Delay-adaptive Step-sizes for Asynchronous Learning

Xuyang Wu [1] **Sindri Magnússon** [2] **Hamid Reza Feyzmahdavian** [3] **Mikael Johansson** [1]

Abstract

In scalable machine learning systems, model training is often parallelized over multiple nodes that run without tight synchronization. Most analysis results for the related asynchronous algorithms use an upper bound on the information delays in the system to determine learning rates. Not only are such bounds hard to obtain in advance, but they also result in unnecessarily slow convergence. In this paper, we show that it is possible to use learning rates that depend on the actual time-varying delays in the system. We develop general convergence results for delay-adaptive asynchronous iterations and specialize these to proximal incremental gradient descent and block-coordinate descent algorithms. For each of these methods, we demonstrate how delays can be measured on-line, present delay-adaptive step-size policies, and illustrate their theoretical and practical advantages over the state-of-the-art.

1. Introduction

This paper considers step-sizes that adapt to the true delays in asynchronous algorithms for solving optimization problems in the form

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

where $f : \mathbb{R}^d \to \mathbb{R}$ is a smooth but possibly non-convex loss function and $R : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ is a convex nonsmooth function. Here, R is typically a regularizer, promoting desired solution properties such as sparsity, or the indicator function of a closed convex set (the constraint set for x).

When either the data dimension (the number of samples defining f) or the variable dimension d is large, we may need to distribute the optimization process over multiple compute nodes. In a distributed environment, synchronous algorithms such as gradient descent or block coordinate descent, are often inefficient. Since they need to wait for the slowest worker node to complete its task, the system tends to spend a significant time idle and becomes sensitive to single node failures. This motivates the development of asynchronous algorithms which allow all nodes to run at their maximal capacity without synchronization overhead.

In the past decade, numerous asynchronous algorithms have been proposed to solve large-scale problems on the form (1). Notable examples include ARock (Peng et al., 2016), PIAG (Aytekin et al., 2016; Vanli et al., 2018), Async-BCD (Liu et al., 2014), AsyFLEXA (Cannelli et al., 2016), DAve-RPG (Mishchenko et al., 2018), and the widely studied asynchronous SGD (Dean et al., 2012; Recht et al., 2011; Sra et al., 2016), to mention a few. Algorithms that use fixed step-sizes often assume bounded asynchrony and require an upper bound of the worst-case information delay to determine the step-size. However, such an upper bound is usually difficult to obtain in advance, and is a crude model for actual system delays. Indeed, actual latencies may be significantly smaller than the worst case for most nodes, and for most of the time. This makes the algorithm hard to tune and inefficient to run, since a large worst-case delay leads to a small step-size and a slow iterate convergence.

1.1. Algorithms and related work

In this paper, we develop general principles and convergence results for asynchronous optimization algorithms that adjust the learning rate on-line to the actual information delays. We then present concrete delay-tracking algorithms and adaptive step-size policies for two specific asynchronous optimization algorithms, PIAG and Async-BCD. These algorithms address two distinct variations of distributed model training: distribution of data over samples (PIAG) and distribution of variable updates across features (Async-BCD). To put our work in context, we review the related literature below.

PIAG: PIAG solves problem (1) with aggregated loss $f(x) = \frac{1}{n}\sum_{i=1}^{n} f^{(i)}(x)$. Here, each $f^{(i)}$ could represent the training loss on sample i, on mini-batch i or on the complete data set held by some worker node. The PIAG

[1]Division of Decision and Control Systems, EECS, KTH Royal Institute of Technology, Stockholm, Sweden [2]Department of Computer and System Science, Stockholm University, Stockholm, Sweden [3]ABB Corporate Research, Västerås, Sweden. Correspondence to: Xuyang Wu <xuyangw@kth.se>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

algorithm is often implemented in a parameter server architecture (Li et al., 2013), where a master node updates the iterate x_k based on the most recent gradient information from each worker. The new iterate is broadcast to idle workers, who proceed to compute the gradient of the training loss on their local data set, and return the gradient to the master node. Both master and worker nodes operate in an event-driven fashion without any global synchronization.

Early works on PIAG (Blatt et al., 2007; Roux et al., 2012; Gurbuzbalaban et al., 2017) focused on smooth problems, *i.e.*, let $R \equiv 0$ in (1). Extensions of PIAG that allow for a non-smooth regularizer include (Aytekin et al., 2016; Vanli et al., 2018; Feyzmahdavian & Johansson, 2021) for convex f and (Deng et al., 2020; Sun et al., 2019) for non-convex f. In addition, a recent work (Wai et al., 2020) compensates for the information delays in PIAG using Hessian information. However, all these papers use an upper bound of the worst-case delay to determine the step-size.

Async-BCD: Async-BCD splits the whole variable x into multiple blocks $\{x^{(i)}\}_{i=1}^m$ and solves problem (1) with separable nonsmooth function $R(x) = \sum_{i=1}^m R^{(i)}(x^{(i)})$. The algorithm is usually implemented in a shared memory architecture (Peng et al., 2016), where the iterate is stored in shared memory and multiple servers asynchronously and continuously update one block at a time based on the delayed iterates they read from the shared memory.

Existing works on Async-BCD include (Liu et al., 2014; Liu & Wright, 2015; Davis, 2016; Sun et al., 2017), among which (Sun et al., 2017) considers smooth problems ($R^{(i)} \equiv 0$), (Liu et al., 2014) requires $R^{(i)}$ to be an indicator function, and (Liu & Wright, 2015; Davis, 2016) allow for general convex $R^{(i)}$. In addition, some asynchronous methods use updates that are similar to Async-BCD, such as ARock (Peng et al., 2016; Hannah & Yin, 2018; Feyzmahdavian & Johansson, 2021) and AsyFLEXA (Cannelli et al., 2016). All these papers except (Hannah & Yin, 2018) consider fixed step-sizes tuned based on a uniform upper bound of the delays. The work (Hannah & Yin, 2018) suggests a step-size that relies on the actual delays but is relatively conservative.

1.2. Contributions

This paper introduces delay-adaptive step-sizes for asynchronous optimization algorithms. We demonstrate how information delays can be accurately recorded on-line, introduce a family of dynamic step-size policies that adapt to the true amount of asynchrony in the system, and give a formal proof for convergence under all bounded delays. This eliminates the need to know an upper bound of the delays to set the learning rate and removes the (typically significant) performance penalty that occurs when this upper bound is larger than the true system delays. We make the following specific contributions:

- We develop simple and practical delay tracking algorithms for PIAG in the parameter server and for Async-BCD in shared memory.
- We derive a novel convergence result that simplifies the analysis of broad classes of asynchronous optimization algorithms, and allows to analyze the effect of a time-varying and delay-dependent learning rate.
- We demonstrate how a natural extension of the fixed step-sizes proposed for asynchronous optimization to the delay-adaptive setting fails, and suggest a general step-size principle that ensures convergence under all bounded delays, even if their upper bound is unknown.
- Under the step-size principle, we design two delay-adaptive step-size policies that use the true delay. We derive explicit convergence rate guarantees for PIAG and Async-BCD under these step-size policies, compare these with the state-of-the-art, and identify scenarios where our new step-sizes give large speed-ups.

Experiments on a classification problem show that the proposed delay-adaptive step-sizes accelerate the convergence of the two methods compared to the best known fixed step-sizes from the literature.

Notation and Preliminaries

We use $\mathbb{N}$ and $\mathbb{N}_0$ to denote the set of natural numbers and the set of natural numbers including zero, respectively. We let $[m] = \{1, \dots, m\}$ for any $m \in \mathbb{N}$ and define the proximal operator of a function $R : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ as

$$\operatorname{prox}_R(x) = \operatorname*{argmin}_{y \in \mathbb{R}^d} R(y) + \frac{1}{2}\|y - x\|^2.$$

We say a function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth if it is differentiable and

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \ \forall x, y \in \mathbb{R}^d.$$

For L-smooth function $f : \mathbb{R}^d \to \mathbb{R}$ and convex function $R : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$, we say that $P(x) = f(x) + R(x)$ satisfies the proximal PL condition (Karimi et al., 2016) with some $\sigma > 0$ if

$$\sigma(P(x) - P^\star) \leq -L\hat{P}(x), \ \forall x \in \operatorname{dom}(P), \qquad (2)$$

where $\hat{P}(x) = \min_{y \in \mathbb{R}^d} \{\langle \nabla f(x), y - x\rangle + \frac{L}{2}\|y - x\|^2 + R(y) - R(x)\}$ and $P^\star = \min_{x \in \mathbb{R}^d} P(x)$.

2. Algorithms with delay-tracking

In this section, we first introduce the PIAG and Async-BCD algorithms and demonstrate how they can record actual system delays with almost no overhead. The key to this observation is that delays in asynchronous algorithms are typically not measured in physical time, but rather in the number of write events that have occurred since the model parameters that are used in the update were computed (see, *e.g.*, (Leblond et al., 2018)). Hence, in the parameter server architecture and the shared memory systems, delays can often be computed accurately without any intricate time synchronization between distributed nodes. We then demonstrate how the natural extension of the state-of-the-art step-size rules for worst-case delays fails to extend to actual delays.

2.1. PIAG in a parameter server architecture

PIAG solves problem (1) with aggregated loss $f(x) = \frac{1}{n}\sum_{i=1}^n f^{(i)}(x)$ and takes the following form:

$$g_k = \frac{1}{n}\sum_{i=1}^{n} \nabla f^{(i)}(x_{k-\tau_k^{(i)}}), \tag{3}$$

$$x_{k+1} = \mathrm{prox}_{\gamma_k R}(x_k - \gamma_k g_k), \tag{4}$$

where $\tau_k^{(i)} \in [0, k]$ is the delay of $\nabla f^{(i)}$ at the kth iteration.

Parameter server: PIAG is usually implemented in a parameter server framework (Li et al., 2013) with one master and n workers, each one capable of computing (stochastic, mini-batch, or full) gradients of a specific $f^{(i)}$. The master maintains the most recent iterate x_k and the most recently received gradients $g^{(i)} = \nabla f^{(i)}(x_{k-\tau_k^{(i)}})$ from each worker. Once the master receives new gradients, it revises the corresponding $g^{(i)}$, updates the iterate, and pushes the new parameters back to idle workers. A detailed implementation of PIAG (3) – (4) in the parameter server setting without delay-tracking is presented in (Aytekin et al., 2016).

Delay-tracking: To compute the delays $\tau_k^{(i)}$, the PIAG algorithm needs to know the iteration index of the model parameters used to compute each $g^{(i)}$. In Algorithm 1, we maintain this information using a simple time-stamping procedure. Specifically, in iteration l, the master pushes the tuple (x_l, l) to idle workers. Workers return $(\nabla f^{(i)}(x_l), l)$ which the master stores as $g^{(i)} \leftarrow \nabla f^{(i)}(x_l)$, $s^{(i)} \leftarrow l$. At any iteration $k \geq l$, the delay $\tau_k^{(i)}$ is then given by $k - s^{(i)}$.

The tracking scheme in Algorithm 1 can be extended to other approaches that can also be implemented in the parameter server setting, such as Asynchronous SGD (Dean et al., 2012; Recht et al., 2011; Sra et al., 2016).

