Journal of Machine Learning Research 21 (2020) 1-51 Submitted 6/16; Revised 4/20; Published 6/20

A Class of Parallel Doubly Stochastic Algorithms for Large-Scale Learning

Aryan Mokhtari MOKHTARI@AUSTIN.UTEXAS.EDU
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712, USA

Alec Koppel ALEC.E.KOPPEL.CIV@MAIL.MIL
Computational and Information Sciences Directorate
U.S. Army Research Laboratory
Adelphi, MD 20783, USA

Martin Takáč TAKAC.MT@GMAIL.COM
Industrial and Systems Engineering
Lehigh University,
Bethlehem, PA 18015, USA

Alejandro Ribeiro ARIBEIRO@SEAS.UPENN.EDU
Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104, USA

Editor: Sathiya Keerthi

Abstract

We consider learning problems over training sets in which both, the number of training examples and the dimension of the feature vectors, are large. To solve these problems we propose the random parallel stochastic algorithm (RAPSA). We call the algorithm random parallel because it utilizes multiple parallel processors to operate on a randomly chosen subset of blocks of the feature vector. RAPSA is doubly stochastic since each processor utilizes a random set of functions to compute the stochastic gradient associated with a randomly chosen sets of variable coordinates. Algorithms that are parallel in either of these dimensions exist, but RAPSA is the first attempt at a methodology that is parallel in both the selection of blocks and the selection of elements of the training set. In RAPSA, processors utilize the randomly chosen functions to compute the stochastic gradient component associated with a randomly chosen block. The technical contribution of this paper is to show that this minimally coordinated algorithm converges to the optimal classifier when the training objective is strongly convex. Moreover, we present an accelerated version of RAPSA (ARAPSA) that incorporates the objective function curvature information by premultiplying the descent direction by a Hessian approximation matrix. We further extend the results for asynchronous settings and show that if the processors perform their updates without any coordination the algorithms are still convergent to the optimal argument. RAPSA and its extensions are then numerically evaluated on a linear estimation problem and a binary image classification task using the MNIST handwritten digit dataset.

Keywords: Stochastic optimization, large-scale learning, asynchronous methods, parallel algorithms

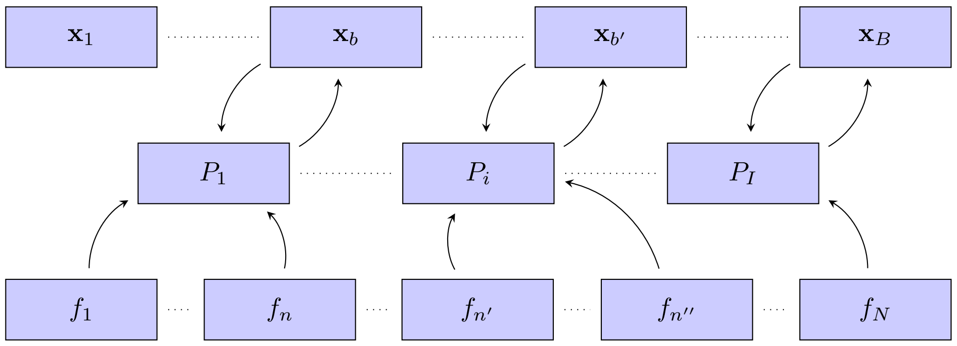


Figure 1: Random parallel stochastic algorithm (RAPSA). At each iteration, processor P_i picks a random block from the set $\{\mathbf{x}_1, \ldots, \mathbf{x}_B\}$ and a random set of functions from the training set $\{f_1, \ldots, f_N\}$. The functions drawn are used to evaluate a stochastic gradient component associated with the chosen block. RAPSA is shown here to converge to the optimal argument $\mathbf{x}^*$ of (1).

1. Introduction

Learning is often formulated as an optimization problem that finds a vector of parameters $\mathbf{x}^* \in \mathbb{R}^p$ that minimizes the average of a loss function across the elements of a training set. For a precise definition consider a training set with N elements and let $f_n : \mathbb{R}^p \to \mathbb{R}$ be a convex loss function associated with the n-th element of the training set. The optimal parameter vector $\mathbf{x}^* \in \mathbb{R}^p$ is defined as the minimizer of the average cost $F(\mathbf{x}) := (1/N)\sum_{n=1}^{N} f_n(\mathbf{x})$,

$$\mathbf{x}^* := \operatorname*{argmin}_{\mathbf{x}} F(\mathbf{x}) := \operatorname*{argmin}_{\mathbf{x}} \frac{1}{N}\sum_{n=1}^{N} f_n(\mathbf{x}). \tag{1}$$

Problems such as support vector machine classification, logistic and linear regression, and matrix completion can be put in the form of problem (1). In this paper, we are interested in large scale problems where both the number of features p and the number of elements N in the training set are very large – which arise, e.g., in text (Sampson et al., 1990), image (Mairal et al., 2010), and genomic (Taşan et al., 2014) processing.

