A Delay-tolerant Proximal-Gradient Algorithm for Distributed Learning

Konstantin Mishchenko[1] Franck Iutzeler[2] Jérôme Malick[3] Massih-Reza Amini[2]

Abstract

Distributed learning aims at computing high-quality models by training over scattered data. This covers a diversity of scenarios, including computer clusters or mobile agents. One of the main challenges is then to deal with heterogeneous machines and unreliable communications. In this setting, we propose and analyze a flexible asynchronous optimization algorithm for solving nonsmooth learning problems. Unlike most existing methods, our algorithm is adjustable to various levels of communication costs, machines computational powers, and data distribution evenness. We prove that the algorithm converges linearly with a fixed learning rate that does not depend on communication delays nor on the number of machines. Although long delays in communication may slow down performance, no delay can break convergence.

1. Introduction

Distributed learning set-up We consider a general learning problem formulated as the following nonsmooth optimization problem

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

where n is the size of the training set, the ℓ_i's are smooth convex empirical loss functions and r is a non-smooth convex regularizer. This template models a broad range of problems arising in machine learning and signal processing: finite-sum structure of the data-fidelity term includes the least-squares or logistic loss functions; the regularization term includes penalties such as ℓ_1 or group lasso.

[1]King Abdullah University of Science & Technology (KAUST) [2]Univ. Grenoble Alpes [3]CNRS and LJK. Correspondence to: Franck Iutzeler <franck.iutzeler@univ-grenoble-alpes.fr>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

In our distributed setting, the n data examples are split across M machines, each machine i having a private subset $\mathcal{S}_i$ of the examples. Problem (1) then reformulates as

$$\min_{x\in\mathbb{R}^d} \sum_{i=1}^{M} \underbrace{\left(\frac{n_i}{n}\right)}_{\substack{:=\pi_i\\ \text{proportion}}} \underbrace{\left(\frac{1}{n_i}\sum_{j\in\mathcal{S}_i} \ell_j(x)\right)}_{\substack{:=f_i(x)\\ \text{local empirical loss}}} + r(x) \tag{2}$$

where $n_i = |\mathcal{S}_i|$ the cardinality of the local data subset, π_i the proportion of data locally stored, and f_i the local empirical loss at machine i. This type of problem arises when one wants to learn over distributed devices, each of which having a local part of the data (this locality coming from the prohibitive size of the data, or its privacy), as in federated learning (Konečný et al., 2016).

The communication constraints of the distributed setting naturally lead us to consider a master-slave framework. More precisely, we present an optimization algorithm where slave machines compute (deterministic) updates based on their local data and asynchronously communicate their result to a master machine where the updates are aggregated to produce a new iterate. To cope up with the diversity of the scenarios included in this framework (e.g. computing clusters, mobile devices), we pay a special attention to have realistic assumptions on communication delays and data distribution.

Contributions We propose a proximal-gradient optimization algorithm for distributed learning with *flexible* asynchronous communications. This algorithm allows every slave to perform an arbitrary number of passes on his local data, thus adjusting the computation/communication trade-off. We show that the algorithm converges linearly, in the strongly convex case, with a rate independent of the computing system, which is highly desirable for scalability. To this end, we develop a new epoch-based mathematical analysis, encompassing computation times and communication delays, to refocus the theory on algorithmics.

Ouline of the paper The paper is structured as follows. In Section 2, we review the related literature and their limitations in our context. Then we present in Section 3 the new asynchronous distributed algorithm, denoted as `DAve-RPG` for Distributed Averaging of Repeated Proximal Gradient steps. In Section 4, we provide local and global convergence analyses in the strongly convex case by showing that the algorithm converges linearly with a rate and with a stepsize, both independent from machines and delays. Finally, we provide and discuss numerical results in section 5 based on typical large-scale problems.

2. Related work

Parallel Stochastic algorithms Stochastic algorithms have received a lot of attention, regarding convergence rates, acceleration, parallelization, generalization to non-smooth or sparse gradient cases; see e.g. (Shalev-Shwartz & Zhang, 2013; Johnson & Zhang, 2013; Defazio et al., 2014). Parrallel versions of stochastic algorithms have also been proposed where subparts of the data are stored in different machines (Hogwild! (Recht et al., 2011), Distributed SDCA (Takáč et al., 2015), Distributed SVRG (Lee et al., 2015; Zhao & Li, 2016), ASAGA (Leblond et al., 2017), ProxASAGA (Pedregosa et al., 2017)).

Despite their theoretical properties and practical success in the context of multicore computers, these algorithms are not well suited for our distributed setting. For example, ASAGA (Leblond et al., 2017; Pedregosa et al., 2017) makes computation in parallel but is not *fully* distributed as it assumes uniform sampling with shared memory between computing parties. Thus, a naive extension of such parallel stochastic methods would be inefficient in practice due to large overheads in communications. For this reason, it is commonly admitted (see e.g. (Ma et al., 2017b;a)) that in a distributed computing environment, one must focus not only on the number of data accesses, but also on the number of communication steps, which rehabilitates batch algorithms. In light of this remark, our batch approach allows for several local iterations which gives a way to adjust the trade-off between communication and computation.