Algorithm 1 PIAG with delay-tracking

Input: initial iterate x_0, number of iteration $k_{\max} \in \mathbb{N}$.
Initialization:
The master sets $s^{(i)} \leftarrow 0$, $g^{(i)} \leftarrow \nabla f^{(i)}(x_0)\ \forall i \in [n]$, and $g_0 \leftarrow \frac{1}{n}\sum_{i=1}^n \nabla f^{(i)}(x_0)$.
The master sets $k \leftarrow 1$, $x_1 \leftarrow \mathrm{prox}_{\gamma_0 R}(x_0 - \gamma_0 g_0)$ and broadcast x_1 to all workers.
while $k \leq k_{\max}$: each worker $i \in [n]$ *asynchronously* and *continuously* **do**
  receive (x_k, k) from the master.
  compute $\nabla f^{(i)}(x_k)$.
  send $(\nabla f^{(i)}(x_k), k)$ to the master.
end while
while $k \leq k_{\max}$: the master **do**
  wait until a set $\mathcal{R}_k$ of workers return.
  for all $w \in \mathcal{R}_k$ **do**
    update $(g^{(w)}, s^{(w)}) \leftarrow (\nabla f^{(w)}(x_l), l)$.
  end for
  set $g_k \leftarrow \frac{1}{n}\sum_{i=1}^n g^{(i)}$.
  calculate the delay $\tau_k^{(i)} = k - s^{(i)}\ \forall i \in [n]$.
  determine the step-size γ_k based on $\tau_k^{(i)}\ \forall i \in [n]$.
  update $x_{k+1} \leftarrow \mathrm{prox}_{\gamma_k R}(x_k - \gamma_k g_k)$.
  set $k \leftarrow k + 1$.
  for all $w \in \mathcal{R}_k$ **do**
    send (x_k, k) to worker w.
  end for
end while

2.2. Async-BCD in the shared memory setting

Block-coordinate descent, BCD, (Hong et al., 2017) can be a powerful alternative for solving (1) when the regularizer is separable. Assume that $R(x) = \sum_{i=1}^m R^{(i)}(x^{(i)})$ where $x = (x^{(1)}, \ldots, x^{(m)})$, $x^{(i)} \in \mathbb{R}^{d_i}$ and $\sum_{i=1}^m d_i = d$. At each iteration of BCD, a random $j \in [m]$ is drawn and

$$x_{k+1}^{(j)} = \mathrm{prox}_{\gamma_k R^{(j)}}(x_k^{(j)} - \gamma_k \nabla_j f(x_k)).$$

Async-BCD parallelizes this update over n workers in a shared memory architecture (Peng et al., 2016). Workers operate without synchronization, repeatedly read the current iterate from shared memory, and update a randomly chosen block. More specifically, suppose that at time k, worker i_k updates the jth block $x_k^{(j)}$ based on the partial gradient $\nabla_j f$ at $\hat{x}_k$, where $\hat{x}_k$ is what the server i_k read from the shared memory. Then, the kth update is

$$x_{k+1}^{(j)} = \mathrm{prox}_{\gamma_k R^{(j)}}(x_k^{(j)} - \gamma_k \nabla_j f(\hat{x}_k)). \tag{5}$$

A specific aspect of Async-BCD is that while i_k reads from the shared memory, other workers may be in the process of writing. Hence, $\hat{x}_k$ itself may never have existed in the shared memory. This phenomenon is known as *inconsistent read* (Liu & Wright, 2015). However, if we assume that

Algorithm 2 Async-BCD with delay tracking

1: **Setup:** initial iterate x_0, number of iteration $k_{\max} \in \mathbb{N}$.
2: **while** $k \leq k_{\max}$: each worker $i \in [n]$ *asynchronously* and *continuously* **do**
3: sample $j \in [m]$ uniformly at random.
4: compute $\nabla_j f(\hat{x}_k)$ based on $\hat{x}_k$ read at time $s^{(i)}$.
5: calculate $\tau_k = k - s^{(i)}$.
6: determine the step-size γ_k.
7: compute $x_{k+1}^{(j)}$ by (5).
8: write on the shared memory.
9: set $k \leftarrow k+1$.
10: set $s^{(i)} = k$ and read x_k from the shared memory.
11: **end while**

each (block) write is atomic, then we can express x_k as

$$x_k = \hat{x}_k + \sum_{j \in J_k} (x_{j+1} - x_j). \tag{6}$$

where $J_k \subseteq \{0, 1, \dots, k\}$. The sum represents all updates that have occurred since i_k began reading $\hat{x}_k$ until the block update is written back to memory. We call $\tau_k = k - \min\{j : j \in J_k\}$ the delay of $\hat{x}_k$ at iteration k.

Delay-tracking: To track the delays in Async-BCD, workers need to record the value of the iterate counter when they begin reading from shared memory, and then again when they begin writing back their result. When worker i begins to read x from the shared memory in Algorithm 2, it stores the current value of the iterate counter into a local variable $s^{(i)}$. In this way, it can compute the delay $\tau_k = k - s^{(i)}$ when it is time to write back the result at iteration k. We assume that during steps 5-9, worker i_k is the only one that updates the shared memory. This is a little more restrictive than standard Async-BCD that only assumes that the write operation on Line 8 is atomic, but is needed to make sure that γ_k calculated in step 6 is used in (5) to update $x_{k+1}^{(j)}$.

The tracking technique in Algorithm 2 is applicable to many other methods for shared memory systems, such as ARock (Peng et al., 2016), asynchronous BCD (Recht et al., 2011), and AsyFLEXA (Cannelli et al., 2016).

2.3. Intuitive extension of a fixed step-size fails

Several of the least conservative results for PIAG (Sun et al., 2019; Deng et al., 2020; Feyzmahdavian & Johansson, 2021) and Async-BCD (Davis, 2016; Sun et al., 2017) use step-sizes on the form $\gamma_k = \frac{c}{\tau+b}$ where b and c are positive constants (independent of the delays) and τ is the maximal delay. A natural candidate for a delay-adaptive step-size would be one where the upper delay bound is replaced by the true system delay, *i.e.*

$$\gamma_k = \frac{c}{\tau_k + b}. \tag{7}$$

However, as the next example demonstrates, this step-size can lead to divergence even for simple problems.

Example 1. *Consider problem (1) with $n = d = 1$, $f(x) = \frac{1}{2}x^2$, and $R(x) = 0$. Suppose that $\tau_k = k \bmod T$ for all $k \in \mathbb{N}_0$ for some $T > b(e^{2/c} - 1)$. Then, the delays are bounded by $T-1$ and both PIAG and Async-BCD update as*

$$x_{k+1} = x_k - \gamma_k \nabla f(x_{k-\tau_k}) = x_k - \gamma_k x_{T \cdot \lfloor k/T \rfloor},$$

so that $x_{(k+1)T} = (1 - \sum_{t=0}^{T-1} \gamma_{kT+t}) x_{kT}$. Then, $\{x_{kT}\}$ diverges if $\sum_{t=0}^{T-1} \gamma_{kT+t} > 2$, which is indeed true by (7):

$$\sum_{t=0}^{T-1} \gamma_{kT+t} \geq \sum_{t=0}^{T-1} \frac{c}{t+b} \geq \int_b^{T+b} \frac{c}{s}\, ds = c \ln \frac{T+b}{b} > 2.$$

However, as we will demonstrate next, convergence can be guaranteed under a slightly more advanced step-size policy.

3. Delay-adaptive step-size

In this section, we prove that both PIAG and Async-BCD converge under step-size policies that satisfy

$$0 \leq \gamma_k \leq \max(0, \gamma' - \sum_{t=k-\tau_k}^{k-1} \gamma_t) \tag{8}$$

provided that also $\sum_{t=0}^{\infty} \gamma_t = +\infty$. Here, the constant γ' only depends on loss function properties, and there is no need to know the maximal value of the system delay to tune, run, or certify the system.

The convergence analysis is based on a new sequence result for asynchronous iterations, that could be applicable to many algorithms beyond the scope of this paper. We provide convergence results for PIAG and Async-BCD for several classes of problems in sections 3.2 and 3.3, respectively. In Section 3.4, we introduce a few specific step-size policies that satisfy the general principle (8) and demonstrate how they extend and improve existing fixed step-sizes both in theory and practice.

3.1. Novel sequence result for delay-adaptive sequences

Lyapunov theory, and related sequence results, are the basis for the convergence analysis of many optimization algorithms (Polyak, 1987). Asynchronous algorithms are no different (Aytekin et al., 2016; Peng et al., 2016; Davis, 2016; Bertsekas & Tsitsiklis, 1989). Several convergence results for asynchronous algorithms are unified and generalized in a recent work (Feyzmahdavian & Johansson, 2021). However, previous work has focused on only scenarios where the maximum delay is known, and existing results cannot be used to analyse delay-adaptive step-sizes like (8). The following theorem generalizes these results to allow adaption to the actually observed delay.

Theorem 1. *Suppose that the non-negative sequences* $\{V_k\}$, $\{X_k\}$, $\{W_k\}$, $\{p_k\}$, $\{r_k\}$, *and* $\{q_k\}$ *satisfy*

$$X_{k+1} + V_{k+1} \leq q_k V_k + p_k \sum_{\ell=k-\tau_k}^{k-1} W_\ell - r_k W_k \quad (9)$$

for all $k \in \mathbb{N}_0$, *where* $q_k \in (0,1]$ *and* $\tau_k \in [0,k]$. *Let* $Q_k = \Pi_{j=0}^{k-1} q_j$, $k \in \mathbb{N}_0$. *If for all* $k \in \mathbb{N}_0$, *either* $p_k = 0$ *or*

$$\frac{p_k}{Q_{k+1}} \leq \frac{r_\ell}{Q_{\ell+1}} - \sum_{t=\ell+1}^{k-1} \frac{p_t}{Q_{t+1}}, \quad \forall \ell \in [k-\tau_k, k], \quad (10)$$

then

$$V_k \leq Q_k V_0, \ \forall k \in \mathbb{N} \quad (11)$$

and

$$\sum_{k=1}^{\infty} \frac{X_k}{Q_k} \leq V_0. \quad (12)$$

Proof. See Appendix A. □

The theorem is a tool for establishing the convergence and convergence rate of X_k and V_k. The condition (9) is quite general, so the result may be useful for many methods beyond PIAG and Async-BCD that we focus on in this paper.

The theorem can be used to establish a linear convergence rate of algorithms. In particular, if $q_k \leq q$ for all $k \in \mathbb{N}_0$ and some $q \in (0,1)$ then (11)–(12) imply the linear rates

$$V_k \leq q^k V_0, \quad X_k \leq q^k V_0.$$

If we can only say that $q_k \leq 1$, then (12) yields that

$$\sum_{k=1}^{\infty} X_k < +\infty,$$

from which we conclude that $\lim_{k\to\infty} X_k = 0$.

3.2. Convergence of PIAG under principle (8)

With the help of Theorem 1, we are able to establish the following convergence guarantees for PIAG under the general step-size principle (8). The main proof idea is to show that some quantities generated by PIAG satisfy the equation (9) when (8) holds (see Lemma 1 in the Appendix).

Theorem 2. *Suppose that each* $f^{(i)}$ *is* L_i*-smooth,* R *is convex and closed, and* $P^\star := \min_x P(x) > -\infty$. *Define* $L = \sqrt{(1/n)\sum_{i=1}^n L_i^2}$. *Let* $\{x_k\}$ *be generated by the PIAG algorithm with a step-size sequence* $\{\gamma_t\}$ *that satisfies (8) with* $\gamma' = h/L$ *for some* $h \in (0,1)$. *Then,*

(1) For each $k \in \mathbb{N}$, *there exists* $\xi_k \in \partial R(x_k)$ *such that*

$$\sum_{k=1}^{\infty} \gamma_{k-1} \|\nabla f(x_k) + \xi_k\|^2 \leq \frac{2(h^2-h+1)(P(x_0)-P^\star)}{1-h}.$$

(2) If each $f^{(i)}$ *is convex, then*

$$P(x_k) - P^\star \leq \frac{P(x_0) - P^\star + \frac{1}{2a_0}\|x_0 - x^\star\|^2}{1 + \frac{1}{a_0}\sum_{t=0}^{k-1} \gamma_t},$$

where $a_0 = \frac{h(h+1)}{L(1-h)}$.

