluded from being the limit in Theorem 2, thanks to Assumption 3. Also, the boundedness assumption on $\{\mathbf{x}(t)\}$ is easy to satisfy, for instance, when F has bounded sublevel sets. We refer to [4, Remark 3.3] for more conditions that guarantee the boundedness. Needless to say, if F is convex, then the whole sequence in Theorem 2 converges to a global minimizer.

The closest result to Theorem 2, to our best knowledge, is [32], which proved an s-step linear rate when f satisfies a certain error bound condition and g is the indicator function of a convex set. In contrast, we allow any convex or nonconvex function g (as long as it is KŁ). Our proof, inspired by [3, 4, 11], differs substantially from [32].

[2] A simple example would be the sequence $x(t) = \sum_{k=1}^{t} \frac{1}{k}$, whose successive difference is square summable but clearly $x(t)$ does not converge. Consequently, $x(t)$ is not absolutely summable.

We now provide some justifications on Assumption 3 by considering the simplified case where $\forall t, i, t \in T_i$, i.e., machines do not skip updates. The general case can also be dealt with but the analysis is much more complicated. The following condition truns out to be a good justification for Assumption 3.

Assumption 5 (Eventual Lipschitz)**.** *The update operators $U_i, i = 1,...,p$ are eventually Lipschitz continuous, i.e., for all large t and small learning rate $\eta > 0$:*

$$\|U_i(\mathbf{x}^i(t+1)) - U_i(\mathbf{x}^i(t))\| \leq C_i\eta\|\mathbf{x}^i(t+1)-\mathbf{x}^i(t)\|, \quad (13)$$

where $C_i \geq L_i, i = 1,...,p$, are positive constants.

Note that when $g \equiv 0$, $U_i = -\eta\nabla_i f$ is Lipschitz continuous (due to Assumption 1.2), thus Assumption 5 is a natural generalization to an arbitrary regularizer g. Equipped with this assumption, we can now justify Assumption 3. (Proof is in Appendix C.) In the sequel, we denote $C = \sum_{i=1}^{p} C_i \geq L$.

Lemma 1. *Assume $\forall t, i, t \in T_i$. Let the step size $\eta < \frac{\rho-1}{4C\rho}\frac{\sqrt{\rho}-1}{\sqrt{\rho}^{s+1}-1}$ for any $\rho > 1$ and all $U_i, i = 1,...,p$ be eventually Lipschitz continuous, then the sequences $\{\mathbf{x}(t)\}$ and $\{\mathbf{x}^i(t)\}, i = 1,...,p$, have finite length.*

Hence it is sufficient to further characterize Assumption 5, which turns out to be a mild condition. Instead of giving a very technical justification, we give here some popular examples where Assumption 5 holds (proof in Appendix D). Some of these will also be tested in our experiments.

Example 1. *Assume $\forall t, i, t \in T_i$, then Assumption 5 holds for the following cases (modulo a technical condition on the 1-norm):*

- *$g \equiv 0$ (no regularization), $g = \|\cdot\|_0$ (nonconvex 0-norm), $g = \|\cdot\|_1$ (1-norm), $g = \|\cdot\|^2$ (squared 2-norm);*
- *elastic net $g = \|\cdot\|_1 + \|\cdot\|^2$ and its nonconvex variation $g = \|\cdot\|_0 + \|\cdot\|^2$;*
- *non-overlapping group norms $g = \|\cdot\|_{0,2}$ and $g = \|\cdot\|_{0,2} + \|\cdot\|^2$.*

Further for this non-skip case ($\forall t, i, t \in T_i$), msPG can be cast as an inexact proximal gradient algorithm (IPGA), which, together with the finite length property in Theorem 2, provide new insights on the nature of staleness in real parallel systems. It also allows us to easily obtain the usual $O(1/t)$ rate of convergence of the objective value.