When N and p are large, the parallel processing architecture in Figure 1 becomes of interest. In this architecture, the parameter vector $\mathbf{x}$ is divided into B blocks each of which contains $p_b \ll p$ features and a set of $I \ll B$ processors work in parallel on randomly chosen parameter blocks while using a stochastic subset of elements of the training set. In the schematic shown, Processor 1 fetches functions f_1 and f_n to operate on block $\mathbf{x}_b$ and Processor i fetches functions $f_{n'}$ and $f_{n''}$ to operate on block $\mathbf{x}_{b'}$. Other processors select other elements of the training set and other blocks with the majority of blocks remaining unchanged and the majority of functions remaining unused. The blocks chosen for update and the functions fetched for determination of block updates are selected independently at random in subsequent slots.

Problems that operate on blocks of the parameter vectors *or* subsets of the training set, but not on both, blocks *and* subsets, exist. Block coordinate descent (BCD) is the generic name for methods in which the variable space is divided in blocks that are processed separately. Early versions operate by cyclically updating all coordinates at each step (Luo and Tseng, 1992; Tseng, 2001; Xu and Yin, 2017), while more recent parallelized versions of coordinate descent have been developed to accelerate convergence of BCD (Richtárik and Takáč, 2015; Lu and Xiao, 2013; Nesterov, 2012; Beck and Tetruashvili, 2013). Closer to the architecture in Figure 1, methods in which subsets of blocks are selected at random have also been proposed (Liu et al., 2015; Yang et al., 2013; Nesterov, 2012; Lu and Xiao, 2015). BCD, serial, parallel, or random, can handle cases where the parameter dimension p is large but requires access to all N training samples at each iteration.

Parallel implementations of block coordinate methods have been developed initially in this setting for composite optimization problems (Richtárik and Takáč, 2015). A collection of parallel processors update randomly selected blocks concurrently at each step. Several variants that select blocks in order to maximize the descent at each step are proposed in (Scherrer et al., 2012; Facchinei et al., 2015; Shalev-Shwartz and Zhang, 2013). The aforementioned works require that parallel processors operate on a common time index. In contrast, asynchronous parallel methods, originally proposed in Bertsekas and Tsitsiklis (1989), have been developed to solve optimization problems where processors are *not* required to operate with a common global clock. This work focused on solving a fixed point problem over a separable convex set, but the analysis is more restrictive than standard convexity assumptions. For a standard strongly convex optimization problem, in contrast, Liu et al. (2015) establish linear convergence to the optimum. All of these works are developed for optimization problems with deterministic objectives.

To handle the case where the number of training examples N is very large, methods have been developed to only process a subset of sample points at a time. These methods are known by the generic name of stochastic approximation and rely on the use of stochastic gradients. In plain stochastic gradient descent (SGD), the gradient of the aggregate function is estimated by the gradient of a randomly chosen function f_n (Robbins and Monro, 1951). Since convergence of SGD is slow more often that not, various recent developments have been aimed at accelerating its convergence. These attempts include methodologies to reduce the variance of stochastic gradients (Schmidt et al., 2017; Johnson and Zhang, 2013; Defazio et al., 2014; Mokhtari et al., 2018b) and the use of ideas from quasi-Newton optimization to handle difficult curvature profiles (Schraudolph et al., 2007; Bordes et al., 2009; Mokhtari and Ribeiro, 2014, 2015; Mokhtari et al., 2018a). More pertinent to the work considered here are the use of cyclic block SGD updates (Xu and Yin, 2015) and the exploitation of sparsity properties of feature vectors to allow for parallel updates (Recht et al., 2011). Moreover, in (Matsushima et al., 2017; Xiao et al., 2019), saddle point reformulations of ERM are proposed, and in (Meng et al., 2016), a derivation of Nesterov momentum-type updates, but both of these methods' advantages relative to SGD are awash for dense data. Moreover, since the ERM problem is unconstrained by definition, these reformulations introduce undue complexity into what can be accomplished by unconstrained optimization. These methods are suitable when the number of elements in the training set N is large but don't allow for parallel feature processing unless parallelism is inherent to the problem's structure.