(3) If P *satisfies the proximal PL-condition (2), then*

$$P(x_k) - P(x^\star) \leq e^{-\frac{3c\sigma(1-\tilde{h})}{4(\tilde{h}^2-\tilde{h}+1)}\sum_{t=0}^{k-1}\gamma_t}(P(x_0)-P^\star),$$

where $\tilde{h} = \frac{1+h}{2}$ *and* $c = \min\left(1, \frac{1-h}{2h}\frac{L}{\sigma}\right)$.

Proof. See Appendix B. □

The three cases roughly represent non-convex (1), convex (2) and strongly convex (3) objective functions, but note that the proximal PL condition is less restrictive that strong convexity and can be satisfied by some non-convex functions.

To get explicit convergence rates, we need to specialize the results to a specific step-size policy; we will do this in Section 3.4. Still, we can already now notice that the sum of the step-sizes, $\sum_{t=0}^{k-1} \gamma_t$, dictates the convergence speed. This is immediate in case (2) and (3), but is also true in case (1), since the non-convex result also implies that

$$\min_{1\leq t\leq k} \|\nabla f(x_t) + \xi_t\|^2 \leq \frac{2(h^2-h+1)(P(x_0)-P^\star)}{(1-h)\sum_{t=0}^{k-1}\gamma_t}.$$

3.3. Convergence of Async-BCD under principle (8)

Next, we establish the convergence of Async-BCD with adaptive step-sizes for non-convex optimization problems. The following assumption is useful.

Assumption 1. f *is differentiable and there exists* $\hat{L} > 0$ *such that for all* $i, j \in [m]$ *and* $x \in \mathbb{R}^d$ *the following holds*[1]

$$\|\nabla_i f(x + U_j h_j) - \nabla_i f(x)\| \leq \hat{L}\|h_j\|, \ \forall h_j \in \mathbb{R}^{d_j}.$$

The assumption implies that f is L-smooth for some $L \in [\hat{L}, m\hat{L}]$. We consider the block-wise constant $\hat{L}$ rather than L because the former one is smaller, which in turn leads to larger step-sizes and faster convergence.

By showing that some quantities generated by Async-BCD satisfy equation (9) in Theorem 1 (see Lemma 2 in the Appendix), we derive the following theorem.

Theorem 3. *Suppose that each* $R^{(i)}$ *is convex and closed,* $P^\star := \min_x P(x) > -\infty$, *and Assumption 1 holds. Let*

[1] $U_j : \mathbb{R}^{d_j} \to \mathbb{R}^d$ maps $x^{(j)} \in \mathbb{R}^{d_j}$ into a d-dimensional vector where the jth block is $x^{(j)}$ and other blocks are 0.

$\{x_k\}$ be generated by the Async-BCD algorithm with a step-size sequence $\{\gamma_t\}$ that satisfies (8) with $\gamma' = h/\hat{L}$ for some $h \in (0,1)$. Then,

$$\sum_{k=0}^{\infty} \gamma_k \mathbb{E}[\|\tilde{\nabla} P(x_k)\|^2] \leq \frac{4m(P(x_0) - P^\star)}{1-h},$$

where $\tilde{\nabla} P(x_k) = \hat{L}(\mathrm{prox}_{\frac{1}{\hat{L}}R}(x_k - \frac{1}{\hat{L}}\nabla f(x_k)) - x_k)$.

Proof. See Appendix C. □

The theorem establishes the convergence of Async-BCD under adaptive step-sizes. Note that $\tilde{\nabla} P(x) = \mathbf{0}$ if and only if $\mathbf{0} \in \partial P(x)$, i.e., x is a stationary point of problem (1). Moreover, Theorem 3 implies

$$\min_{1 \leq t \leq k} \mathbb{E}[\|\tilde{\nabla} P(x_t)\|^2] \leq \frac{4m(P(x_0) - P^\star)}{(1-h)\sum_{t=0}^{k} \gamma_t}.$$

Similar to PIAG, a larger step-size integral leads to a smaller error bound in the above equation, which intuitively implies faster convergence of Async-BCD.

3.4. Delay-adaptive step-size satisfying (8)

By the analysis in Section 3.2–3.3, all step-sizes satisfying $\sum_{t=0}^{\infty} \gamma_t = +\infty$ and the principle (8) guarantee convergence of PIAG and Async-BCD. In this section, we make these results more concrete for two specific adaptive step-size policies that both satisfy (8):

Adaptive 1: for some $\alpha \in (0,1]$,

$$\gamma_k = \alpha \max\{\gamma' - \sum_{t=k-\tau_k}^{k-1} \gamma_t, 0\}. \tag{13}$$

Adaptive 2:

$$\gamma_k = \begin{cases} \frac{\gamma'}{\tau_k+1}, & \frac{\gamma'}{\tau_k+1} \leq \gamma' - \sum_{t=k-\tau_k}^{k-1} \gamma_t, \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

In contrast to existing step-size proposals for asynchronous optimization algorithms, these step-size policies use the actual system delays and do not depend on a (potentially large) upper bound of the maximal delay. When the system operates with small or no delays, these step-sizes approach γ', and if the delays grow large, the step-sizes will be automatically reduced (and occasional updates may be skipped) to guarantee convergence. The performance improvements of these policies over fixed step-size policies depend on the precise nature of the actual delays.

We begin by proving that the two adaptive step-size policies are no worse than the state-of-the-art step-sizes (that require knowledge of the maximal delay). As shown in sections 3.2–3.3, the convergence speed depends on the sum of step-sizes. Our first observation is therefore the following.

Proposition 1. *Suppose that $\tau_k \leq \tau$ for all $k \in \mathbb{N}_0$. Under the step-size policy (13), it holds that*

$$\sum_{t=0}^{k} \gamma_t \geq (k+1) \cdot \frac{\alpha\gamma'}{\tau+1}, \tag{15}$$

while the step-size policy (14) guarantees that

$$\sum_{t=0}^{k} \gamma_t \geq (k+1) \cdot \frac{\tau\gamma'}{(\tau+1)^2}. \tag{16}$$

Proof. See Appendix D. □

The lower bounds in Proposition 1 are comparable with $k+1$ applications of the state-of-the-art fixed step-sizes for PIAG (Sun et al., 2019; Deng et al., 2020) and for Async-BCD (Davis, 2016), respectively. This suggests that the adaptive step-size policies should be able to guarantee the same convergence rate. The next result shows that this is indeed the case.

Corollary 1. *Suppose that $\tau_k \leq \tau$ for all $k \in \mathbb{N}_0$ and that the step-size is determined using either (13) or (14). Then*

- *for PIAG under the conditions of Theorem 2, in case (1) $\min_{1 \leq t \leq k} \|\nabla f(x_t) + \xi_t\|^2 = O(1/k)$, in case (2) $P(x_k) - P^\star = O(1/k)$, and in case (3) $P(x_k) - P^\star \leq O(\lambda^k)$ for some $\lambda \in (0,1)$.*
- *for Async-BCD under the conditions in Theorem 3, $\min_{1 \leq t \leq k} \mathbb{E}[\|\tilde{\nabla} P(x_t)\|^2] = O(1/k)$.*

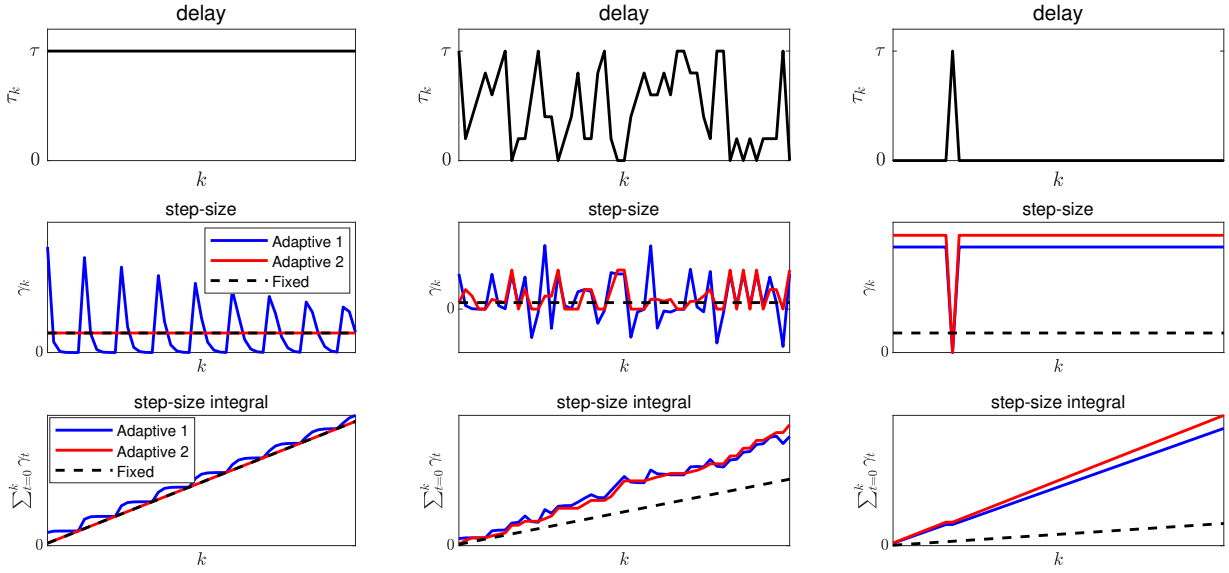
Proof. Immediate from Theorem 2–3 and Prop. 1. □

Although the two adaptive step-sizes do not rely on the delay bound, the rates in Corollary 1 still reach the best-known order compared to related works on PIAG (Aytekin et al., 2016; Vanli et al., 2018; Sun et al., 2019; Deng et al., 2020; Feyzmahdavian & Johansson, 2021) and Async-BCD (Davis, 2016; Sun et al., 2017; Liu et al., 2014; Liu & Wright, 2015) that use such information in their step-sizes.

On the other hand, there are time-varying delays for which the adaptive step-sizes are guaranteed to perform much better than the fixed step-sizes. At the extreme, if the worst-case delay only occurs once and the system operates without delays afterwards (we call this a "burst" delay) the adaptive step-sizes will run with step-size γ'. The sum of step-sizes then tends to a value that is $\tau+1$ times larger than for the fixed step-sizes, with a corresponding speed-up.

To obtain a more balanced comparison, we simulate the two adaptive step-size policies under the following delays:

Figure 1. Comparison of delay-adaptive step-size and fixed step-size in delay models. The legends in (b),(c) follow those in (a).



(a) constant delay (b) random delay (c) burst delay

1) constant: $\tau_k = \tau$.

2) random: τ_k is drawn from $[0, \tau]$ uniformly at random.

3) burst: $\tau_k = \tau$ at one epoch and $\tau_k = 0$ otherwise.

and compare these with the fixed step-size $\gamma_k = \gamma'/(\tau+1)$. This step-size satisfies (8) and is comparable to state-of-the-art fixed step-sizes for PIAG and Async-BCD.