Let us introduce an error term $\mathbf{e}(t) = \big(e_1(t), \dots, e_p(t)\big)$, with which we can rewrite the global sequence of msPG as:

$$\forall i,\ x_i(t+1) = \mathrm{prox}_{g_i}^{\eta}\left(x_i(t) - \eta\nabla_i f(\mathbf{x}(t)) + e_i(t)\right), \quad (14)$$

where $e_i(t) = \eta[\nabla_i f(\mathbf{x}(t)) - \nabla_i f(\mathbf{x}^i(t))]$. This alternative representation of msPG falls under the IPGA in [30], where $\mathbf{e}(t)$ is the (gradient) error at iteration t. Note, however, that the error $\mathbf{e}(t)$ is not caused by computation but by communication delay and network latency, which only presents itself in a real stale synchronous parallel system. For convex functions F, [30] showed that the convergence rate of the objective value of IPGA can be controlled by the summability of the error magnitude $\|\mathbf{e}(t)\|$. Interestingly, our next result proves that the finite length property in Theorem 2 immediately implies the summability of the errors $\|\mathbf{e}\|$, even for nonconvex functions f and g. Moreover, for convex F, this also leads to the usual $O(1/t)$ rate of convergence in terms of the objective value.

Theorem 3 (Global rate of convergence)**.** *If the finite length property in Theorem 2 holds, then*

1. $\sum_{t=0}^{\infty} \|\mathbf{e}(t)\| < \infty$;
2. $F(\frac{1}{t}\sum_{k=1}^{t} \mathbf{x}(k)) - \inf F \leq O\left(t^{-1}\right)$.

The proof is in Appendix E. Intuitively, if the error $\mathbf{e}(t)$ decreases (slightly) faster than $O(1/t)$, then the rate of convergence of msPG is not affected even under the model parallel and stale synchronous setting (provided F is convex). To the best of our knowledge, this is the first *deterministic* rate of convergence result in the model parallel and stale synchronous setting.

5 Economical Implementation

In this section, we show how to economically implement msPG for the widely used linear models:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(A\mathbf{x}) + g(\mathbf{x}), \tag{15}$$

where $A \in \mathbb{R}^{n\times d}$. Typically $f : \mathbb{R}^n \to \mathbb{R}$ is the likelihood function and $g : \mathbb{R}^d \to \mathbb{R}$ is the regularizer (we absorb the regularization constant into g). Each row of A corresponds to a sample point and we have suppressed the labels in classification or responses in regression. Support vector machines, Lasso, logistic regression, boosting, etc. all fit under this framework. Our interest here is when the model dimension d is much higher than the number of samples n (d can be up to hundreds of millions and n can be up to millions). This is the usual setup in many computational biology and health care problems.

A naive implementation of msPG might be inefficient in terms of both network communication and parameter storage. First, Each machine needs to communicate with every other machine, to exchange the latest block of parameters. If using a peer-to-peer network topology, the resulting connections will be too dense and crowded when the system holds hundreds of machines. We resolve this issue by adopting the parameter server system advocated in previous works [19, 21], that is, we dedicate a specific server (which can span a set of machines if needed) to store the key parameters (will be specified later) and let each worker machine to communicate only with the server. There is a second advantage for this master-slave network topology, as we shall see momentarily.

Second, each machine needs to keep a local copy of the full model (i.e. $\mathbf{x}^i(t)$ in msPG), which can incur a very expensive storage cost when the dimension is high. This is where the linear model structure in (15) comes into help. Note that the local models $\mathbf{x}^i(t)$ are kept solely for the convenience of evaluating the partial gradient $\nabla_i f : \mathbb{R}^d \to \mathbb{R}^{d_i}$. For some problems such as the Lasso, a seemingly workaround is to pre-compute the Hessian $H = A^\top A$ and distribute the corresponding row blocks of H to each worker machine. This scheme, however, is problematic in the high dimensional setting: the pre-computation of the Hessian can be very costly, and each row block of H has a very large size $(d_i \times d)$.

Instead, we use the column partition scheme [e.g. 12, 28], namely, we partition the matrix A into p *column* blocks $A = [A_1, \ldots, A_p]$ and distribute the block $A_i \in \mathbb{R}^{n\times d_i}$ to machine i. Now the local update computed by machine i at the t-th iteration can be rewritten as

$$U_i(\mathbf{x}^i(t)) = \mathrm{prox}_{g_i}^{\eta}\left(x_i(t) - \eta A_i^\top f'(A\mathbf{x}^i(t))\right) - x_i(t) \tag{16}$$