The random parallel stochastic algorithm (RAPSA) proposed in this paper represents the first effort at implementing the architecture in Figure 1 that randomizes over both parameters and sample functions, and may be implemented in parallel. In RAPSA, the functions fetched by a processor are used to compute the stochastic gradient component associated with a randomly chosen block (Section 2). The processors do not coordinate in either choice except to avoid selection of the same block. Our main technical contribution is to show that RAPSA iterates converge to the optimal classifier $\mathbf{x}^*$ when using a sequence of decreasing stepsizes and to a neighborhood of the optimal classifier when using constant stepsizes (Section 5). In the latter case, we further show that the rate of convergence to this optimality neighborhood is linear in expectation. These results are interesting because only a subset of features are updated per iteration and the functions used to update different blocks are, in general, different. We would like to highlight that RAPSA is doubly stochastic since each processor utilizes a random set of functions to compute the stochastic gradient associated with a randomly chosen sets of variable coordinates. We propose two extensions of RAPSA. Firstly, motivated by the improved performance results of quasi-Newton methods relative to gradient methods in online optimization, we propose an extension of RAPSA which incorporates approximate second-order information of the objective, called Accelerated RAPSA. We also consider an extension of RAPSA in which parallel processors are not required to operate on a common time index, which we call Asynchronous RAPSA. We further show how these extensions yield an accelerated doubly stochastic algorithm for an asynchronous system. We establish that the performance guarantees of RAPSA carry through to asynchronous computing architectures. We then numerically evaluate the proposed methods on a large-scale linear regression problem as well as the MNIST digit recognition problem (Section 6).

2. Random Parallel Stochastic Algorithm (RAPSA)

We consider a more general formulation of (1) in which the number of functions f_n is not necessarily finite. Introduce then a random variable $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subset \mathbb{R}^q$ that determines the choice of the random smooth convex function $f(\cdot, \boldsymbol{\theta}) : \mathbb{R}^p \to \mathbb{R}$. We consider the problem of minimizing the expectation of the random functions $F(\mathbf{x}) := \mathbb{E}_{\boldsymbol{\theta}}[f(\mathbf{x}, \boldsymbol{\theta})]$,

$$\mathbf{x}^* := \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^p} F(\mathbf{x}) := \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^p} \mathbb{E}_{\boldsymbol{\theta}} \left[f(\mathbf{x}, \boldsymbol{\theta}) \right]. \tag{2}$$

Problem (1) is a particular case of (2) in which each of the functions f_n is drawn with probability $1/N$. Observe that when $\boldsymbol{\theta} = (\mathbf{z}, \mathbf{y})$ with feature vector $\mathbf{z} \in \mathbb{R}^p$ and target variable $\mathbf{y} \in \mathbb{R}^q$ or $y \in \{0, 1\}$, the formulation in (2) encapsulates generic supervised learning problems such as regression or classification, respectively. We refer to $f(\cdot, \boldsymbol{\theta})$ as instantaneous functions and to $F(\mathbf{x})$ as the average function.

RAPSA utilizes I processors to update a random subset of blocks of the variable $\mathbf{x}$, with each of the blocks relying on a subset of randomly and independently chosen elements of the training set; see Figure 1. Formally, decompose the variable $\mathbf{x}$ into B blocks to write $\mathbf{x} = [\mathbf{x}_1; \ldots; \mathbf{x}_B]$, where block b has length p_b so that we have $\mathbf{x}_b \in \mathbb{R}^{p_b}$. At iteration t, processor i selects a random index b_i^t for updating and a random subset $\boldsymbol{\Theta}_i^t$ of L instantaneous functions. It then uses these instantaneous functions to determine stochastic gradient components for the subset of variables $\mathbf{x}_b = \mathbf{x}_{b_i^t}$ as an average of the components of the gradients of the

Algorithm 1 Random Parallel Stochastic Algorithm (RAPSA)

```
1: for $t = 0, 1, 2, \ldots$ do
2:     loop in parallel, processors $i = 1, \ldots, I$ execute:
3:         Select block $b_i^t \in \{1, \ldots, B\}$ uniformly at random from set of blocks
4:         Choose training subset $\mathbf{\Theta}_i^t$ for block $\mathbf{x}_b$,
5:         Compute stochastic gradient : $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t) = \frac{1}{L} \sum_{\boldsymbol{\theta} \in \mathbf{\Theta}_i^t} \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \boldsymbol{\theta}),\quad b = b_i^t$ [cf. (3)]
6:         Update the coordinates $b_i^t$ of the decision variable $\mathbf{x}_b^{t+1} = \mathbf{x}_b^t - \gamma^t \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$
7:     end loop; Transmit updated blocks $i \in \mathcal{I}^t \subset \{1, \ldots, B\}$ to shared memory
8: end for
```

functions $f(\mathbf{x}^t, \boldsymbol{\theta})$ for $\boldsymbol{\theta} \in \mathbf{\Theta}_i^t$,

$$\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t) = \frac{1}{L} \sum_{\boldsymbol{\theta} \in \mathbf{\Theta}_i^t} \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \boldsymbol{\theta}), \qquad b = b_i^t. \tag{3}$$