We visualize the three delay models, the step-size γ_k, and the step-size integral $\sum_{t=0}^{k} \gamma_t$ in Figure 1, in which we set $\alpha = 0.9$ in Adaptive 1 and $\tau = 5$ in all three models. We can make the following observations:

- In all three delay models, the sum of step-sizes for the two adaptive policies are at least similar to that of the fixed step-size, which validates Proposition 1.
- The adaptive policies show the greatest superiority compared to the fixed step-size under the burst delay, where the sum is asymptotically $\alpha(\tau+1)$ and $\tau+1$ times that of the fixed step-size, respectively.
- When the proportion of small delays increases (constant→random→burst), so does the sum of step-sizes for the two delay-adaptive policies, reflecting their excellent adaption abilities to the true delay.
- Adaptive 2 is smoother and closer to its average behaviour than Adaptive 1, which often implies better robustness against noise.

4. Numerical experiments

Although the case for delay-adaptive step-sizes should be clear by now, we also demonstrate the end-effect on a simple machine learning problem. We consider classification problem on the training data sets of RCV1 (Lewis et al., 2004), MNIST (Deng, 2012), and CIFAR100 (Krizhevsky et al., 2009), using the regularized logistic regression model:

$$f(x) = \frac{1}{N}\sum_{i=1}^{N}\left(\log(1+e^{-b_i(a_i^T x)}) + \frac{\lambda_2}{2}\|x\|^2\right),$$
$$R(x) = \lambda_1\|x\|_1,$$

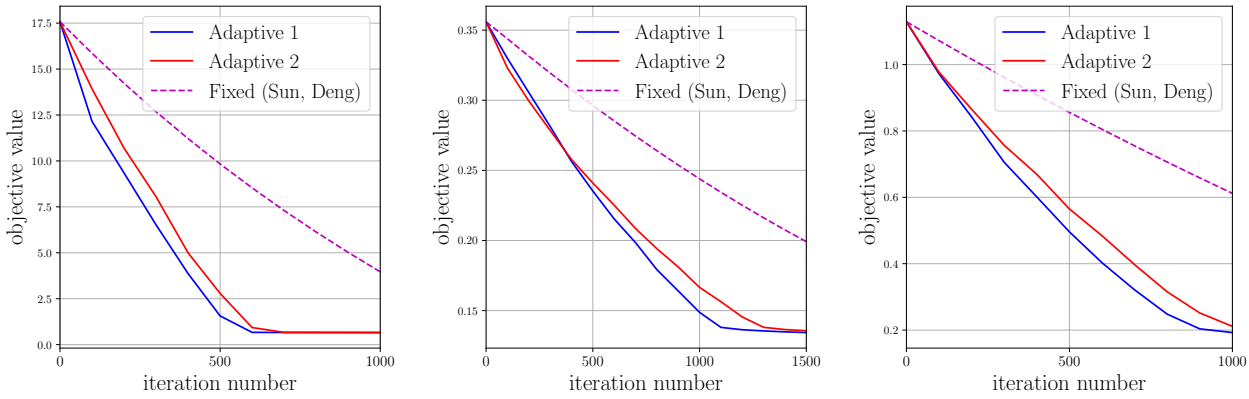
where a_i is the feature of the ith sample, b_i is the corresponding label, and N is the number of samples. We pick $(\lambda_1, \lambda_2) = (10^{-3}, 10^{-4})$ for all three datasets. We run both PIAG and Async-BCD on a 10-core machine and compare the performance of delay-adaptive step-sizes and fixed step-sizes. In the first adaptive policy (Adaptive 1), we let $\alpha = 0.9$.

4.1. PIAG

We split the samples in each data set into $n = 8$ batches and assign each batch to a single worker. We choose one core as master and 8 cores as workers.

We compare the two delay-adaptive step-sizes with $\gamma' = \frac{h}{L}$ against the fixed step-size $\gamma_k = \frac{h}{L(\tau+1/2)}$ from Sun et al. (2019); Deng et al. (2020), where $h = 0.99$ for all three step-

Figure 2. Convergence of PIAG



(a) RCV1 (b) MNIST (c) CIFAR100

sizes (larger step-sizes usually lead to faster convergence). In each iteration, the master updates all mini-batch gradients that it has received. The distributions of the realized delays $\{\tau_k\}$ are plotted in Figure 3(a). Note that the maximal delays for the three datasets, 19, 27, and 28 iterations, respectively, are much larger than the typical τ_k's (over 94% τ_k's are smaller than or equal to 10, 19, 18, respectively, for the three data sets). Moreover, in repeated runs of this experiment, the delay distribution remained similar, but the maximum delay varied and could be as large as 50, which reinforces our message that the maximum delay is hard to estimate and can be much larger than the average delay.

The objective value of PIAG with the three step-sizes is shown in Figure 2. Clearly, PIAG converges much faster under the delay-adaptive step-sizes than under the fixed step-size on all data sets. For example in Figure 2(a), compared to the fixed step-size, PIAG with Adaptive 1 and Adaptive 2 only need approximately $1/3$ and $1/2$ the number of iterations, respectively, to achieve the objective value of 7.5. This demonstrates the effectiveness of our adaptive policies.

4.2. Async-BCD

We use $n = 8$ workers and split x into $m = 20$ blocks almost evenly, with some blocks having one dimension more than the others. We compare the two delay-adaptive step-sizes with $\gamma' = \frac{h}{\hat{L}}$ against the fixed step-sizes $\gamma_k = \frac{h}{L(\tau+1/2)}$ from Sun et al. (2017) and $\gamma_k = \frac{h}{\hat{L}+2L\tau/\sqrt{m}}$ from Davis (2016). In all cases, we use $h = 0.99$.

Figure 4 plots the objective value of Async-BCD with the aforementioned step-sizes. For all datasets, Async-BCD needs a substantially longer time to converge under the fixed step-sizes than under the adaptive policies. This exhibits once again the advantages of our delay-adaptive step-sizes. The distributions of the realized $\{\tau_k\}$ for the three data sets are plotted in Figure 3(b), where the maximal delays are 23, 19, and 14 for RCV1, MNIST, and CIFAR100, respectively. Once again, these are much larger than the typical τ_k's (over 95% τ_k's are smaller than 8, 10, 9, respectively). Moreover, in repeated runs of this experiment, the delay distribution was similar but the maximal delay could be as large as 55.

5. Conclusions

We have shown that it is possible to design, implement and analyze asynchronous optimization algorithms that adapt to the true system delays. This is a significant departure from the state-of-the-art, that rely on an (often conservative) upper bound of the system delays and use fixed learning rates that are tuned to the worst-case situation.

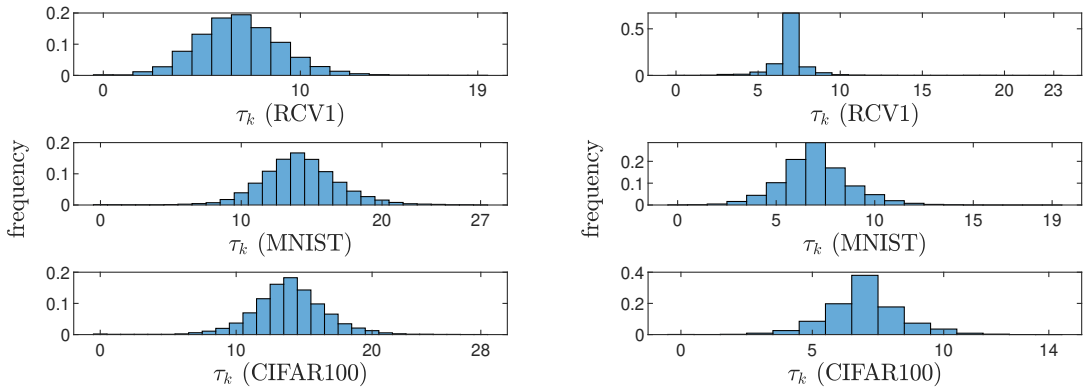
Although many of the principles that we have put forward apply to broad classes of algorithms and systems, we have provided detailed treatments of two specific algorithms: PIAG and Async-BCD. Explicit convergence rate bounds and numerical experiments on different data sets and delay traces demonstrate substantial advantages over the state-of-the-art.

Future work includes developing delay-adaptive step-sizes for other asynchronous algorithms such as Asynchronous SGD (Dean et al., 2012; Recht et al., 2011; Sra et al., 2016) and extending the adaptive mechanism to also estimate the Lipschitz constant (and possibly other parameters) on-line.

Acknowledgements

This work was supported in part by the funding from Digital Futures, Sweden, and in part by the Swedish Research Council (Vetenskapsrådet) under grants 2019-05319 and 2020-03607. We thank the anonymous reviewers for their detailed and valuable feedback.

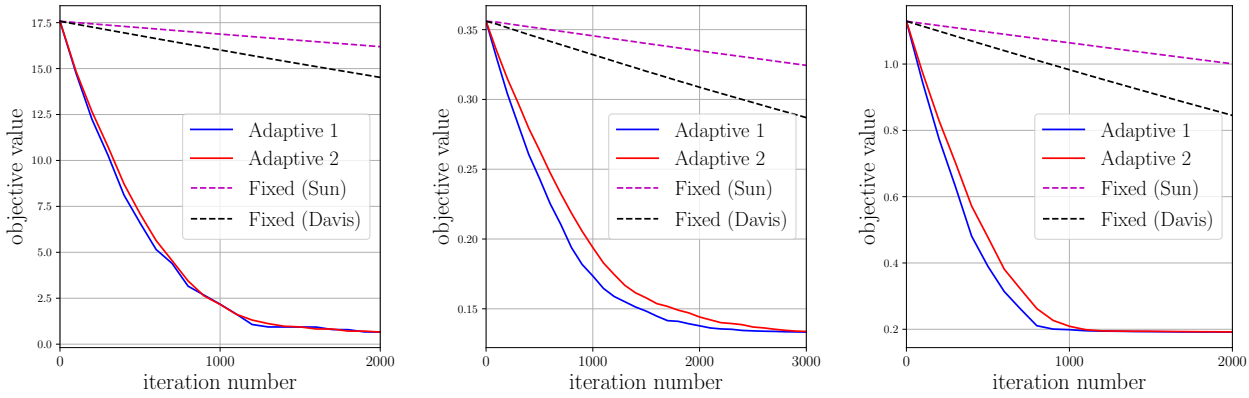
Figure 3. Delay distribution (the largest xtick is the observed maximal delay)



(a) PIAG (8 workers)

(b) Async-BCD (8 workers)

Figure 4. Convergence of Async-BCD.



(a) RCV1

(b) MNIST

(c) CIFAR100

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.

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

Blatt, D., Hero, A. O., and Gauchman, H. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

Cannelli, L., Facchinei, F., Kungurtsev, V., and Scutari, G. Asynchronous parallel algorithms for nonconvex big-data optimization: Model and convergence. *arXiv preprint arXiv:1607.04818*, 2016.

Davis, D. The asynchronous palm algorithm for nonsmooth nonconvex problems. *arXiv preprint arXiv:1604.00526*, 2016.

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., et al. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems-Volume 1*, pp. 1223–1231, 2012.

Deng, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Deng, X., Sun, T., Liu, F., and Huang, F. PRIAG: Proximal reweighted incremental aggregated gradient algorithm for distributed optimizations. In *Algorithms and Architectures for Parallel Processing*, pp. 495–511, 2020.

Feyzmahdavian, H. R. and Johansson, M. Asynchronous iterations in optimization: New sequence results and sharper algorithmic guarantees. *arXiv preprint arXiv:2109.04522*, 2021.