Since machine i is in charge of updating the i-th block $x_i(t)$ of the global model, to compute the local update (16) it is sufficient to have the matrix-vector product $A\mathbf{x}^i(t)$. For simplicity we initialize $\forall i,\ \mathbf{x}^i(0) \equiv \mathbf{0}$, then we have the following cumulative form:

$$A\mathbf{x}^i(t) = \sum_{j=1}^{p} A_j[\mathbf{x}^i(t)]_j = \sum_{j=1}^{p}\sum_{k=0}^{\tau_j^i(t)} \underbrace{A_j \mathbb{I}_{\{k\in T_j\}} U_j(\mathbf{x}^j(k))}_{\Delta_j(k)},$$

where recall that when machine i conducts its t-th iteration it only has access to a delayed copy $x_j(\tau_j^i(t))$ of the parameters in machine j. Since this matrix-vector product is needed by every machine to conduct their local updates in (16), we aggregate $\Delta_j(t) \in \mathbb{R}^n$ on the parameter server whenever it is generated and sent by the worker machines. In details, the worker machines first pull this aggregated matrix-vector product (denoted as ▲) from the server to conduct the local computation (16) in an economical way (by replacing $A\mathbf{x}^i(t)$ in (16) with ▲). Then machine i performs the simple update:

$$x_i(t+1) = x_i(t) + U_i(\mathbf{x}^i(t)). \tag{17}$$

Note that machine i does not maintain or update other blocks of parameters $x_j(t), j \neq i$. Lastly, machine i

Algorithm 1 Economic Implementation of msPG

For the server:
  while recieves update Δ_i from machine i **do**
    $\blacktriangle \leftarrow \blacktriangle + \Delta_i$
  end while
  while machine i sends a pull request **do**
    send $\blacktriangle$ to machine i
  end while
For machine i at active clock $t \in T_i$:
  pull $\blacktriangle$ from the server
  $U_i \leftarrow \mathrm{prox}_{g_i}^{\eta}\big(x_i - \eta A_i^\top f'(\blacktriangle)\big) - x_i$
  send $\Delta_i = A_i U_i$ to the server
  update $x_i \leftarrow x_i + U_i$

computes and sends the vector $\Delta_i(t) = A_i U_i(\mathbf{x}^i(t)) \in \mathbb{R}^n$ to the server, and the server immediately performs the aggregation:

$$\blacktriangle \leftarrow \blacktriangle + \Delta_i(t). \tag{18}$$

We summarize the above economical implementation in Algorithm 1, where $\blacktriangle$ denotes the aggregation of individual matrix-vector products Δ on the server. The storage cost for each worker machine is $O(nd_i)$ (for storing A_i). Each iteration requires two matrix-vector products that cost $O(nd_i)$ in the dense case, and the communication of a length n vector between the server and the worker machines.

6 Experiments

We first test the convergence properties of msPG via a non-convex Lasso problem with the group regularizer $\|\cdot\|_{0,2}$, which takes the form

$$\min_{\mathbf{x}} \ \tfrac{1}{2}\|A\mathbf{x} - \mathbf{b}\|^2 + \lambda \textstyle\sum_{i=1}^{20} \mathbb{I}(\|x_i\| = 0), \tag{19}$$

where $A \in \mathbb{R}^{1000\times 2000}$ and we refer to Appendix F for the specifications of data generation. We use 4 machines (cores) with each handling five groups of coordinates, and consider staleness $s = 0, 10, 20, 30$, respectively. To better demonstrate the effect of staleness, we let machines only communicate when exceed the maximum staleness. This can be viewed as the worst case communication scheme and a larger s brings more staleness into the system. We set the learning rate to have the form $\eta(\alpha s) = 1/(L_f + 2L\alpha s), \alpha > 0$, that is, a linear dependency on staleness s as suggested by Theorem 1. Then we run Algorithm 1 with different staleness and use $\eta(0), \eta(10), \eta^*(\alpha s)$, respectively, where $\eta^*(\alpha s)$ is the largest step size we tuned for each s that achieves a stable convergence. We track the global model $\mathbf{x}(t)$ and plot the results in Figure 2. Note that with the large step size $\eta(0)$ all instances (with nonzero staleness) diverge hence are not presented. With $\eta(10)$ (Figure 2, left), the staleness does not substantially affect the convergence in terms of the objective value. We note that the objective curves converge to slightly different minimal values due to the non-convexity of problem (19). With $\eta^*(\alpha s)$ (Figure 2, middle), it can be observed that adding a slight penalty αs on the learning rate suffices to achieve a stable convergence, and the penalty grows as s increases, which is intuitive since a larger staleness requires a smaller step size to cancel the inconsistency. In particular, for $s = 10$ the best convergence is comparable to the bulk synchronized case $s = 0$. (Figure 2, right) further shows the asymptotic convergence behavior of the global model $\mathbf{x}(t)$ under the step size $\eta^*(\alpha s)$. It is clear that a linear convergence is eventually attained, which confirms the finite length property in Theorem 2.