Note that L can be interpreted as the size of mini-batch for gradient approximation. The stochastic gradient block in (3) is then modulated by a possibly time varying stepsize γ^t and used by processor i to update the block $\mathbf{x}_b = \mathbf{x}_{b_i^t}$

$$\mathbf{x}_b^{t+1} = \mathbf{x}_b^t - \gamma^t \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t), \qquad\qquad b = b_i^t. \tag{4}$$

RAPSA is defined by the joint implementation of (3) and (4) across all I processors, and is summarized in Algorithm 1. We would like to emphasize that the number of updated blocks which is equivalent to the number of processors I is not necessary equal to the total number of blocks B. In other words, we may update only a subset of coordinates $I/B < 1$ at each iteration. We define $r := I/B$ as the ratio of the updated blocks to the total number of blocks which is smaller than 1.

The selection of blocks is coordinated so that no processors operate in the same block. The selection of elements of the training set is uncoordinated across processors. The fact that at any point in time a random subset of blocks is being updated utilizing a random subset of elements of the training set means that RAPSA requires coordination between processors only for the choice of blocks. The contribution of this paper is to show that this very lean algorithm converges to the optimal argument $\mathbf{x}^*$ as we show in Section 5.

3. Accelerated Random Parallel Stochastic Algorithm (ARAPSA)

As we mentioned in Section 2, RAPSA operates on first-order information which may lead to slow convergence in ill-conditioned problems. We introduce Accelerated RAPSA (ARAPSA) as a parallel doubly stochastic algorithm that incorporates second-order information of the objective by separately approximating the function curvature for each block. We do this by implementing the oLBFGS algorithm for different blocks of the variable $\mathbf{x}$. For related approaches, see, for instance, Broyden et al. (1973); Byrd et al. (1987); Dennis and Moré (1974); Li and Fukushima (2001). Define $\hat{\mathbf{B}}_b^t$ as an approximation for the Hessian inverse

Algorithm 2 Computation of the ARAPSA step $\hat{\mathbf{d}}_b^t = \hat{\mathbf{B}}_b^t \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ for block $\mathbf{x}_b$.

```
1: function $\hat{\mathbf{d}}_b^t = \mathbf{q}^s = \text{ARAPSA Step}\Big(\hat{\mathbf{B}}_b^{t,0},\ \mathbf{p}^0 = \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t),\ \{\mathbf{v}_b^u, \hat{\mathbf{r}}_b^u\}_{u=\hat{t}_b-s+1}^{\hat{t}_b}\Big)$
2: for $u = 0, 1, \ldots, s-1$ do {Loop to compute constants $\alpha^u$ and sequence $\mathbf{p}^u$}
3:     Compute and store scalar $\alpha^u = \hat{\rho}_b^{\hat{t}_b-u} (\mathbf{v}_b^{\hat{t}_b-u})^T \mathbf{p}^u$
4:     Update sequence vector $\mathbf{p}^{u+1} = \mathbf{p}^u - \alpha^u \hat{\mathbf{r}}_b^{\hat{t}_b-u}$.
5: end for
6: Multiply $\mathbf{p}^s$ by initial matrix: $\mathbf{q}^0 = \hat{\mathbf{B}}_b^{t,0} \mathbf{p}^s$
7: for $u = 0, 1, \ldots, s-1$ do {Loop to compute constants $\beta^u$ and sequence $\mathbf{q}^u$}
8:     Compute scalar $\beta^u = \hat{\rho}_b^{\hat{t}_b-s+u+1} (\hat{\mathbf{r}}_b^{\hat{t}_b-s+u+1})^T \mathbf{q}^u$
9:     Update sequence vector $\mathbf{q}^{u+1} = \mathbf{q}^u + (\alpha^{s-u-1} - \beta^u) \mathbf{v}_b^{\hat{t}_b-s+u+1}$
10: end for {return $\hat{\mathbf{d}}_b^t = \mathbf{q}^s$}
```

of the objective function that corresponds to the block b with the corresponding variable $\mathbf{x}_b$. If we consider b_i^t as the block that processor i chooses at step t, then the update of ARAPSA is defined as multiplication of the descent direction of RAPSA by $\hat{\mathbf{B}}_b^t$, i.e.,

$$\mathbf{x}_b^{t+1} \ = \ \mathbf{x}_b^t - \gamma^t \ \hat{\mathbf{B}}_b^t \ \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t), \qquad b = b_i^t. \tag{5}$$