Distributed algorithms There exists a rich literature on distributed optimization algorithms with no shared memory. We mention e.g. ARock (Peng et al., 2016), Asynchronous ADMM (Zhang & Kwok, 2014), COCOA (Ma et al., 2015), Delayed Proximal Gradient algorithms (Feyzmahdavian et al., 2014; Aytekin et al., 2016; Vanli et al., 2016), or dSAGA (Calauzènes & Roux, 2017). These methods often have strong assumptions about synchrony of communications, or boundedness of the delays between fastest and slowest machines. For instance, the asynchronous distributed ADMM of (Zhang & Kwok, 2014) allows asynchronous updates only until a maximal delay, after which every worker has to wait for the slowest one.

Usually, the bounds on delays also impact the learning rates as in (Peng et al., 2016) and/or the convergence rates as in (Feyzmahdavian et al., 2014). Except (Sun et al., 2017) in a stochastic context, the assumptions on delays of the above algorithms are very restrictive. In contrast, we propose an asynchronous algorithm with no assumption on delays in the sense that, while they may slow down the convergence, the stepsize of our algorithm and its convergence are independent of the delays. Moreover, the convergence rate is expressed along a suitable epoch sequence which makes it adapted to varying delays or crashes.

3. `DAve-RPG`: Distributed Averaging of Repeat Proximal Gradient steps

This section presents our distributed proximal-gradient algorithm, that we call `DAve-RPG`, where `DAve` stands for the global communication scheme based on distributed averaging of iterates, and `RPG` stands for the local optimization scheme, based on repeated proximal-gradient steps.

3.1. `DAve` Communication scheme

Recall our distributed learning problem (2) where data subsets are stored privately on M different machines

$$\min_{x\in\mathbb{R}^d} \sum_{i=1}^{M} \pi_i f_i(x) + r(x)$$

with π_i being the proportion n_i/n of data locally stored at machine i, and $f_i(x) = \frac{1}{n_i}\sum_{j\in\mathcal{S}_i} \ell_j(x)$ the local empirical risk[1] at machine i. Each machine makes local computations in view of solving globally the above problem.

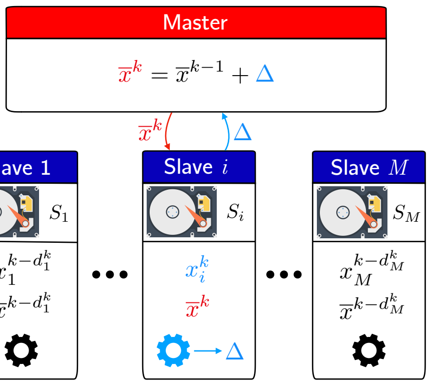

Figure 1: Schematic view on the asynchronous algorithm

Our communication scheme `DAve` is based on a distributed averaging of the parameters of the machines done by a master machine in a fully asynchronous manner. More precisely, as soon as a slave machine, say i, finishes the computation of a new *local parameter* x_i and the corresponding *adjustment* Δ, defined later, it sends this adjustment to the

[1]Note that, unlike in many stochastic optimization settings, each function f_i correspond to arbitrary number of data points. This asymmetry is automatically handled by the algorithm.

master node that adds it to its *master parameter* $\overline{x}$. The master immediately sends back this parameter to machine i which begins a new computation step. This scheme is thus inherently fully asynchronous as the actions and communications of the machines are triggered by the reception of new information. Notably, the slaves machines compute updates restlessly without idle times.

A key point of this scheme is that the master parameter always represents the average of the machines local parameters, weighted with their proportion of data $\overline{x} = \sum_{i=1}^{M} \pi_i x_i$. This property is forced by assigning the difference between new and previous values of x_i with coefficient π_i. At the same time, due to asynchrony, the local parameters are not computed from the same version of the master parameter which calls for a proper time index and delays notations.

In terms of information storage, slave i only has its own subset of data $\mathcal{S}_i$, the last received master parameter, its last computed local parameter, and potentially local variables; the master machine only stores its master parameter.

Asynchrony and Delays. We call iteration/time k (denoted by a superscript k) the k-th time the master has updated $\overline{x}$. The corresponding master parameter, $\overline{x}^k$, is equal to the average of the received parameter of the agents (x_i^k), where x_i^k is the latest parameter received from slave i before time k.

For a slave i and a time k, we denote by d_i^k the delay relative to i and k, i.e. the number of master updates, since slave i sent information to the master. For instance, if slave i updates at time k, then $d_i^k = 0$; we thus have $x_i^k = x_i^{k-d_i^k}$ and

$$\overline{x}^k = \sum_{i=1}^{M} \pi_i x_i^k = \sum_{i=1}^{M} \pi_i x_i^{k-d_i^k}. \tag{3}$$

In addition, we denote by D_i^k the delay from the penultimate update, so that $D_i^k = d_i^k + d_i^{k-d_i^k-1} + 1$. These definitions are illustrated in Fig. 2.

Intuitively, our algorithm holds at the master the weighted average of *each* agent *local computation output* in order to cope with delays in an efficient manner. Indeed, when one agent is updating much more frequently than the others, this kind of update prevents the master value to derail towards a biased solution.