Next, we verify the time and communication efficiency of msPG via an l_1 norm Lasso problem with very high dimensions, taking the form

$$\min_{\mathbf{x}} \ \tfrac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_1. \tag{20}$$

We generate $A \in \mathbb{R}^{n\times d}$ of size $n = 1$Million and $d = 100$Millions. See Appendix F for the specifications of data generation. We implement Algorithm 1 on Petuum [13, 19] — a stale synchronous parallel system which eagerly updates the local parameter caches via stale synchronous communications. The system contains 100 computing nodes and each is equipped with 16 AMD Opteron processors and 16GB RAM linked by 1Gbps ethernet. We fix the learning rate $\eta = 10^{-3}$ and consider maximum staleness $s = 0, 1, 3, 5, 7$, respectively. (Figure 3, left) shows that per-iteration progress is virtually indistinguishable among various staleness settings, which is consistent with our previous experiment. (Figure 3, middle) shows that system throughput is significantly higher when we introduce staleness. This is due to lower synchronization overheads, which offsets any potential loss due to staleness in progress per iteration. We also track the distributions of staleness during the experiments, where we record in $\blacktriangle$ the clocks of the freshest updates that accumulate from all the machines. Then whenever a machine pulls $\blacktriangle$ from the server, it compares its local clock with these clocks and records the clock differences. (Figure 3, right) shows the distributions of staleness under different maximal staleness settings. Observe that bulk synchronous ($s = 0$) peaks at staleness 0 by design, and the distribution concentrates in small staleness area due to the eager communication mechanism of Petuum. It can be seen that a small amount of staleness is sufficient to relax the communication bottlenecks without affecting the iterative convergence rate much.

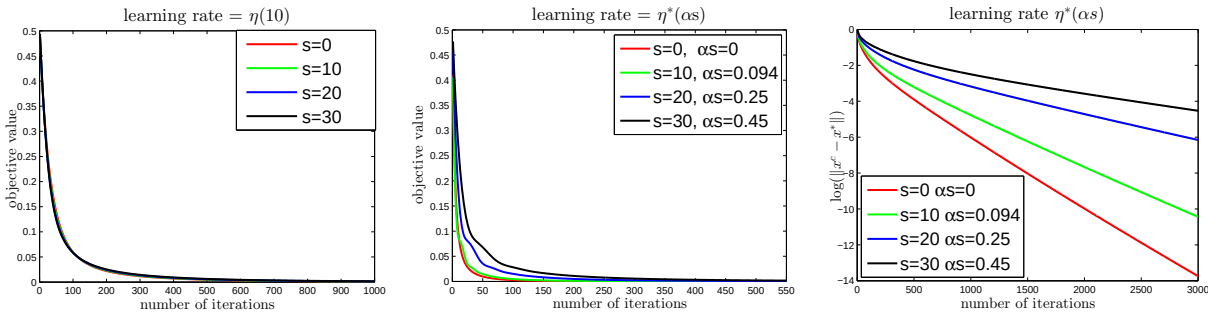

Figure 2: Convergence curves of msPG under different staleness parameter s and step size η.