Subsequently, we define the $\hat{\mathbf{d}}_b^t := \hat{\mathbf{B}}_b^t \ \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$. We next detail how to properly specify the block approximate Hessian $\hat{\mathbf{B}}_b^t$ so that it behaves in a manner comparable to the true Hessian. To do so, define for each block coordinate $\mathbf{x}_b$ at step t the variable variation $\mathbf{v}_b^t$ and the stochastic gradient variation $\hat{\mathbf{r}}_b^t$ as

$$\mathbf{v}_b^t \ = \ \mathbf{x}_b^{t+1} - \mathbf{x}_b^t, \qquad \hat{\mathbf{r}}_b^t \ = \ \nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+1}, \mathbf{\Theta}_i^t) - \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t). \tag{6}$$

Observe that the stochastic gradient variation $\hat{\mathbf{r}}_b^t$ is defined as the difference of stochastic gradients at times $t+1$ and t corresponding to the block $\mathbf{x}_b$ for a common set of realizations $\mathbf{\Theta}_i^t$. The term $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ is the same as the stochastic gradient used at time t in (5), while $\nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+1}, \mathbf{\Theta}_i^t)$ is computed only to determine the stochastic gradient variation $\hat{\mathbf{r}}_b^t$. An alternative and perhaps more natural definition for the stochastic gradient variation is $\nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+1}, \mathbf{\Theta}_i^{t+1}) - \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$. However, as pointed out by Schraudolph et al. (2007), this formulation is insufficient for establishing the convergence of stochastic quasi-Newton methods. We proceed to developing a block-coordinate quasi-Newton method by first noting an important property of the true Hessian, and design our approximate scheme to satisfy this property. The secant condition may be interpreted as stating that the stochastic gradient of a quadratic approximation of the objective function evaluated at the next iteration agrees with the stochastic gradient at the current iteration. We select a Hessian inverse approximation matrix associated with block $\mathbf{x}_b$ such that it satisfies the secant condition $\hat{\mathbf{B}}_b^{t+1} \hat{\mathbf{r}}_b^t = \mathbf{v}_b^t$, and thus behaves in a comparable manner to the true block Hessian.

The oLBFGS Hessian inverse update rule maintains the secant condition at each iteration by using information of the last $s \geq 1$ pairs of variable and stochastic gradient variations for the last s time indices that the block $\mathbf{x}_b$ is updated. As an example assume that $s = 3$ and block $\mathbf{x}_b = \mathbf{x}_1$ is chosen to be updated at time $t = 10$. Further, consider that

Algorithm 3 Accelerated Random Parallel Stochastic Algorithm (ARAPSA)

```
1:  for $t = 0, 1, 2, \ldots$ do
2:      loop in parallel, processors $i = 1, \ldots, I$ execute:
3:          Select block $b_i^t$ uniformly at random from set of blocks $\{1, \ldots, B\}$
4:          Choose a set of realizations $\mathbf{\Theta}_i^t$ for the block $\mathbf{x}_b$
5:          Compute stochastic gradient : $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t) = \frac{1}{L} \sum_{\boldsymbol{\theta} \in \mathbf{\Theta}_i^t} \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \boldsymbol{\theta})$ [cf. (3)]
6:          Compute the initial Hessian inverse approximation: $\hat{\mathbf{B}}_b^{t,0} = \eta_b^t \mathbf{I}$
7:          Compute descent direction:
                $\hat{\mathbf{d}}_b^t = \text{ARAPSA Step}\left(\hat{\mathbf{B}}_b^{t,0},\ \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t),\ \{\mathbf{v}_b^u, \hat{\mathbf{r}}_b^u\}_{u=\hat{t}_b-s+1}^{\hat{t}_b}\right)$
8:          Update the coordinates of the decision variable $\mathbf{x}_b^{t+1} = \mathbf{x}_b^t - \gamma^t \hat{\mathbf{d}}_b^t$
9:          Compute updated stochastic gradient: $\nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+1}, \mathbf{\Theta}_i^t) = \frac{1}{L} \sum_{\boldsymbol{\theta} \in \mathbf{\Theta}_i^t} \nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+1}, \boldsymbol{\theta})$
10:         Update variations $\mathbf{v}_b^t = \mathbf{x}_b^{t+1} - \mathbf{x}_b^t$ and $\hat{\mathbf{r}}_i^t = \nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+1}, \mathbf{\Theta}_i^t) - \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ [ cf.(6)]
11:     end loop; Transmit updated blocks $i \in \mathcal{I}^t \subset \{1, \ldots, B\}$ to shared memory
12: end for
```