Gurbuzbalaban, M., Ozdaglar, A., and Parrilo, P. A. On the convergence rate of incremental aggregated gradient algorithms. *SIAM Journal on Optimization*, 27(2):1035–1048, 2017.

Hannah, R. and Yin, W. On unbounded delays in asynchronous parallel fixed-point algorithms. *Journal of Scientific Computing*, 76(1):299–326, 2018.

Hong, M., Wang, X., Razaviyayn, M., and Luo, Z.-Q. Iteration complexity analysis of block coordinate descent methods. *Mathematical Programming*, 163(1-2):85–114, 2017.

Karimi, H., Nutini, J., and Schmidt, M. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 795–811. Springer, 2016.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. Improved asynchronous parallel optimization analysis for stochastic incremental methods. *Journal of Machine Learning Research*, 2018.

Lewis, D. D., Yang, Y., Russell-Rose, T., and Li, F. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

Li, M., Zhou, L., Yang, Z., Li, A., Xia, F., Andersen, D. G., and Smola, A. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, pp. 2, 2013.

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

Liu, J., Wright, S., Ré, C., Bittorf, V., and Sridhar, S. An asynchronous parallel stochastic coordinate descent algorithm. In *International Conference on Machine Learning*, pp. 469–477. PMLR, 2014.

Mishchenko, K., Iutzeler, F., Malick, J., and Amini, M.-R. A delay-tolerant proximal-gradient algorithm for distributed learning. In *International Conference on Machine Learning*, pp. 3587–3595. PMLR, 2018.

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.

Polyak, B. T. Introduction to optimization. optimization software. *Inc., Publications Division, New York*, 1, 1987.

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

Roux, N. L., Schmidt, M., and Bach, F. A stochastic gradient method with an exponential convergence rate for finite training sets. *arXiv preprint arXiv:1202.6258*, 2012.

Sra, S., Yu, A. W., Li, M., and Smola, A. Adadelay: Delay adaptive distributed stochastic optimization. In *Artificial Intelligence and Statistics*, pp. 957–965. PMLR, 2016.

Sun, T., Hannah, R., and Yin, W. Asynchronous coordinate descent under more realistic assumption. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6183–6191, 2017.

Sun, T., Sun, Y., Li, D., and Liao, Q. General proximal incremental aggregated gradient algorithms: Better and novel results under general scheme. *Advances in Neural Information Processing Systems*, 32:996–1006, 2019.

Vanli, N. D., Gurbuzbalaban, M., and Ozdaglar, A. Global convergence rate of proximal incremental aggregated gradient methods. *SIAM Journal on Optimization*, 28(2): 1282–1300, 2018.

Wai, H.-T., Shi, W., Uribe, C. A., Nedić, A., and Scaglione, A. Accelerating incremental gradient optimization with curvature information. *Computational Optimization and Applications*, 76(2):347–380, 2020.

A. Proof of Theorem 1

Dividing both sides of (9) by Q_{k+1} and summing the resulting equation from $k=0$ to $k=K-1$, we obtain

$$\frac{V_K}{Q_K}+\sum_{k=1}^{K}\frac{X_k}{Q_k}\leq V_0+\sum_{k=0}^{K-1}\left(-\frac{r_k}{Q_{k+1}}W_k+\frac{p_k}{Q_{k+1}}\sum_{\ell=k-\tau_k}^{k-1}W_\ell\right). \tag{17}$$

Define $\mathcal{T}_\ell:=\{t\in\mathbb{N}_0:\ell\in[t-\tau_t,t-1]\}$. We have

$$\sum_{k=0}^{K-1}\frac{p_k}{Q_{k+1}}\sum_{\ell=k-\tau_k}^{k-1}W_\ell\leq\sum_{\ell=0}^{K-2}\left(\sum_{k\in\mathcal{T}_\ell}\frac{p_k}{Q_{k+1}}\right)W_\ell. \tag{18}$$

To see (18), note that in the left-hand side, W_ℓ occurs only if $\ell\in\{0,1,\ldots,K-2\}$ and $\frac{p_k}{Q_{k+1}}W_\ell$ occurs only if $k\in\mathcal{T}_\ell$. Fix $\ell\in\mathbb{N}_0$. For any $k\in\mathcal{T}_\ell$, because $\ell\in[k-\tau_k,k-1]$ and because of (10), either $p_k=0$ or

$$\frac{p_k}{Q_{k+1}}\leq\frac{r_\ell}{Q_{\ell+1}}-\sum_{t=\ell+1}^{k-1}\frac{p_t}{Q_{t+1}}.$$

Let $k':=\max\{k\in\mathcal{T}_\ell:p_k>0\}$. By the above equation,

$$\frac{r_\ell}{Q_{\ell+1}}\geq\sum_{t=\ell+1}^{k'}\frac{p_t}{Q_{t+1}}\geq\sum_{t\in\mathcal{T}_\ell}\frac{p_t}{Q_{t+1}}.$$

Substituting (18) and the above equation into (17) gives

$$\frac{V_K}{Q_K}+\sum_{k=1}^{K}\frac{X_k}{Q_k}\leq V_0-\frac{r_{K-1}}{Q_K}W_{K-1}\leq V_0,$$

which derives the result.

B. Proof of Theorem 2

For any $k\in\mathbb{N}$, if $\gamma_{k-1}>0$, then

$$\xi_k=-\frac{1}{\gamma_{k-1}}(x_k-x_{k-1})-g_{k-1}.$$

Otherwise, ξ_k can be any subgradient of R at x_k. By the first-order optimality condition of (4),

$$\xi_k\in\partial R(x_k),\ \forall k\in\mathbb{N}. \tag{19}$$

The proof mainly uses Theorem 1 and the following lemma, which shows that some quantities in PIAG satisfy the asynchronous sequence (9).

Lemma 1. *Suppose that all the conditions in Theorem 2 hold. Then, the asynchronous sequence (9) holds with*

$$W_k=\begin{cases}\frac{1}{\gamma_k}\|x_{k+1}-x_k\|^2, & \gamma_k>0,\\ 0, & \gamma_k=0,\end{cases}\quad\forall k\in\mathbb{N}_0,$$

and

(1) ***Non-convex****:*

$$X_{k+1}=\frac{\gamma_k}{2}\frac{1-h}{h^2-h+1}\|\nabla f(x_{k+1})+\xi_{k+1}\|^2,$$
$$V_k=P(x_k)-P^\star,\ p_k=\frac{\gamma_k hL}{2},$$
$$q_k=1,\ r_k=\frac{h^2}{2}-p_k.$$

(2) ***Convex****: If each $f^{(i)}$ is convex, then*

$$X_k = 0,\ V_k = a_k(P(x_k) - P^\star) + \frac{1}{2}\|x_k - x^\star\|^2,$$
$$p_k = \frac{\gamma_k}{2}(a_k L + 1),\ r_k = \frac{a_k}{2} - p_k,\ q_k = 1,$$

where $a_k = \frac{h(h+1)}{L(1-h)} + \sum_{\ell=0}^{k-1}\gamma_\ell$.

(3) ***PL****: If P satisfies the proximal PL-condition (2), then*

$$V_k = P(x_k) - P^\star,\ q_k = \frac{1}{1 + \frac{c\sigma(1-\tilde{h})}{\tilde{h}^2-\tilde{h}+1}\gamma_k},$$
$$X_{k+1} = 0,\ p_k = \frac{\gamma_k \tilde{h} L}{2},\ r_k = \frac{q_k \tilde{h}^2}{2} - \frac{\gamma_k \tilde{h} L}{2},$$

where $\tilde{h} = \frac{1+h}{2}$ and $c = \min\left(1, \frac{1-h}{2h}\frac{L}{\sigma}\right)$.

In all the three cases, either $p_k = 0$ or (10) holds.

Using Lemma 1 and Theorem 1, the result on nonconvex and convex case in Theorem 2 is straightforward. To see the proximal-PL case in Theorem 2, note that because $c \le 1$, $\gamma_k \le \gamma' = \frac{h}{L} \le \frac{\tilde{h}}{L}$, and $\sigma \le L$,

$$\frac{c\sigma(1-\tilde{h})}{\tilde{h}^2-\tilde{h}+1}\gamma_k \le \frac{c\sigma}{L}\frac{\tilde{h}(1-\tilde{h})}{\tilde{h}^2-\tilde{h}+1} \le \frac{\tilde{h}(1-\tilde{h})}{\tilde{h}^2-\tilde{h}+1} \le \frac{1}{3}.$$

In addition, for any $\epsilon \in (0, 1/3]$, $\frac{1}{1+\epsilon} \le 1 - \frac{3}{4}\epsilon \le e^{-\frac{3}{4}\epsilon}$. Therefore,

$$\frac{1}{1 + \frac{c\sigma(1-\tilde{h})}{\tilde{h}^2-\tilde{h}+1}\gamma_k} \le e^{-\frac{3}{4}\frac{c\sigma(1-\tilde{h})}{\tilde{h}^2-\tilde{h}+1}\gamma_k},$$

which further gives

$$Q_k \le e^{-\frac{3c\sigma(1-\tilde{h})}{4(\tilde{h}^2-\tilde{h}+1)}\sum_{t=0}^{k-1}\gamma_t}.$$

Using the above equation and (11), we obtain the result.

B.1. Proof of Lemma 1

When $\gamma_k = 0$, $p_k = 0$ in all three cases. Below, we assume $\gamma_k > 0$ and prove (10) in all the three cases.

B.1.1. Proof of the nonconvex case

We first prove that for any $k \in \mathbb{N}_0$,

$$P(x_{k+1}) - P^\star - (P(x_k) - P^\star) \le \frac{1}{2}\gamma_k h L \sum_{j=k-\tau_k}^{k-1} W_j - \frac{\gamma_k}{2}\|\nabla f(x_k) + \xi_{k+1}\|^2 - \frac{1-\gamma_k L}{2}W_k. \tag{20}$$

By (19) and the convexity of R,

$$R(x_{k+1}) - R(x_k) \le \langle \xi_{k+1}, x_{k+1} - x_k\rangle. \tag{21}$$

Moreover, f is $L-$smooth due to the L_i-smoothness of each $f^{(i)}$. Then,

$$f(x_{k+1}) - f(x_k) \le \langle \nabla f(x_k), x_{k+1} - x_k\rangle + \frac{L}{2}\|x_{k+1} - x_k\|^2. \tag{22}$$

By adding (21) and (22), we have

$$P(x_{k+1}) - P(x_k) \le \langle \nabla f(x_k) + \xi_{k+1}, x_{k+1} - x_k\rangle + \frac{L}{2}\|x_{k+1} - x_k\|^2, \tag{23}$$

where

$$\begin{aligned}&\langle \nabla f(x_k) + \xi_{k+1}, x_{k+1} - x_k \rangle = \gamma_k \langle \nabla f(x_k) + \xi_{k+1}, \frac{1}{\gamma_k}(x_{k+1} - x_k) \rangle \\ =&\frac{\gamma_k}{2}\|\nabla f(x_k) + \xi_{k+1} + \frac{1}{\gamma_k}(x_{k+1} - x_k)\|^2 - \frac{\gamma_k}{2}\|\nabla f(x_k) + \xi_{k+1}\|^2 - \frac{1}{2\gamma_k}\|x_{k+1} - x_k\|^2.\end{aligned} \tag{24}$$

From the definition of ξ_{k+1}, we have

$$\nabla f(x_k) + \xi_{k+1} + \frac{1}{\gamma_k}(x_{k+1} - x_k) = \nabla f(x_k) - g_k.$$