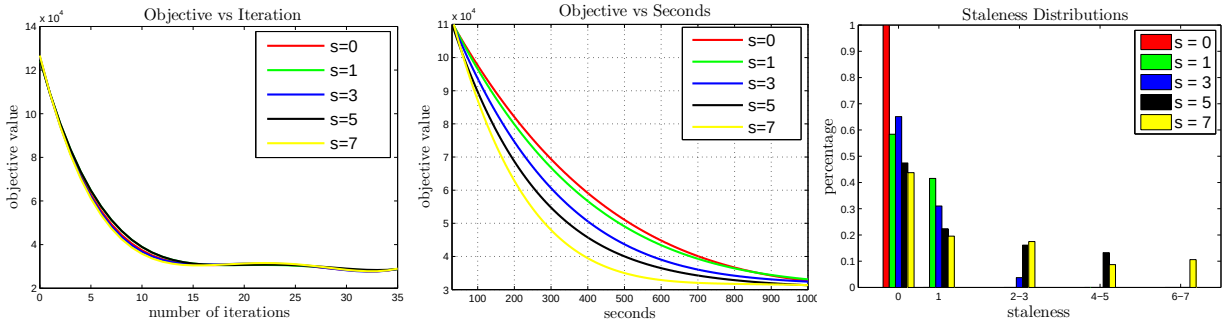

Figure 3: Efficiency of msPG on a large scale Lasso problem.

7 Related Work

The stale synchronous parallel system dates back to [7, 8, 32, 33], where it is also refered to as partially asyncrhonous system. These work consider using stale synchronous systems to solving different kinds of optimizations problems with allowing machines to skip updates during the process. Asymptotic convergence of partially asynchronous gradient descent algorithm for solving unconstraint smooth optimizations is established in [8], with its stochastic version being analyzed in [33]. Asymptotic convergence of partially asynchronous gradient projection algorithm for solving smooth optimizations with convex constraint is established in [8], and a "B-step" linear convergence is further established in [32] with an error bound condition. Linear convergence of partially asynchronous algorithm for finding the fixed point of maximum norm contraction mappings is established in [7, 17].

Another series of work focus on SSP systems where machines are not allowed to skip updates [21–23]. In their settings, The system imposes an upper bound on the maximum clock difference between machines. Asymptotic convergence is established for proximal gradient algorithm for data parallelism [23] and for block coordinate descent [22] with a smooth objective and convex regularizer. Other works consider stochastic algorithms on stale synchronous system. [19] proposes an SSP system for stochastic gradient descent, and establishes $O(1/\sqrt{k})$ regret bound under bounded diameter and bounded sub-gradient assumption. [16, 27] consider a delayed stochastic gradient descent algorithm. Linear convergence to a neighborhood of optimum is established with strong convexity assumption in [16] and with additional bounded gradient assumption in [27]. [2] proposes a distributed delayed dual averaging and mirror descent algorithm, and establishes $O(1/\sqrt{k})$ regret bound under standard stochastic assumptions.

8 Conclusion

We have proposed msPG as an extension of the proximal gradient algorithm to the model parallel and stale synchronous setting. msPG allows worker machines to operate asynchronously as long as they are not too far apart, hence greatly improves the system throughput. Theoretically, we provide a rigorous analysis of msPG that simultaneously covers nonsmooth and nonconvex functions. In particular, under mild conditions, the whole iterate sequence generated by msPG converges to a critical point. We implement msPG using the parameter server platform, and completely bypass the need of keeping a local full model. Preliminary numerical experiments confirm the effectiveness of msPG on solving very high dimensional problems.

Acknowledgement

We thank the reviewers for their valuable comments. This work is supported by NIH R01GM114311, DARPA FA87501220324 and NSF IIS1447676.

References

[1] P.-A. Absil, R. Mahony, and B. Andrews. Convergence of the iterates of descent methods for analytic cost functions. *SIAM Journal on Optimization*, 16(2):531–547, 2005.

[2] A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems 24*, pages 873–881. 2011.

[3] H. Attouch and J. Bolte. On the convergence of the proximal algorithm for nonsmooth functions involving analytic features. *Mathematical Programming*, 116(1-2):5–16, 2009.

[4] H. Attouch, J. Bolte, P. Redont, and A. Soubeyran. Proximal alternating minimization and projection methods for nonconvex problems: An approach based on the Kurdyka-Łojasiewicz inequality. *Mathematics of Operations Research*, 35(2):438–457, 2010.

[5] H. Attouch, J. Bolte, and B. Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward-backward splitting, and regularized Gauss-Seidel methods. *Mathematical Programming*, 137(1-2):91–129, 2013.

[6] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202, 2009.