Notion of epoch. To encompass general delays (no bounding assumptions, no probability assumption), we introduce a new time scale. We define the *epoch sequence* $(k_m)_m$ as a stopping time in the martingale sense: k_{m+1} is the first moment k when $\overline{x}^k$ no longer directly depends on information from moments $k < k_m$. Mathematically,

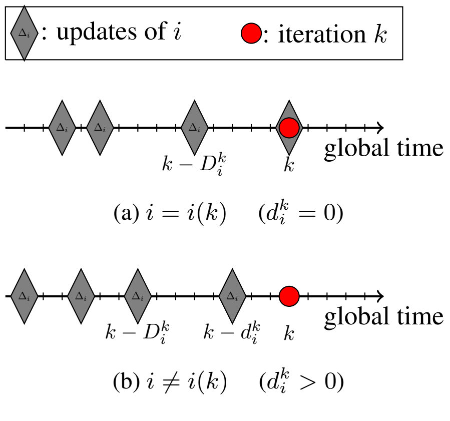

Figure 2: Illustration of the delays notation.

$$\begin{aligned} k_0 &= 0 \\ k_{m+1} &= \min \{k : \text{each machine made at least} \\ &\qquad\qquad \text{2 updates on the interval } [k_m, k]\} \end{aligned}$$

For any M, we see from (3) that this definition implies that $\overline{x}^k$ depends on local parameters $(x_i^{k-d_i^k})_i$, which themselves were computed using (once more delayed) global parameters $(\overline{x}_i^{k-D_i^k})_i$. Thus k_{m+1} is exactly the first time such that $\overline{x}^k$ does not depend directly on any $\overline{x}^{k'}$ with $k' < k_m$. These notions will be crucial in our analysis.

A nice property of this definition is that if $M = 1$, we have $k_m = m$ as on the interval $[m, m+1]$ there are exactly two updates of the one and only slave. Also, if the delays d_i^k are uniformly bounded by d (which is not our working assumption), then we have $D_i^k \leqslant D := 2d+1$ and $k_{m+1} - k_m \leqslant D$.

3.2. RPG Optimization scheme

A core feature of our proposed algorithm is the way we compute the update Δ (or equivalently x_i^k), which we call RPG. As Problem (2) features a smooth and a non-smooth part, it is natural to use proximal gradient steps; also, as we wish to be flexible in terms of computing time, we allow the repetition of local proximal gradient steps before exchanging with the master. The number of repetitions (called p in the algorithm) can vary over time and agent. We present our RPG scheme in 3 stages.

♦ **G.** If $r \equiv 0$, then the objective function is merely an average of smooth functions, thus Δ might be generated by a simple gradient step:

$$\begin{aligned} x_i^k &\leftarrow \overline{x}^{k-d_i^k} - \gamma \nabla f_i(\overline{x}^{k-d_i^k}) \\ \Delta &\leftarrow \pi_i \left(x_i^k - x_i^{k-d_i^k} \right). \end{aligned} \tag{4}$$

The first line in (4) corresponds to one gradient step and the second maintains the averaging property. It is noteworthy that if we use (4) to express $\overline{x}^k$ in terms of the previous $\overline{x}$'s, then the recursion will be

$$\begin{aligned}\overline{x}^k &= \sum_{i=1}^{M} \pi_i x_i^{k-d_i^k} \\ &= \sum_{i=1}^{M} \pi_i \overline{x}^{k-D_i^k} - \gamma \sum_{i=1}^{M} \pi_i \nabla f_i(\overline{x}^{k-D_i^k}). \end{aligned} \quad (5)$$

Each update is equivalent to an averaged gradient step but in a conservative manner; instead of updating from $\overline{x}^{k-1}$ as it is usually done, the proposed method uses some averaged point $\sum_{i=1}^{M} \pi_i \overline{x}^{k-D_i^k}$. Thus, though each master update relies on only one agent (and thus part of the data), all the data is always implicitly involved in the master variable, with even proportions. This allows us to cope with the heterogeneity of the computing system (data distribution, agents delays,...). This also prevents breaking convergence in the case where some slave is silent for too long.

♦ **PG.** To tackle the non-smooth part r, a standard approach is to use a proximal operator, defined by

$$\operatorname{prox}_{\gamma r}(x) = \arg\min_z \left\{ r(z) + \frac{1}{2\gamma} \|z - x\|^2 \right\}.$$

Using this, we can generalize (4) in the same way as $\operatorname{prox}_{\gamma r}(x - \gamma \nabla f(x))$ generalizes a gradient step.

$$\begin{aligned} z &\leftarrow \operatorname{prox}_{\gamma r}(\overline{x}^{k-d_i^k}), \\ x_i^k &\leftarrow z - \gamma \nabla f_i(z) \\ \Delta &\leftarrow \pi_i \left(x_i^k - x_i^{k-d_i^k} \right). \end{aligned} \quad (6)$$

One can notice here that we are performing the proximal operation *before* the gradient step; this way, the master averages output of gradient step and not the output of proximity operators. This is a crucial point as the average of proximal operators, known as the proximal average (Bauschke et al., 2008) is itself a proximal operator but on a Moreau-like envelop of the original function (that has been applied in learning contexts, see e.g. (Yu, 2013)). In order to converge to a solution of (2) itself, the average must be taken on the output of gradient steps and not the output of proximity operators. Thus, the converging master variable is

$$\hat{x}^k := \operatorname{prox}_{\gamma r}\left(\overline{x}^k\right). \quad (7)$$

♦ **RPG.** After computing iteration (6), the slave could send the adjustment Δ to the master and get $\overline{x}^k$ in response. However, the difference between the latest $\overline{x}$ and $\overline{x}^{k-d_i^k}$ might be small, so the slave would not get much of information. When communications costs are prohibitive, it is preferable to limit the number of communications. Instead, we suggest to perform additional proximal gradient updates by taking as the starting point $\overline{x}^{k-d_i^k} + \Delta$. We will show in Section 4.2 that there is no restrictions on the number of repetitions (p in the algorithm), as any value can be chosen and it can vary both across machines and over time.