Substituting this equation into (24) yields

$$\langle \nabla f(x_k) + \xi_{k+1}, x_{k+1} - x_k \rangle \leq \frac{\gamma_k}{2}\|\nabla f(x_k) - g_k\|^2 - \frac{\gamma_k}{2}\|\nabla f(x_k) + \xi_{k+1}\|^2 - \frac{1}{2\gamma_k}\|x_{k+1} - x_k\|^2. \tag{25}$$

By the L_i-smoothness of each $f^{(i)}$ and the definition of g_k,

$$\begin{aligned}\|\nabla f(x_k) - g_k\|^2 &= \|\frac{1}{n}\sum_{i=1}^n (\nabla f^{(i)}(x_k) - \nabla f^{(i)}(x_{k-\tau_k^{(i)}}))\|^2 \\ &= \frac{1}{n^2}\|\sum_{i=1}^n (\nabla f^{(i)}(x_k) - \nabla f^{(i)}(x_{k-\tau_k^{(i)}}))\|^2 \\ &\leq \frac{1}{n}\sum_{i=1}^n \|\nabla f^{(i)}(x_k) - \nabla f^{(i)}(x_{k-\tau_k^{(i)}})\|^2 \\ &\leq \frac{1}{n}\sum_{i=1}^n L_i^2 \|x_k - x_{k-\tau_k^{(i)}}\|^2 \\ &= \frac{1}{n}\sum_{i=1}^n L_i^2 \|\sum_{j=k-\tau_k^{(i)}}^{k-1} (x_{j+1} - x_j)\|^2.\end{aligned} \tag{26}$$

In addition,

$$\begin{aligned}\|\sum_{j=k-\tau_k^{(i)}}^{k-1} (x_{j+1} - x_j)\|^2 \leq &(\sum_{j=k-\tau_k^{(i)}}^{k-1} \|x_{j+1} - x_j\|)^2 = (\sum_{j=k-\tau_k^{(i)}}^{k-1} \sqrt{\gamma_j W_j})^2 \\ \leq &(\sum_{j=k-\tau_k^{(i)}}^{k-1} \sqrt{\gamma_j}^2)(\sum_{j=k-\tau_k^{(i)}}^{k-1} \sqrt{W_j}^2) \\ = &(\sum_{j=k-\tau_k^{(i)}}^{k-1} \gamma_j)(\sum_{j=k-\tau_k^{(i)}}^{k-1} W_j) \leq \frac{h}{L}\sum_{j=k-\tau_k}^{k-1} W_j,\end{aligned} \tag{27}$$

where the second inequality is due to the Cauchy–Schwarz inequality and the last step is due to $\tau_k^{(i)} \leq \tau_k$, (8) with $\gamma' = \frac{h}{L}$, and $\gamma_k > 0$. By (26), (27), and $\frac{1}{n}\sum_{i=1}^n L_i^2 = L^2$,

$$\|\nabla f(x_k) - g_k\|^2 \leq hL \sum_{j=k-\tau_k}^{k-1} W_j.$$

Combining the above equation with (25) and (23) yields (20).

Next, we use (20) to derive (9). The equation (20) can be rewritten as

$$\begin{aligned}&P(x_{k+1}) - P^\star - (P(x_k) - P^\star) \\ \leq &\frac{1}{2}\gamma_k hL \sum_{j=k-\tau_k}^{k-1} W_j - \frac{\gamma_k}{2}\|\nabla f(x_k) + \xi_{k+1}\|^2 - \frac{1 - h^2 - (1-h)\gamma_k L}{2} W_k - \frac{h^2 - \gamma_k hL}{2} W_k.\end{aligned} \tag{28}$$

Because of (8) with $\gamma' = \frac{h}{L}$, we have $\gamma_k \leq \frac{h}{L}$ and

$$\begin{aligned}
&\frac{\gamma_k}{2}\|\nabla f(x_k)+\xi_{k+1}\|^2+\frac{1-h^2-(1-h)\gamma_k L}{2}W_k \\
=&\frac{\gamma_k}{2}(\|\nabla f(x_k)+\xi_{k+1}\|^2+\frac{1-h^2-(1-h)\gamma_k L}{\gamma_k^2 L^2}L^2\|x_{k+1}-x_k\|^2) \\
\geq&\frac{\gamma_k}{2}(\|\nabla f(x_k)+\xi_{k+1}\|^2+\frac{1-h}{h^2}L^2\|x_{k+1}-x_k\|^2),
\end{aligned} \tag{29}$$

where the last step is due to $1-h^2-(1-h)\gamma_k L \geq 1-h^2-(1-h)h = 1-h > 0$ and $\gamma_k^2 L^2 \leq h^2$. By the L-smoothness of f and the AM-GM inequality, for any $\eta > 0$,

$$\begin{aligned}
&\|\nabla f(x_{k+1})+\xi_{k+1}\|^2 \\
=&\|(\nabla f(x_{k+1})-\nabla f(x_k))+(\nabla f(x_k)+\xi_{k+1})\|^2 \\
\leq&(1+\eta)\|\nabla f(x_k)+\xi_{k+1}\|^2+(1+1/\eta)\|\nabla f(x_{k+1})-\nabla f(x_k)\|^2 \\
\leq&(1+\eta)\|\nabla f(x_k)+\xi_{k+1}\|^2+(1+1/\eta)L^2\|x_{k+1}-x_k\|^2.
\end{aligned}$$

Letting $\eta = \frac{h^2}{1-h}$ in the above equation, we have

$$\frac{1-h}{h^2-h+1}\|\nabla f(x_{k+1})+\xi_{k+1}\|^2 \leq \|\nabla f(x_k)+\xi_{k+1}\|^2+\frac{(1-h)L^2}{h^2}\|x_{k+1}-x_k\|^2,$$

which, together with (29) and (28), gives

$$\begin{aligned}
&P(x_{k+1})-P^\star-(P(x_k)-P^\star) \\
\leq&\frac{1}{2}\gamma_k h L\sum_{j=k-\tau_k}^{k-1}W_j-\frac{h^2-\gamma_k h L}{2}W_k-\frac{\gamma_k}{2}\frac{1-h}{h^2-h+1}\|\nabla f(x_{k+1})+\xi_{k+1}\|^2,
\end{aligned} \tag{30}$$

i.e., (9) holds.

Finally, it is straightforward to see that (8) with $\gamma' = h/L$ guarantees (10).

B.1.2. Proof of the convex case

Define $a_0 = \frac{h(h+1)}{L(1-h)}$ and $a_k = a_0 + \sum_{\ell=0}^{k-1}\gamma_\ell$ for all $k \in \mathbb{N}$. Multiplying both sides of (20) by a_k and using $h < 1$ gives

$$a_k(P(x_{k+1})-P^\star)-a_k(P(x_k)-P^\star) \leq \frac{a_k\gamma_k L}{2}\sum_{j=k-\tau_k}^{k-1}W_j-\frac{a_k(1-\gamma_k L)}{2}W_k. \tag{31}$$

In addition, using a similar derivation of equation (47) in (Feyzmahdavian & Johansson, 2021), we have

$$\gamma_k(P(x_{k+1})-P^\star) \leq \gamma_k\sum_{i=1}^n\frac{L_i}{2n}\|x_k-x_{k-\tau_k^{(i)}}\|^2+\frac{1}{2}(\|x_k-x^\star\|^2-\|x_{k+1}-x^\star\|^2-\|x_{k+1}-x_k\|^2),$$

which, together with (27), $h < 1$, and $\frac{1}{n}\sum_{i=1}^n L_i \leq \sqrt{\frac{1}{n}\sum_{i=1}^n L_i^2} = L$, leads to

$$\gamma_k(P(x_{k+1})-P^\star) \leq \frac{1}{2}\gamma_k\sum_{j=k-\tau_k}^{k-1}W_j-\frac{1}{2}\gamma_k W_k+\frac{1}{2}(\|x_k-x^\star\|^2-\|x_{k+1}-x^\star\|^2). \tag{32}$$

Adding (32) with (31) gives

$$\begin{aligned}
&a_{k+1}(P(x_{k+1})-P^\star)+\frac{1}{2}\|x_{k+1}-x^\star\|^2-(a_k(P(x_k)-P^\star)+\frac{1}{2}\|x_k-x^\star\|^2) \\
\leq&\frac{\gamma_k}{2}(a_k L+1)\sum_{j=k-\tau_k}^{k-1}W_j-\frac{1}{2}(a_k(1-\gamma_k L)+\gamma_k)W_k \\
=&\frac{\gamma_k}{2}(a_k L+1)\sum_{j=k-\tau_k}^{k-1}W_j-\frac{1}{2}(a_k-\gamma_k(a_k L+1))W_k,
\end{aligned}$$

i.e., (9) holds.

Finally, we prove (10). In this case, (10) reduces to

$$\sum_{j=\ell}^{k}\gamma_j(a_jL+1)\le a_\ell, \forall \ell\in[k-\tau_k,k],\quad \forall k\in\mathbb{N}_0.$$

Since a_k is monotonically non-decreasing, the above equation is equivalent to

$$\sum_{j=k-\tau_k}^{k}\gamma_j(a_jL+1)\le a_{k-\tau_k},\quad \forall k\in\mathbb{N}_0. \tag{33}$$

Since for each $j\in[k-\tau_k,k]$, $a_j\le a_k=a_{k-\tau_k}+\sum_{j=k-\tau_k}^{k-1}\gamma_t\le a_{k-\tau_k}+\frac{h}{L}$ by (8) and $\gamma'=\frac{h}{L}$, we have

$$\sum_{j=k-\tau_k}^{k}\gamma_j(a_jL+1)\le\frac{h(a_{k-\tau_k}L+h+1)}{L}.$$

Moreover, since $a_{k-\tau_k}\ge a_0$ and $\frac{La_{k-\tau_k}}{h(a_{k-\tau_k}L+h+1)}$ increases at $a_{k-\tau_k}$,

$$\frac{La_{k-\tau_k}}{h(a_{k-\tau_k}L+h+1)}\ge\frac{La_0}{h(a_0L+h+1)}=1.$$

Combining the above two equations, we have (33), so that (10) also holds.

B.1.3. Proof of the proximal PL case

By Appendix G in (Karimi et al., 2016), (2) is equivalent to:

$$\sigma(P(x)-P^\star)\le\frac{\|s\|^2}{2},\ \forall s\in\partial P(x),\ \forall x\in\mathbb{R}^d. \tag{34}$$

Because $\sum_{t=k-\tau_k}^{k}\gamma_t\le\frac{h}{L}\le\frac{\tilde{h}}{L}$, (30) with h being replaced by $\tilde{h}$ also holds by its derivation, which, together with (34) and $c\le1$, yields

$$\begin{aligned}&(1+\frac{c\sigma(1-\tilde{h})\gamma_k}{\tilde{h}^2-\tilde{h}+1})(P(x_{k+1})-P^\star)-(P(x_k)-P^\star)\\&\le\frac{1}{2}\gamma_k\tilde{h}L\sum_{j=k-\tau_k}^{k-1}W_j-\frac{\tilde{h}^2-\gamma_k\tilde{h}L}{2}W_k.\end{aligned} \tag{35}$$

Dividing both sides of (35) by $1+\frac{c\sigma(1-\tilde{h})\gamma_k}{\tilde{h}^2-\tilde{h}+1}$ ensures

$$\begin{aligned}&P(x_{k+1})-P^\star-\frac{P(x_k)-P^\star}{1+\frac{c\sigma(1-\tilde{h})\gamma_k}{\tilde{h}^2-\tilde{h}+1}}\\\le&\frac{\frac{1}{2}\gamma_k\tilde{h}L\sum_{j=k-\tau_k}^{k-1}W_j-\frac{\tilde{h}^2}{2}W_k+\frac{\gamma_k\tilde{h}L}{2}W_k}{1+\frac{c\sigma(1-\tilde{h})\gamma_k}{\tilde{h}^2-\tilde{h}+1}}\\\le&\frac{1}{2}\gamma_k\tilde{h}L\sum_{j=k-\tau_k}^{k-1}W_j+(\frac{\gamma_k\tilde{h}L}{2}-\frac{\tilde{h}^2}{2(1+\frac{c\sigma(1-\tilde{h})\gamma_k}{\tilde{h}^2-\tilde{h}+1})})W_k,\end{aligned}$$

i.e., (9) holds.