the block $\mathbf{x}_b = \mathbf{x}_1$ was previously updated at time indices $t = 8$, $t = 5$, $t = 4$, and $t = 1$. Then to update $\mathbf{x}_b = \mathbf{x}_1$ at time $t = 10$ we use the variable and gradient variations vectors $\{\mathbf{v}_b^u, \hat{\mathbf{r}}_b^u\}_{u \in \{8,5,4\}}$. To properly define this notation we introduce the subset $\mathcal{T}_b(t)$ which contains the time indices that block $\mathbf{x}_b$ is updated before step t. According to this notation, when at step t we aim to update block $\mathbf{x}_b$ we use the curvature information vectors $\{\mathbf{v}_b^u, \hat{\mathbf{r}}_b^u\}$ for s largest indices $u \in \mathcal{T}_b$. For simplicity we indicate this set by $\{\mathbf{v}_b^u, \hat{\mathbf{r}}_b^u\}_{u=\hat{t}_b-s+1}^{\hat{t}_b}$ where $u = \hat{t}_b$ corresponds to the largest element in the set $\mathcal{T}_b(t)$ and $u = \hat{t}_b - s + 1$ denotes the s-th largest element in $\mathcal{T}_b(t)$.

To state the update rule of oLBFGS for revising the Hessian inverse approximation matrices of the blocks, define a matrix as $\hat{\mathbf{B}}_b^{t,0} := \eta_b^t \mathbf{I}$ for each block b and t, where the constant η_b^t for $t > 0$ is given by

$$\eta_b^t := \frac{(\mathbf{v}_b^{\hat{t}_b})^T \hat{\mathbf{r}}_b^{\hat{t}_b}}{\|\hat{\mathbf{r}}_b^{\hat{t}_b}\|^2}, \tag{7}$$

where $\hat{t}_b$ is the last time index before step t that block b is updated, i.e., the largest element of $\mathcal{T}_b$ before step t. The matrix $\hat{\mathbf{B}}_b^{t,0}$ is the initial approximate for the Hessian inverse associated with block $\mathbf{x}_b$. The approximate matrix $\hat{\mathbf{B}}_b^t$ is computed by updating the initial matrix $\hat{\mathbf{B}}_b^{t,0}$ using the last s pairs of curvature information $\{\mathbf{v}_b^u, \hat{\mathbf{r}}_b^u\}_{u=\hat{t}_b-s+1}^{\hat{t}_b}$.

We define the approximate Hessian inverse $\hat{\mathbf{B}}_b^t = \hat{\mathbf{B}}_b^{t,s}$ corresponding to block $\mathbf{x}_b$ at step t as the outcome of s recursive applications of the update

$$\hat{\mathbf{B}}_b^{t,u+1} = (\hat{\mathbf{Z}}_b^{\hat{t}_b-s+u+1})^T \,\hat{\mathbf{B}}_b^{t,u}\, (\hat{\mathbf{Z}}_b^{\hat{t}_b-s+u+1}) + \hat{\rho}_b^{\hat{t}_b-s+u+1}\, (\mathbf{v}_b^{\hat{t}_b-s+u+1})\, (\mathbf{v}_b^{\hat{t}_b-s+u+1})^T, \tag{8}$$

where the matrices $\hat{\mathbf{Z}}_b^{\hat{t}_b-s+u+1}$ and the constants $\hat{\rho}_b^{\hat{t}_b-s+u+1}$ in (8) for $u = 0, \ldots, s-1$ are defined as

$$\hat{\rho}_b^k \;=\; \frac{1}{(\mathbf{v}_b^k)^T \hat{\mathbf{r}}_b^k} \quad \text{and} \quad \hat{\mathbf{Z}}_b^k \;=\; \mathbf{I} - \hat{\rho}_b^k \hat{\mathbf{r}}_b^k (\mathbf{v}_b^k)^T. \tag{9}$$