3.3. Full algorithm

Putting together our communication and optimization schemes, we end up with the following distributed algorithm, called DAve-RPG. We present it using the notation of the initial learning problem (2).

DAve-RPG

```
Master:
Initialize $\overline{x} = \overline{x}^0$, $k = 0$
while not converge do
    when an agent finishes an iteration:
    Receive an adjustment $\Delta$ from it
    $\overline{x} \leftarrow \overline{x} + \Delta$
    Send $\overline{x}$ to the agent in return
    $k \leftarrow k + 1$
end
Interrupt all slaves
Output $\hat{x} = \operatorname{prox}_{\gamma r}(\overline{x})$

Slave $i$:
Initialize $x = x_i^0 = \overline{x}^0$
while not interrupted by master do
    Receive the most recent $\overline{x}$
    Take $x$ from the previous iteration
    Select a number of repetitions $p$
    Initialize $\Delta = 0$
    for $q = 1$ to $p$ do
        $z \leftarrow \operatorname{prox}_{\gamma r}(\overline{x} + \Delta)$
        $x^+ \leftarrow z - \gamma \frac{1}{n_i} \sum_{j \in \mathcal{S}_i} \nabla \ell_j(z)$
        $\Delta \leftarrow \Delta + \pi_i (x^+ - x)$
        $x \leftarrow x^+$
    end
    Send the adjustment $\Delta$ to the master
end
```

4. Convergence analysis

In this section, we prove that DAve-RPG has a linear convergence rate with respect to the epoch sequence when the objective is strongly convex.

Assumption 1. *The functions (ℓ_i) are μ-strongly convex and L-smooth, that is differentiable with L-Lipschitz gradients; and r is convex and proper.*

In this situation, we have existence and uniqueness of the solution of (1), that we denote by $x^\star$. We prove in Section 4.1 a general convergence result that we refine later for multiple local iterates in Section 4.2, and for specific learning problems in Section 4.3.

4.1. General convergence under strong convexity

We first prove that we have a control on the convergence of local iterates of a slave with respect to the other slaves values, and that for any number of inner repetitions p, as stated in the next lemma (the proof of which is reported in the supplementary material).

Lemma 1. *Let Assumption 1 hold. For any i, define $x_i^\star = x^\star - \gamma \frac{1}{n_i} \sum_{j \in \mathcal{S}_i} \nabla \ell_j(x^\star)$. Then, the local iterates at slave i satisfy, for all k,*

$$\left\|x_i^k - x_i^\star\right\|^2 \leqslant (1-\gamma\mu)^2\, c_{k-D_i^k}$$

where for any moment k we define

$$c_k = \max\left(\left\|\overline{x}^k - \overline{x}^\star\right\|^2, \left\|\overline{x}^k_{-i(k)} - \overline{x}^\star_{-i(k)}\right\|^2\right)$$

with $i(k)$ being the slave making the update k, and

$$\begin{aligned}
\overline{x}^\star &= \sum_{i=1}^M \pi_i x_i^\star, \\
\overline{x}^\star_{-i} &= \Big(\sum_{j\neq i} \pi_j\Big)^{-1} \sum_{j \neq i} \pi_j x_j^\star = \sum_{j\neq i} \frac{n_j}{n-n_i} x_j^\star, \\
\overline{x}^k_{-i} &= \Big(\sum_{j\neq i} \pi_j\Big)^{-1} \sum_{j \neq i} \pi_j x_j^k = \sum_{j\neq i} \frac{n_j}{n-n_i} x_j^k.
\end{aligned}$$

This technical lemma is the working horse bringing the following global convergence result. We prove that, along the *epoch sequence* (k_m), the proposed algorithm converges linearly.

Theorem 1. *Let Assumption 1 hold and take $\gamma \in \left(0, \frac{2}{\mu+L}\right]$. Then for all $k \in [k_m, k_{m+1})$, `DAve-RPG` produces iterate verifying*

$$\left\|\hat{x}^k - x^\star\right\|^2 \leqslant (1-\gamma\mu)^{2m} \max_i \left\|x_i^0 - x_i^\star\right\|^2.$$

Proof. Let us take $i = i(k)$,the slave making the update at moment k. Using this definition, we are going to prove convergence of the sequence

$$c_k = \max\left(\left\|\overline{x}^k - \overline{x}^\star\right\|^2, \left\|\overline{x}^k_{-i} - \overline{x}^\star_{-i}\right\|^2\right),$$

where $\overline{x}^\star$, $\overline{x}^\star_{-i}$ and $\overline{x}^k_{-i}$ are defined as in Lemma 1. According to this lemma, for any j

$$\left\|x_j^k - x_j^\star\right\|^2 \leqslant (1-\rho)^2 c_{k-D_j^k}$$

with $\rho = \gamma\mu$. Using the convexity of the squared norm, we deduce that

$$\begin{aligned}
\left\|\overline{x}^k - \overline{x}^\star\right\|^2 &\leqslant \sum_{j=1}^M \pi_j \left\|x_j^k - x_j^\star\right\|^2 \\
&\leqslant (1-\rho)^2 \sum_{j=1}^M \pi_j c_{k-D_j^k} \leqslant (1-\rho)^2 \max_j c_{k-D_j^k},
\end{aligned}$$

because $\sum_j \pi_j = 1$. Similarly,

$$\left\|\overline{x}^k_{-i} - \overline{x}^\star_{-i}\right\|^2 \leqslant (1-\rho) \max_{j \neq i} c_{k-D_j^k}.$$