Below, we derive (10). Since $Q_{k+1} \leq Q_{t+1}\ \forall t \in [k-\tau_k, k]$, (10) can be guaranteed by

$$\sum_{t=\ell+1}^{k} p_t \leq r_\ell \frac{Q_{k+1}}{Q_{\ell+1}},\ \forall \ell \in [k-\tau_k, k], \quad \forall k \in \mathbb{N}_0. \tag{36}$$

Note that because $Q_{k+1} \leq Q_{\ell+1}$,

$$\begin{aligned} r_\ell \frac{Q_{k+1}}{Q_{\ell+1}} &= \frac{Q_{k+1}}{Q_{\ell+1}}(\frac{\tilde{h}^2 q_\ell}{2} - \frac{\gamma_\ell \tilde{h} L}{2}) \\ &\geq \frac{Q_{k+1}}{Q_{\ell+1}} \frac{\tilde{h}^2 q_\ell}{2} - \frac{\gamma_\ell \tilde{h} L}{2} \\ &\geq (\Pi_{t=\ell}^{k} q_t) \frac{\tilde{h}^2}{2} - \frac{\gamma_\ell \tilde{h} L}{2} \\ &\geq (\Pi_{t=k-\tau_k}^{k} q_t) \frac{\tilde{h}^2}{2} - \frac{\gamma_\ell \tilde{h} L}{2}. \end{aligned} \tag{37}$$

Because $\sigma \leq L$, $\tilde{h} = \frac{1+h}{2}$, $c \leq \frac{1-h}{2h}\frac{L}{\sigma}$, and because of (8) and $\gamma' = \frac{h}{L}$, we have

$$\sum_{t=k-\tau_k}^{k} \gamma_t \leq \frac{h}{L} \leq \frac{\tilde{h}}{L + c\sigma},$$

and therefore,

$$\begin{aligned} \Pi_{t=k-\tau_k}^{k} q_t &= \Pi_{t=k-\tau_k}^{k} \frac{1}{1 + \frac{c\sigma(1-\tilde{h})\gamma_t}{\tilde{h}^2 - \tilde{h} + 1}} \\ &\geq \Pi_{t=k-\tau_k}^{k} (1 - \frac{c\sigma(1-\tilde{h})\gamma_t}{\tilde{h}^2 - \tilde{h} + 1}) \\ &\geq 1 - \sum_{t=k-\tau_k}^{k} \frac{c\sigma(1-\tilde{h})\gamma_t}{\tilde{h}^2 - \tilde{h} + 1} \\ &\geq 1 - \frac{c\sigma}{L + c\sigma} \frac{\tilde{h}(1-\tilde{h})}{\tilde{h}^2 - \tilde{h} + 1} \\ &\geq \frac{L}{L + c\sigma}, \end{aligned} \tag{38}$$

where the last step is due to $\frac{\tilde{h}(1-\tilde{h})}{\tilde{h}^2 - \tilde{h} + 1} \leq 1$ because of $\tilde{h} \in (0, 1)$. By (37), (38), (8), $\gamma' = h/L$, and $c \leq \frac{1-h}{2h}\frac{L}{\sigma}$, for any $\ell \in [k-\tau_k, k]$,

$$\sum_{t=\ell+1}^{k} p_t - r_\ell \frac{Q_{k+1}}{Q_{\ell+1}} \leq \frac{\tilde{h} L}{2} \sum_{t=\ell}^{k} \gamma_t - \frac{1}{2} \frac{L \tilde{h}^2}{L + c\sigma} \leq 0,$$

i.e., (36) holds, which guarantees (10).

C. Proof of Theorem 3

The proof mainly uses Theorem 1 and following Lemma, which indicates that some quantities produced by Async-BCD satisfy (9).

Lemma 2. *Suppose that all the conditions in Theorem 3 hold. Then, (9) holds with*

$$W_k = \begin{cases} \frac{1}{\gamma_k}\mathbb{E}[\|x_{k+1}-x_k\|^2], & \gamma_k > 0, \\ 0, & \gamma_k = 0, \end{cases},$$
$$X_{k+1} = \frac{\gamma_k(1-h)}{4m}\mathbb{E}[\|\tilde{\nabla} P(x_k)\|^2],$$
$$V_k = \mathbb{E}[P(x_k)] - P^\star,\ p_k = \frac{\hat{L}\gamma_k}{2},\ q_k = 1,\ r_k = \frac{h}{2} - p_k.$$

In addition, (10) holds.

By Theorem 1 and Lemma 2, Theorem 3 is straightforward.

C.1. Proof of Lemma 2

If $\gamma_k = 0$, then $p_k = 0$. Below we consider the case where $\gamma_k > 0$ and prove (9) and (10).

Suppose that at the kth iteration, the j'-th block is updated. Define $u_k = \frac{x_k^{(j')} - x_{k+1}^{(j')}}{\gamma_k} - \nabla_{j'} f(\hat{x}_k)$. By the first-order optimality condition of (5),

$$u_k \in \partial R^{(j')}(x_{k+1}^{(j')}),$$

which, together with the convexity of $R^{(j')}$, yields

$$\begin{aligned} R(x_{k+1}) - R(x_k) &= R^{(j')}(x_{k+1}^{(j')}) - R^{(j')}(x_k^{(j')}) \\ &\le \langle u_k, x_{k+1}^{(j')} - x_k^{(j')} \rangle. \end{aligned}$$

By the Lipschitz continuity in Assumption 1,

$$f(x_{k+1}) - f(x_k) \le \langle \nabla_{j'} f(x_k), x_{k+1}^{(j')} - x_k^{(j')} \rangle + \frac{\hat{L}}{2}\|x_{k+1}^{(j')} - x_k^{(j')}\|^2.$$

Adding two equations above gives

$$P(x_{k+1}) - P(x_k) \le \langle \nabla_{j'} f(x_k) + u_k, x_{k+1}^{(j')} - x_k^{(j')} \rangle + \frac{\hat{L}}{2}\|x_{k+1}^{(j')} - x_k^{(j')}\|^2. \tag{39}$$

In the above equation,

$$\begin{aligned} &\langle \nabla_{j'} f(x_k) + u_k, x_{k+1}^{(j')} - x_k^{(j')} \rangle \\ =& \frac{\gamma_k}{2}\|\nabla_{j'} f(x_k) + u_k + \frac{x_{k+1}^{(j')} - x_k^{(j')}}{\gamma_k}\|^2 - \frac{\gamma_k}{2}\|\nabla_{j'} f(x_k) + u_k\|^2 - \frac{1}{2\gamma_k}\|x_{k+1}^{(j')} - x_k^{(j')}\|^2. \end{aligned} \tag{40}$$

By the definition of u_k,

$$\|\nabla_{j'} f(x_k) + u_k + \frac{x_{k+1}^{(j')} - x_k^{(j')}}{\gamma_k}\|^2 = \|\nabla_{j'} f(x_k) - \nabla_{j'} f(\hat{x}_k)\|^2. \tag{41}$$

Substituting (40) and (41) into (39) gives

$$\begin{aligned} &P(x_{k+1}) - P(x_k) \\ \le& \frac{\gamma_k}{2}\|\nabla_{j'} f(x_k) - \nabla_{j'} f(\hat{x}_k)\|^2 - \frac{\gamma_k}{2}\|\nabla_{j'} f(x_k) + u_k\|^2 - \frac{1/\gamma_k - \hat{L}}{2}\|x_{k+1}^{(j')} - x_k^{(j')}\|^2. \end{aligned} \tag{42}$$

In addition,

$$\begin{aligned}
&\gamma_k \|\frac{\text{prox}_{\gamma_k R^{(j')}}(x_k^{(j')} - \gamma_k \nabla_{j'} f(x_k)) - x_k^{(j')}}{\gamma_k}\|^2 \\
\leq &\frac{2}{\gamma_k}\|\text{prox}_{\gamma_k R^{(j')}}(x_k^{(j')} - \gamma_k \nabla_{j'} f(x_k)) - x_{k+1}^{(j')}\|^2 \\
&+ \frac{2}{\gamma_k}\|x_{k+1}^{(j')} - x_k^{(j')}\|^2 \\
\leq &2\gamma_k \|\nabla_{j'} f(\hat{x}_k) - \nabla_{j'} f(x_k)\|^2 + \frac{2}{\gamma_k}\|x_{k+1}^{(j')} - x_k^{(j')}\|^2,
\end{aligned} \tag{43}$$

where the last inequality comes from the non-expansive property of the proximal operator. Multiplying both sides of (43) by $\frac{1-h}{4}$ and adding the resulting equation with (42), we derive

$$\begin{aligned}
P(x_{k+1}) - P(x_k) \leq &-\frac{\gamma_k(1-h)}{4} \\
&\cdot \|\frac{\text{prox}_{\gamma_k R^{(j)}}(x_k^{(j')} - \gamma_k \nabla_{j'} f(x_k)) - x_k^{(j')}}{\gamma_k}\|^2 \\
&+ \frac{\gamma_k(2-h)}{2}\|\nabla_{j'} f(x_k) - \nabla_{j'} f(\hat{x}_k)\|^2 \\
&- \frac{h/\gamma_k - \hat{L}}{2}\|x_{k+1}^{(j')} - x_k^{(j')}\|^2.
\end{aligned} \tag{44}$$

Moreover, by (6), $J_k \subseteq [k-\tau_k, k]$, Assumption 1, the step-size condition (8), $\gamma' = \frac{h}{\hat{L}}$, and the Cauchy-Schwartz inequality,

$$\begin{aligned}
&\|\nabla_{j'} f(x_k) - \nabla_{j'} f(\hat{x}_k)\|^2 \\
= &\|\sum_{t \in J_k} \nabla_{j'} f(x_{t+1}) - \nabla_{j'} f(x_t)\|^2 \\
\leq &\|\sum_{t=k-\tau_k}^{k-1} \nabla_{j'} f(x_{t+1}) - \nabla_{j'} f(x_t)\|^2 \\
\leq &(\sum_{t=k-\tau_k}^{k-1} \|\nabla_{j'} f(x_{t+1}) - \nabla_{j'} f(x_t)\|)^2 \\
= &(\sum_{t=k-\tau_k}^{k-1} \sqrt{\gamma_t}\frac{\|\nabla_{j'} f(x_{t+1}) - \nabla_{j'} f(x_t)\|}{\sqrt{\gamma_t}})^2 \\
\leq &(\sum_{t=k-\tau_k}^{k-1} \gamma_t)\sum_{t=k-\tau_k}^{k-1} \frac{\|\nabla_{j'} f(x_{t+1}) - \nabla_{j'} f(x_t)\|^2}{\gamma_t} \\
\leq &\hat{L}h \sum_{t=k-\tau_k}^{k-1} \frac{\|x_{t+1} - x_t\|^2}{\gamma_t},
\end{aligned} \tag{45}$$

and according to Lemma 1 in (Karimi et al., 2016), because $\gamma_k \leq \frac{1}{\tilde{L}}$,

$$\|\frac{\text{prox}_{\gamma_k R}(x_k - \gamma_k \nabla f(x_k)) - x_k}{\gamma_k}\| \geq \|\tilde{\nabla} P(x_k)\|. \tag{46}$$

Substituting (45) and (46) into (44) and using $(2-h)h \leq 1$, we have

$$\begin{aligned}
&\mathbb{E}[P(x_{k+1})|x_k] - P(x_k) \\
\leq &-\frac{\gamma_k(1-h)}{4m}\|\tilde{\nabla} P(x_k)\|^2 + \frac{\hat{L}\gamma_k}{2}\sum_{t=k-\tau_k}^{k-1} W_t - \frac{h - \hat{L}\gamma_k}{2} W_k.
\end{aligned}$$

Therefore, (9) holds.