The block-wise oLBFGS update defined by (6) - (9) is summarized in Algorithm 2. The computation cost of $\hat{\mathbf{B}}_b^t$ in (8) is in the order of $O(p_b^2)$, however, for the update in (5) the descent direction $\hat{\mathbf{d}}_b^t := \hat{\mathbf{B}}_b^t \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ is required. Liu and Nocedal (1989) introduced an efficient implementation of product $\hat{\mathbf{B}}_b^t \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ that requires computation complexity of order $O(sp_b)$. We use the same idea for computing the descent direction of ARAPSA for each block. Therefore, the computation complexity of updating each block for ARAPSA is in the order of $O(sp_b)$, while RAPSA requires $O(p_b)$ operations. On the other hand, ARAPSA accelerates the convergence of RAPSA by incorporating the second order information of the objective function for the block updates, as may be observed in the numerical analyses provided in Section 6.

For reference, ARAPSA is also summarized in algorithmic form in Algorithm 3. Steps 2 and 3 are devoted to assigning random blocks to the processors. In Step 2 a subset of available blocks $\mathcal{I}^t$ is chosen. These blocks are assigned to different processors in Step 3. In Step 5 processors compute the partial stochastic gradient corresponding to their assigned blocks $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ using the acquired samples in Step 4. Steps 6 and 7 are devoted to the computation of the ARAPSA descent direction $\hat{\mathbf{d}}_i^t$. In Step 6 the approximate Hessian inverse $\hat{\mathbf{B}}_b^{t,0}$ for block $\mathbf{x}_b$ is initialized as $\hat{\mathbf{B}}_b^{t,0} = \eta_b^t \mathbf{I}$ which is a scaled identity matrix using the expression for η_b^t in (7) for $t > 0$. The initial value of η_b^t is $\eta_b^0 = 1$. In Step 7 we use Algorithm 2 for efficient computation of the descent direction $\hat{\mathbf{d}}_b^t = \hat{\mathbf{B}}_b^t \; \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$. The descent direction $\hat{\mathbf{d}}_b^t$ is used to update the block $\mathbf{x}_b^t$ with stepsize γ^t in Step 8. Step 9 determines the value of the partial stochastic gradient $\nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+1}, \mathbf{\Theta}_i^t)$ which is required for the computation of stochastic gradient variation $\hat{\mathbf{r}}_b^t$. In Step 10 the variable variation $\mathbf{v}_b^t$ and stochastic gradient variation $\hat{\mathbf{r}}_b^t$ associated with block $\mathbf{x}_b$ are computed to be used in the next iteration.

4. Asynchronous Architectures

Up to this point, the RAPSA method dictates that distinct parallel processors select blocks $b_i^t \in \{1, \ldots, B\}$ uniformly at random at each time step t as in Figure 1. However, the requirement that each processor operates on a common time index is burdensome for parallel operations on large computing clusters, as it means that nodes must wait for the processor which has the longest computation time at each step before proceeding. Remarkably, we are able to extend the methods developed in Sections 2 and 3 to the case where the parallel processors need not to operate on a common time index (lock-free) and establish that their performance guarantees carry through, so long as the degree of their asynchronicity is bounded in a certain sense. In doing so, we alleviate the computational bottleneck in the parallel architecture, allowing processors to continue processing data as soon as their local task is complete.

Algorithm 4 Asynchronous RAPSA at processor i

```
1: while $t < T$ do
2:   Processor $i \in \{1, \dots, I\}$ at time index $t$ executes the following steps:
3:     Select block $b_i^t$ uniformly at random from set of blocks $\{1, \dots, B\}$
4:     Choose a set of realizations $\mathbf{\Theta}_i^t$ for the block $\mathbf{x}_b$, $b = b_i^t$
5:     Compute stochastic gradient : $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t) = \frac{1}{L} \sum_{\boldsymbol{\theta} \in \mathbf{\Theta}_i^t} \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \boldsymbol{\theta})$ [cf. (3)]
6:     Update the coordinates of the decision variable $\mathbf{x}_b^{t+\tau+1} = \mathbf{x}_b^{t+\tau} - \gamma^{t+\tau} \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$
7:   Send updated parameters $\mathbf{x}_b^{t+\tau+1}$ associated with block $b = b_i^t$ to shared memory
8:   If another processor is also operating on block $b_i^t$ at time $t$, randomly overwrite
9: end while
```