What values may $k - D_j^k$ take? Clearly, $D_j^k > 0$, and on the other hand, for $k \in [k_m, k_{m+1})$, there were at least two updates by slave j in $[k_{m-1}, k_m]$ so $k_{m-1} \leqslant k - D_j^k$ thus

$$c_k \leqslant (1-\rho)^2 \max_j c_{k-D_j^k} \leqslant (1-\rho)^2 \max_{k' \in [k_{m-1}, k)} c_{k'}.$$

If we apply this inequality sequentially to $k_m, k_m + 1, \ldots, k_{m+1} - 1$, we get

$$\begin{aligned}
c_{k_m} &\leqslant (1-\rho)^2 \max_{k' \in [k_{m-1}, k_m)} c_{k'}, \qquad (8)\\
c_{k_m+1} &\leqslant (1-\rho)^2 \max\left(\max_{k' \in [k_{m-1}, k_m)} c_{k'}, c_{k_m}\right) \\
&\leqslant (1-\rho)^2 \max_{k' \in [k_{m-1}, k_m)} c_{k'} \quad \text{(using Eq. (8))} \\
&\ldots \\
\max_{k \in [k_m, k_{m+1})} c_k &\leqslant (1-\rho)^2 \max_{k' \in [k_{m-1}, k_m)} c_{k'} \\
&\leqslant (1-\rho)^{2m} \max_{k' < k_0} c_{k'} \\
&\leqslant (1-\rho)^{2m} \max_i \left\|x_i^0 - x_i^\star\right\|^2.
\end{aligned}$$

From the linear convergence of (c_k) along the epoch sequence, we are now able to prove the convergence rate of the produced iterates. The optimality of $x^\star$ implies that it is the fixed point of a (full) proximal gradient step

$$\begin{aligned}
& 0 \in \sum_{i=1}^M \pi_i \nabla f_i(x^\star) + \partial r(x^\star) \\
\Leftrightarrow\ & x^\star = \mathrm{prox}_{\gamma r}\left(x^\star - \gamma \sum_{i=1}^M \pi_i \nabla f_i(x^\star)\right) \\
& \quad = \mathrm{prox}_{\gamma r}\left(\sum_{i=1}^M \pi_i x_i^\star\right) = \mathrm{prox}_{\gamma r}\left(\overline{x}^\star\right).
\end{aligned}$$

Finally, using the non-expansiveness of the proximal operator, we get

$$\begin{aligned}
\left\|\hat{x}^k - x^\star\right\|^2 &= \left\|\mathrm{prox}_{\gamma r}\left(\overline{x}^k\right) - \mathrm{prox}_{\gamma r}\left(\overline{x}^\star\right)\right\|^2 \\
&\leqslant \left\|\overline{x}^k - \overline{x}^\star\right\|^2 \leqslant c_k
\end{aligned}$$

which concludes the proof. □

This result states that the proposed algorithm converges linearly at a rate that depends only on the function properties but neither on the number of machines nor on the delays as they are directly embedded in the sequence (k_m). When there is only one machine, the epochs sequence corresponds to the time sequence ($m = k$), and we recover the standard rate for the proximal-gradient algorithm.

It is important to notice here that the stepsize γ can be taken in the usual range for (proximal) gradient descent and in particular does not depend on the delays in any way, in contrast with many existing asynchronous algorithms presented in Section 2.

Thus the delays and the computing system do not appear in the choice of the stepsize and in fact anywhere of the algorithm. As a consequence, the algorithm shows a stable, efficient behavior, even in the presence of arbitrary delays. This is illustrated in the numerical experiments with long delays, simulating crashes/repairs. This is also studied in the appendix in the case of infinite delays, simulating crashes with loss of data; in this case, the algorithm still provides an approximate solution, depending on previously computed information.

4.2. Improved rate for multiple inner iterations

The general convergence analysis of the previous section can be refined in the case when the slaves make at least p_0 proximal-gradient iterations (the proof of this result is postponed to the supplementary).

Theorem 2. *In addition to the assumptions of Theorem 1, assume that every local loop in* `DAve-RPG` *uses $p \geqslant p_0$. Then,*

$$\left\|\hat{x}^k - x^\star\right\|^2 \leqslant [\eta(p_0)]^m \max_i \left\|x_i^0 - x_i^\star\right\|^2$$

where

$$\eta(p_0) = (1-\rho)^2 \left(1 - \frac{\rho}{M} - \cdots - \frac{\rho(1-\rho)^{p_0-2}}{M^{p_0-1}}\right)^2$$

with $\rho = \gamma\mu$.

When $p_0 = 1$, we retrieve the rate of Theorem 1. When $p_0 > 1$, the guaranteed convergence rate is better at the price of more expensive iterations. We have here the usual trade-off in distributed optimization between local computations and global exchanges.