[7] D. P. Bertsekas and J. N. Tsitsiklis. Convergence rate and termination of asynchronous iterative algorithms. In *Proceedings of the 3rd International Conference on Supercomputing*, pages 461–470, 1989.

[8] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[9] J. Bolte, A. Daniilidis, and A. Lewis. The Łojasiewicz inequality for nonsmooth subanalytic functions with applications to subgradient dynamical systems. *SIAM Journal on Optimization*, 17:1205–1223, 2007.

[10] J. Bolte, A. Danilidis, O. Ley, and L. Mazet. Characterizations of Łojasiewicz inequalities and applications: Subgradient flows, talweg, convexity. *Transactions of the American Mathematical Society*, 362(6):3319–3363, 2010.

[11] J. Bolte, S. Sabach, and M. Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, 146(1-2):459–494, 2014.

[12] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2010.

[13] W. Dai, A. Kumar, J. Wei, Q. Ho, G. Gibson, and E. P. Xing. High-performance distributed ml at scale through parameterserver consistency models. In *AAAI*, 2014.

[14] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of ACM*, 51(1):107–113, 2008.

[15] J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

[16] H. Feyzmahdavian, A. Aytekin, and M. Johansson. A delayed proximal gradient method with linear convergence rate. In *2014 IEEE International Workshop on Machine Learning for Signal Processing*.

[17] H. Feyzmahdavian and M. Johansson. On the convergence rates of asynchronous iterations. In *2014 IEEE 53rd Annual Conference on Decision and Control*, pages 153–159, 2014.

[18] M. Fukushima and H. Mine. A generalized proximal point algorithm for certain non-convex minimization problems. *International Journal of Systems Science*, 12(8):989–1000, 1981.

[19] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems 26*, pages 1223–1231. 2013.

[20] K. Kurdyka. On gradients of functions definable in o-minimal structures. *Annales de l'institut Fourier*, 48(3):769–783, 1998.

[21] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, 2014.

[22] M. Li, D. G. Andersen, and A. Smola. Distributed delayed proximal gradient methods. *Big Learning NIPS Workshop*, 2013.

[23] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola. Parameter server for distributed machine learning. *Big Learning NIPS Workshop*, 2013.

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

[25] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, 2012.

[26] R. Mazumder, J. H. Friedman, and T. Hastie. Sparsenet: Coordinate descent with nonconvex penalties. *Journal of the American Statistical Association*, 106(495):1125–1138, 2011.

[27] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*, pages 693–701. 2011.

[28] P. Richtárik and M. Takáč. Distributed Coordinate Descent Method for Learning with Big Data. *arXiv*, 2013.

[29] R. Rockafellar and R. Wets. *Variational Analysis*. Springer, 1997.

[30] M. Schmidt, N. L. Roux, and F. R. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In *Advances in Neural Information Processing Systems 24*, pages 1458–1466. 2011.

[31] Y. She. Thresholding-based iterative selection procedures for model selection and shrinkage. *Electronic Journal of Statistics*, 3:384–415, 2009.

[32] P. Tseng. On the rate of convergence of a partially asynchronous gradient projection algorithm. *SIAM Journal on Optimization*, 1(4):603–619, 1991.

[33] J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.

[34] L. G. Valiant. A bridging model for parallel computation. *Communications of ACM*, 33(8):103–111, 1990.

[35] Z. Wang, H. Liu, and T. Zhang. Optimal computational and statistical rates of convergence for sparse nonconvex learning problems. *The Annals of Statistics*, 42(6):2164–2201, 2014.

[36] Y. Yu, X. Zheng, M. Marchetti-Bowick, and E. P. Xing. Minimizing nonconvex non-separable functions. In *The* 17^{th} *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[37] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. pages 10–10, 2010.

[38] C.-H. Zhang. Nearly unbaised variable selection under minimax concave penalty. *Annals of Statistics*, 38(2):894–942, 2010.

[39] C.-H. Zhang and T. Zhang. A general theory of concave regularization for high-dimensional sparse estimation problems. *Statistical Science*, 27(4):576–593, 2012.

[40] Y. Zhu, X. Shen, and W. Pan. Simultaneous grouping pursuit and feature selection over an undirected graph. *Journal of the American Statistical Association*, 108(502):713–725, 2013.