D. Proof of Proposition 1

Proof of (15): To derive (15) for adaptive step-size (13), define $\{t_k\}_{k=0}^{\infty}$ as $t_0 = 0$, $t_{k+1} = \min\{t : \ t - \tau_t > t_k\}\ \forall k \in \mathbb{N}_0$ and $N_k = 1 + \max\{j : t_j \leq k\}\ \forall k \in \mathbb{N}_0$. Because $\tau_j \leq \tau\ \forall j \in \mathbb{N}_0$, by the definition of t_j,

$$t_{j+1} \leq t_j + \tau + 1,\ \forall j \in \mathbb{N}_0.$$

In addition, $t_0 = 0$. Then, we have $t_j \leq j(\tau+1)$, which implies $t_{\lfloor k/(\tau+1)\rfloor} \leq k$. By definition of N_k,

$$\begin{aligned} N_k &\geq 1 + \max\{j : t_j \leq k\} \\ &\geq 1 + \lfloor k/(\tau+1)\rfloor \geq \frac{k+1}{\tau+1}. \end{aligned}$$

Note that because $t_k + 1 \leq t_{k+1} - \tau_{t_{k+1}}$,

$$\sum_{j=t_k+1}^{t_{k+1}} \gamma_j \geq \sum_{j=t_{k+1}-\tau_{t_{k+1}}}^{t_{k+1}} \gamma_j.$$

In addition, by the definition of N_k, $t_{N_k-1} \leq k$, which, together with the above equation, gives

$$\begin{aligned} \sum_{j=0}^{k} \gamma_j &\geq \sum_{j=0}^{t_{N_k-1}} \gamma_j = \gamma_0 + \sum_{\ell=0}^{N_k-2} \sum_{j=t_\ell+1}^{t_{\ell+1}} \gamma_j \\ &\geq \gamma_0 + \sum_{\ell=0}^{N_k-2} \sum_{j=t_{\ell+1}-\tau_{t_{\ell+1}}}^{t_{\ell+1}} \gamma_j. \end{aligned} \tag{47}$$

If $\gamma_{t_{\ell+1}} = 0$, then $a_{t_{\ell+1}} < \gamma' - \sum_{j=t_{\ell+1}-\tau_{t_{\ell+1}}}^{t_{\ell+1}-1} \gamma_j$, which implies

$$\sum_{j=t_{\ell+1}-\tau_{t_{\ell+1}}}^{t_{\ell+1}-1} \gamma_j \geq \gamma'.$$

Otherwise, $\gamma_{t_{\ell+1}} = \alpha(\gamma' - \sum_{j=t_{\ell+1}-\tau_{t_{\ell+1}}}^{t_{\ell+1}-1} \gamma_j)$ and

$$\sum_{j=t_{\ell+1}-\tau_{t_{\ell+1}}}^{t_{\ell+1}} \gamma_j = \alpha\gamma' + (1-\alpha) \sum_{j=t_{\ell+1}-\tau_{t_{\ell+1}}}^{t_{\ell+1}-1} \geq \alpha\gamma'.$$

From the above two equations, we have $\sum_{j=t_{\ell+1}-\tau_{\ell+1}}^{t_{\ell+1}} \gamma_j \geq \alpha\gamma'$. In addition, because $\tau_k \in [0,k]\ \forall k \in \mathbb{N}_0$, $\tau_0 = 0$ and $\gamma_0 = \alpha\gamma'$. Substituting these into (47) yields (15).

Proof of (16): We use mathematical induction to prove (16) for adaptive step-size (14). Suppose that the following equation holds at some $k \in \mathbb{N}_0$:

$$\sum_{j=0}^{\ell} \gamma_j \geq \frac{\tau}{\tau+1} \cdot \frac{\gamma'(\ell+1)}{\tau+1}, \forall \ell \leq k-1, \tag{48}$$

which naturally holds when $k = 0$. Below, we prove that (48) holds at $k+1$ by showing

$$\sum_{j=0}^{k} \gamma_j \geq \frac{\tau}{\tau+1} \cdot \frac{\gamma'(k+1)}{\tau+1}. \tag{49}$$

If $\gamma_k = 0$, which is possible only when $\tau_k > 0$, then by (14),

$$\sum_{j=k-\tau_k}^{k-1} \gamma_j > \frac{\tau_k\gamma'}{\tau_k+1} \geq \frac{\tau}{\tau+1} \cdot \frac{(\tau_k+1)\gamma'}{\tau+1},$$

where the last step is due to $\frac{\tau_k}{(\tau_k+1)^2} \geq \frac{\tau}{(\tau+1)^2}$ when $\tau_k \in [1, \tau]$. In addition, because $k - \tau_k - 1 \leq k - 1$, by (48),

$$\sum_{j=0}^{k-\tau_k-1} \gamma_j \geq \frac{\tau}{\tau+1} \cdot \frac{\gamma'(k-\tau_k)}{\tau+1}.$$

Adding the two equations above and using $\gamma_k = 0$, we have (49). If $\gamma_k > 0$, then by (14), $\gamma_k \geq \frac{\gamma'}{\tau_k+1} \geq \frac{\gamma'}{\tau+1}$. In addition, $\sum_{j=0}^{k-1} \gamma_j \geq \frac{\gamma' k}{\tau+1}$ by (48). Then, we have (49), which indicates (48) at $k+1$. Conclude all the above, (48) as well as (16) holds for all $k \in \mathbb{N}_0$.

E. Wall-clock-time Convergence and Comparison with Synchronous Counterparts

We also plot the convergence of PIAG and Async-BCD in terms of wall-clock time and compare them with their synchronous counterparts by solving the same problem in Section 4 under the same experiment settings. We provide the details of the three simulated datasets in Table 1.

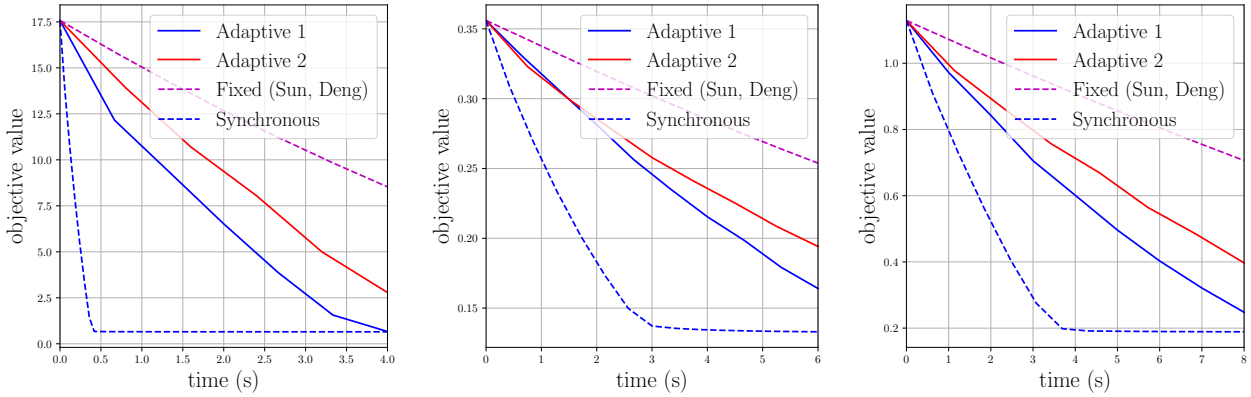
Table 1. data sets

Data set	sample number	feature dimension
RCV1	20242	47236
MNIST	60000	784
CIFAR100	50000	3072

E.1. PIAG

The synchronous version of PIAG is distributed proximal gradient (PG) method, where at each iteration, the master updates after it received mini-batch gradients from all the workers. We set the step-size in distributed PG as $1/L$ and plot the experiment result in Figure 5.

Figure 5. PIAG **vs.** distributed PG.



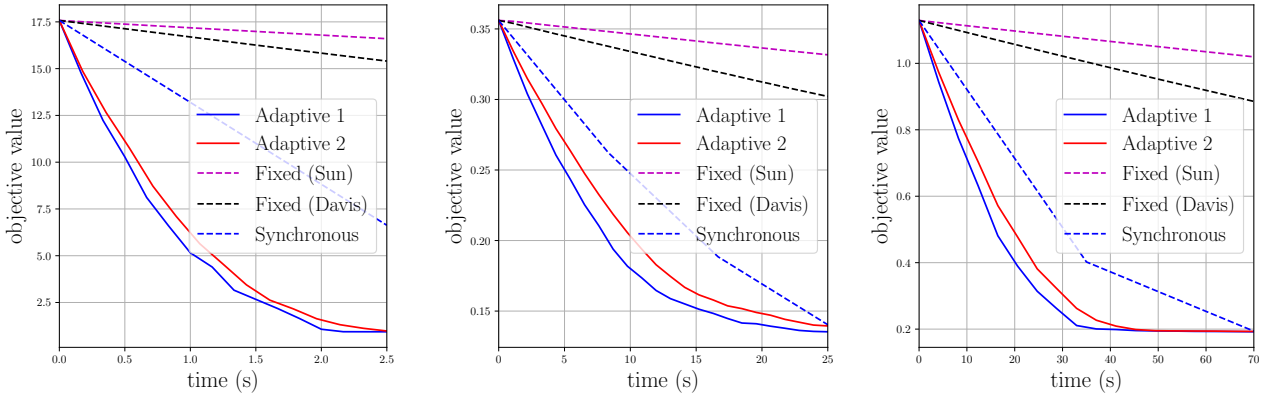
(a) RCV1 (b) MNIST (c) CIFAR100

Observe from Figure 5 that in terms of running time, PIAG with the two adaptive step-sizes still converge significantly faster than that with the fixed step-size. Although intuitively, asynchronous algorithms are supposed to outperform the synchronous version in terms of running time, it is *not* true for PIAG in our experiment. This may mainly because of the small step-sizes in PIAG caused by delays.

E.2. Async-BCD

The synchronous counterpart of Async-BCD is synchronous distributed BCD. We run synchronous distributed BCD with 8 workers and randomly sample 8 blocks over 20 blocks to update at each iteration. We set the step-size in synchronous distributed BCD as $8/\hat{L}$ (still significantly smaller than $1/L$ in the experiment), which is a standard step-size because each iteration of synchronous distributed BCD updates 8 blocks. The experiment result is plotted in Figure 6.

Figure 6. Async-BCD **vs.** synchronous distributed BCD.



(a) RCV1 (b) MNIST (c) CIFAR100

Observe from Figure 6 that Async-BCD with two adaptive step-sizes outperforms not only Async-BCD with fixed step-sizes but also synchronous distributed BCD.