4.3. Analysis for learning with elastic net regularization

The strong convexity of the functions ℓ_i's required by the analysis of the previous sections may seem restrictive at first glance. But it is actually verified in many machine learning objectives. It is for example the case when learning with ℓ_1 and ℓ_2 regularization (a.k.a. elastic net (Zou & Hastie, 2005)). The learning optimization problem can be formulated as

$$\min_x \frac{1}{n}\sum_{j=1}^n l_j(x) + \lambda_1\|x\|_1 + \frac{\lambda_2}{2}\|x\|_2^2, \tag{9}$$

where the (l_j) are losses over some example (z_j, y_j). Typical functions include least-squares regression $l_j(x) = (z_j^T x - y_j)^2$, logistic loss $l_j(x) = \log(1 + \exp(-y_j z_j^T x))$. The only assumption that we require on the (l_j)'s is the L-smoothness. The squared ℓ_2 norm can be split over all examples, which actually brings strong convexity in the smooth part:

$$\min_x \frac{1}{n}\sum_{j=1}^n \underbrace{\left(l_j(x) + \frac{\lambda_2}{2}\|x\|_2^2\right)}_{:=\ell_j(x)} + \lambda_1\|x\|_1, \tag{10}$$

where the smooth local losses ℓ_j's are now $(L+\lambda_2)$-smooth λ_2-strongly convex functions.

Moreover, the proximal operator of the ℓ_1 norm has the explicit form as an elementwise soft thresholding

$$\left[\mathrm{prox}_{\gamma\lambda_1\|\cdot\|_1}(x)\right]_a = \begin{cases} [x]_a - \gamma\lambda_1 & \text{if } x_a > \gamma\lambda_1, \\ 0 & \text{if } |x_a| \leqslant \gamma\lambda_1, \\ [x]_a + \gamma\lambda_1 & \text{if } x_a < -\gamma\lambda_1, \end{cases}$$

where $[\cdot]_a$ denotes the a-th coordinate of a vector. Thus, if the data is split into subsets $(\mathcal{S}_i)$ distributed over M machines, an inner *proximal gradient* update for our `DAve-RPG` algorithm writes

$$z \leftarrow \mathrm{prox}_{\gamma\lambda_1\|\cdot\|_1}\left(\overline{x} + \Delta\right) \tag{11a}$$

$$x^+ \leftarrow (1-\lambda_2)\, z - \gamma\frac{1}{n_i}\sum_{j\in\mathcal{S}_i} \nabla l_j(z) \tag{11b}$$

$$\Delta \leftarrow \Delta + \pi_i\left(x^+ - x\right) \tag{11c}$$

$$x \leftarrow x^+. \tag{11d}$$

The linear convergence result of Theorem 1 can then be restated in the following specific form.

Corollary 1. *Assume that $\lambda_2 > 0$ and that the functions (l_i) in* (10) *are (non-strongly) convex and L-smooth. Take $\gamma \in \left(0, \frac{2}{L+2\lambda_2}\right]$, then the iterations of* `DAve-RPG` *for $k \in [k_m, k_{m+1})$ with update rules* (11) *for machine $i \in \{1, \ldots, m\}$ verify :*

$$\left\|\hat{x}^k - x^\star\right\|^2 \leqslant (1-\gamma\lambda_2)^{2m} \max_i \left\|x_i^0 - x_i^\star\right\|^2.$$

5. Numerical experiments

We illustrate the performance of our algorithm `DAve-RPG` on a standard setting of large-scale learning. We consider the regularized logistic surrogate loss

$$\frac{1}{n}\sum_{j=1}^{n}\log(1+\exp(-y_j z_j^T x)) + \lambda_1\|x\|_1 + \frac{\lambda_2}{2}\|x\|_2^2\,, \quad (12)$$

with the hyperparameter λ_2 fixed to the typical value $\lambda_2 = \frac{1}{n}$. In forthcoming plots, we will report the decrease of this function with respect to wall clock time.

We use publicly available datasets[2], presented in Table 1. We consider $M = 100$ machines connected by fast data transmission. Each slave is allocated 1 CPU with 4 GB of memory. We report some of the numerical results here; others are included in supplementary material.

Dataset	n	d	L	λ_1
RCV1	697641	47236	0.25	$3 \cdot 10^{-6}$
URL	2396130	3231961	128,4	10^{-6}
Covtype	581012	54	21930585,25	10^{-6}

Table 1: Characteristics of datasets used in our experiments; n, d, L and λ_1 denote respectively, the size of the training set, the number of features, the Lipschitz constant, and the value of the hyperparamater corresponding to the ℓ_1 regularization.

Comparison. We compare our algorithm `DAve-RPG` in terms of speed of convergence with its two main competitors in our distributed framework (with splitting of examples and no shared memory): i) Proximal Incremental Average Gradient (PIAG), using the maximum of two stepsizes provided in (Aytekin et al., 2016) and (Vanli et al., 2016) (for better performance); and ii) the direct extension of the proximal-gradient in Map-Reduce that we refer to as synchronous Proximal Gradient (Synchronous PG). These algorithms were implemented in Python by using sockets for communications between the workers and the master. All communications are processed one by one to avoid inconsistent reads and support the theory.

Figure 3 illustrates the difference between the regularized loss (12) and its minimum, denoted as suboptimality, with respect to time for the three above methods on RCV1.binary and the URL datasets. We put a significant proportion of the data on the first machine (9% and 15% respectively) and the rest is evenly distributed among the other 99 workers. We fixed the number of inner iterations per worker to $p = 1$. It comes out that `DAve-RPG` consistently outperforms the other two approaches.

Delay-tolerance In Figure 4, we exhibit the resilience of our algorithm to delays by introducing additional simulated delays. We use the RCV1 dataset distributed evenly among $M = 10$ machines, meaning that a long delays for one machine would hold out 10% of the data. Delays are simulated by randomly stopping any machine for some random time. We can see that while delays obviously affect the convergence rate, the speed remains comparable. This is an important feature of our algorithm, especially when looking at the maximal delay $d^k = \max_i d_i^k$ record which is varying a lot as expected from a practical point of view. Notice that the delays are only upper bounded by a large value $d \approx 300$ which would deeply affect the stepsize and convergence of competitor algorithms, but not ours.

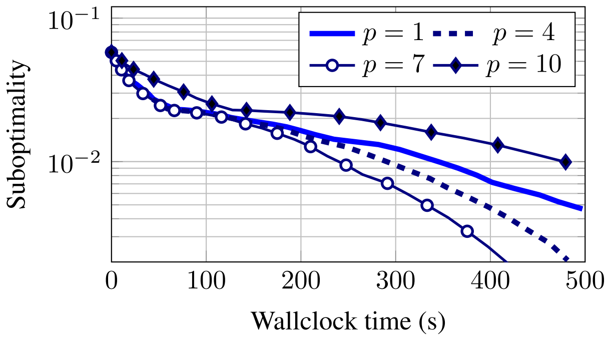

Figure 5: Trade-off computation/communication with p for URL dataset.

Trade-off communication vs computation To illustrate the trade-off between communication and computation, we increase the number of inner-iterations of each worker ($p = 1, 4, 7, 10$). These results are depicted in Figure 5 for the URL dataset splitted evenly among the 100 workers. We see that the increase of p from 1 to 7 allows to reach the minimum faster, up to a certain point, where the time spent on inner iterations affect the speed of convergence as for $p = 10$.

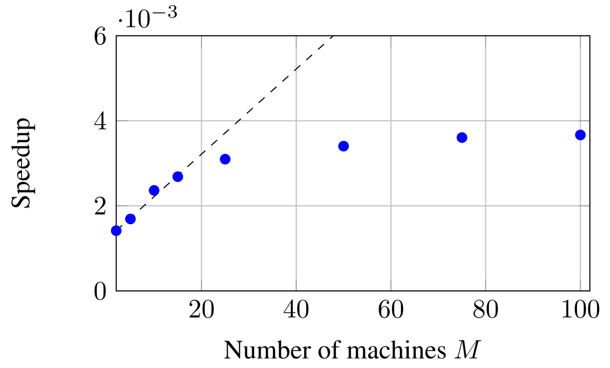

Figure 6: Speedup with respect to number of machines on RCV1.

Scalability We ran the algorithm with different number of workers and measured its speedup as the inverse of the required time to reach suboptimality 10^{-2}; this is represented

[2] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

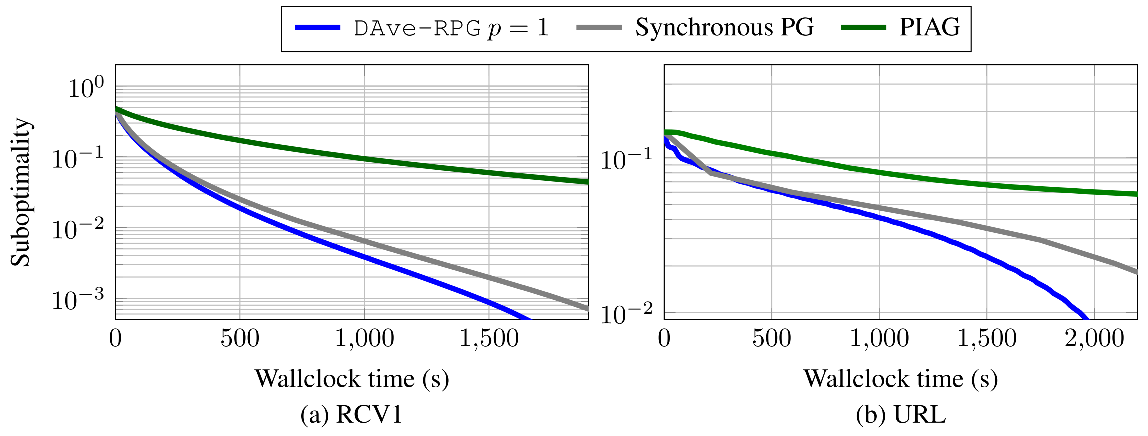

Figure 3: Regularized loss (12) suboptimality on the training set versus wall clock time. On the left: RCV1 dataset with 9% of the data on the 1st machine. On the right: URL dataset with 15% of the data on the 1st machine.

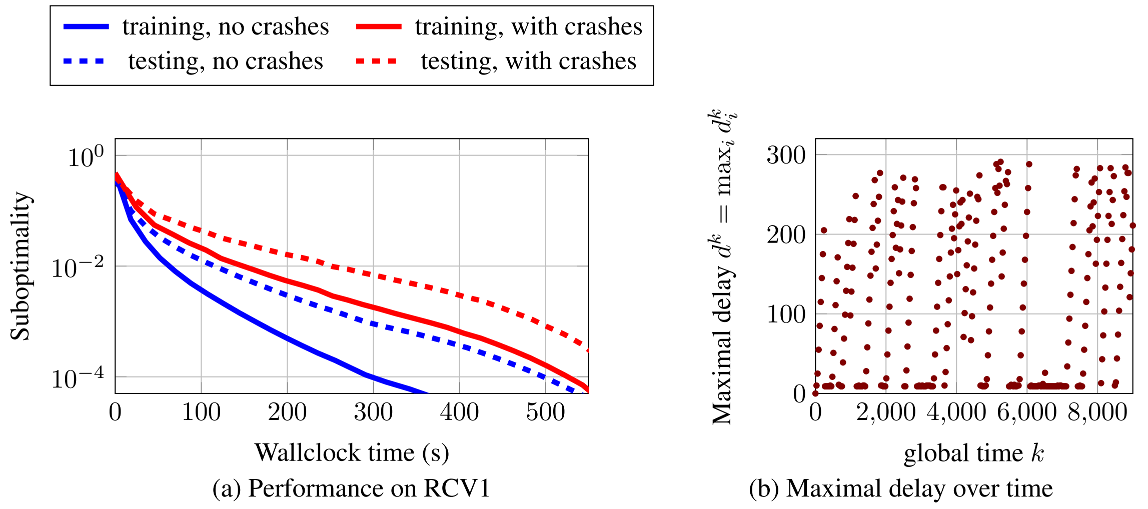

Figure 4: Illustration of the resilience of DAve-RPG to additional delays.

in Fig. 6. We set $p = 50$ and distributed RCV1 dataset evenly in all runs. In the ideal case, performance would scale linearly with the number of workers. We observe that the speedup is far from linear but it is not surprising from a practical point of view. Indeed, the main reason for this behaviour is that the greater the number of workers, the bigger the asynchrony becomes. Besides, the communication bottleneck becomes narrower especially as we processed communications one by one to avoid inconsistent reads.

6. Conclusion

We proposed and analyzed an asynchronous distributed optimization algorithm for learning objectives. The key features of our algorithm are i) a distributed averaging of the parameters; and ii) repeated local proximal-gradient iterations. Dave-RPG enjoys a linear convergence rate with a fixed stepsize independent of delays. These convergence properties and the possibility to adjust the computation vs. communication trade-off by repeating the local iterations make it particularly well suited for distributed learning, as demonstrated in the numerical experiments.

References

Aytekin, A., Feyzmahdavian, H. R., and Johansson, M. Analysis and implementation of an asynchronous optimization algorithm for the parameter server. *arXiv preprint arXiv:1610.05507*, 2016.

Bauschke, H. H., Goebel, R., Lucet, Y., and Wang, X. The proximal average: basic theory. *SIAM Journal on Optimization*, 19(2):766–785, 2008.

Calauzènes, C. and Roux, N. L. Distributed saga: Maintaining linear convergence rate with limited communication. *arXiv preprint arXiv:1705.10405*, 2017.

Defazio, A., Bach, F., and Lacoste-Julien, S. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pp. 1646–1654, 2014.

Feyzmahdavian, H. R., Aytekin, A., and Johansson, M. A delayed proximal gradient method with linear convergence rate. In *Machine Learning for Signal Processing (MLSP), 2014 IEEE International Workshop on*, pp. 1–6. IEEE, 2014.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pp. 315–323, 2013.

Konečnỳ, J., McMahan, H. B., Ramage, D., and Richtárik, P. Federated optimization: distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. ASAGA: Asynchronous Parallel SAGA. In *20th International Conference on Artificial Intelligence and Statistics*, pp. 46–54, 2017.

Lee, J. D., Lin, Q., Ma, T., and Yang, T. Distributed stochastic variance reduced gradient methods and a lower bound for communication complexity. *arXiv preprint arXiv:1507.07595*, 2015.

Ma, C., Smith, V., Jaggi, M., Jordan, M., Richtarik, P., and Takac, M. Adding vs. averaging in distributed primal-dual optimization. In *International Conference on Machine Learning*, pp. 1973–1982, 2015.

Ma, C., Jaggi, M., Curtis, F. E., Srebro, N., and Takáč, M. An accelerated communication-efficient primal-dual optimization framework for structured machine learning. *arXiv preprint arXiv:1711.05305*, 2017a.

Ma, C., Konecny, J., Jaggi, M., Smith, V., Jordan, M. I., Richtarik, P., and Takac;, M. Distributed optimization with arbitrary local solvers. *Optimization Methods Software*, 32(4):813–848, 2017b.

Pedregosa, F., Leblond, R., and Lacoste-Julien, S. Breaking the nonsmooth barrier: A scalable parallel method for composite optimization. *Advances in Neural Information Processing System 30 (NIPS)*, 2017.

Peng, Z., Xu, Y., Yan, M., and Yin, W. Arock: an algorithmic framework for asynchronous parallel coordinate updates. *SIAM Journal on Scientific Computing*, 38(5): A2851–A2879, 2016.

Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pp. 693–701, 2011.

Shalev-Shwartz, S. and Zhang, T. Accelerated mini-batch stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pp. 378–385, 2013.

Sun, T., Hannah, R., and Yin, W. Asynchronous coordinate descent under more realistic assumptions. *arXiv preprint arXiv:1705.08494*, 2017.

Takáč, M., Richtárik, P., and Srebro, N. Distributed mini-batch sdca. *arXiv preprint arXiv:1507.08322*, 2015.

Vanli, N. D., Gurbuzbalaban, M., and Ozdaglar, A. A stronger convergence result on the proximal incremental aggregated gradient method. *arXiv preprint arXiv:1611.08022*, 2016.

Yu, Y.-L. Better approximation and faster algorithm using the proximal average. In *Advances in neural information processing systems*, pp. 458–466, 2013.

Zhang, R. and Kwok, J. Asynchronous distributed admm for consensus optimization. In *International Conference on Machine Learning*, pp. 1701–1709, 2014.

Zhao, S.-Y. and Li, W.-J. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *AAAI*, pp. 2379–2385, 2016.

Zou, H. and Hastie, T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.