4.1. Asynchronous RAPSA

Consider the case where each node operates asynchronously. In this case, at an instantaneous time index t, only one processor executes an update, as all others are assumed to be busy. If two processors complete their prior task concurrently, then they draw the same time index at the next available slot, in which case the tie is broken at random. Suppose processor i selects block $b_i^t \in \{1, \dots, B\}$ at time t. Then it grabs the associated component of the decision variable $\mathbf{x}_b^t$ and computes the stochastic gradient $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ associated with the samples $\mathbf{\Theta}_i^t$. This process may take time and during this process other processors may overwrite the variable $\mathbf{x}_b$. Consider the case that the process time of computing stochastic gradient or equivalently the descent direction is τ. Thus, when processor i updates the block b using the evaluated stochastic gradient $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$, it performs the update

$$\mathbf{x}_b^{t+\tau+1} = \mathbf{x}_b^{t+\tau} - \gamma^{t+\tau} \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t) \qquad b = b_i^t. \tag{10}$$

Thus, the descent direction evaluated based on the available information at time index t is used to update the variable at time $t+\tau$. Asynchronous RAPSA is summarized in Algorithm 4. Note that the delay comes from asynchronous implementation of the algorithm and the fact that other processors are able to modify the variable $\mathbf{x}_b$ during the time that processor i computes its descent direction. We assume the the random time τ that each processor requires to compute its descent direction is bounded above by a constant Δ, i.e., $\tau \leq \Delta$ – see Assumption 4.

Despite the minimal coordination of the asynchronous random parallel stochastic algorithm in (10), we may establish the same performance guarantees as that of RAPSA in Section 2. These analytical properties are investigated at length in Section 5.

Remark 1 *One may raise the concern that there could be instances that two processors or more work on a same block. Although, this event is not very likely since $I \ll B$, there is a positive chance that it might happen. This is true since the available processor picks the block that it wants to operate on uniformly at random from the set $\{1, \dots, B\}$. We show that this event does not cause any issues and the algorithm can eventually converge to the optimal argument even if more than one processor work on a specific block at the same time*

Algorithm 5 Asynchronous Accelerated RAPSA at processor i

```
1: while $t < T$ do
2:   Processor $i \in \{1, \ldots, I\}$ at time index $t$ executes the following steps:
3:     Select block $b_i^t$ uniformly at random from set of blocks $\{1, \ldots, B\}$
4:     Choose a set of realizations $\mathbf{\Theta}_i^t$ for the block $\mathbf{x}_b$, $b = b_i^t$
5:     Compute stochastic gradient : $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t) = \frac{1}{L} \sum_{\boldsymbol{\theta} \in \mathbf{\Theta}_i^t} \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \boldsymbol{\theta})$ [cf. (3)]
6:     Compute the initial Hessian inverse approximation: $\hat{\mathbf{B}}_b^{t,0} = \eta_b^t \mathbf{I}$
7:     Compute descent direction:
         $\hat{\mathbf{d}}_b^t = \text{ARAPSA Step}\left(\hat{\mathbf{B}}_b^{t,0},\ \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t),\ \{\mathbf{v}_b^u, \hat{\mathbf{r}}_b^u\}_{u=\hat{t}_b-s+1}^{\hat{t}_b}\right)$
8:     Update the coordinates of the decision variable $\mathbf{x}_b^{t+\tau+1} = \mathbf{x}_b^{t+\tau} - \gamma^{t+\tau}\, \hat{\mathbf{d}}_b^t$
9:     Compute updated stochastic gradient: $\nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+\tau+1}, \mathbf{\Theta}_i^t) = \frac{1}{L} \sum_{\boldsymbol{\theta} \in \mathbf{\Theta}_i^t} \nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+\tau+1}, \boldsymbol{\theta})$
10:    Update variations $\mathbf{v}_b^t = \mathbf{x}_b^{t+\tau+1} - \mathbf{x}_b^t$ and $\hat{\mathbf{r}}_b^t = \nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+\tau+1}, \mathbf{\Theta}_i^t) - \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$
11:    Overwrite the oldest pairs of $\mathbf{v}_b$ and $\hat{\mathbf{r}}_b$ in local memory by $\mathbf{v}_b^t$ and $\hat{\mathbf{r}}_b^t$, respectively.
12:  Send updated parameters and $\{\mathbf{v}_b^t, \hat{\mathbf{r}}_b^t\}$ to shared memory.
13:  If another processor is operating on block $b_i^t$, choose to overwrite with probability 1/2.
14: end while
```

– see Section 5.2. Functionally, this means that if one block is worked on concurrently by two processors, the memory coordination requires that the result of one of the two processors is written to memory with probability $1/2$. *This random overwrite rule applies to the case that three or more processors are operating on the same block as well. In this case, the result of one of the conflicting processors is written to memory with probability* $1/C$ *where* C *is the number of conflicting processors.*

4.2. Asynchronous ARAPSA

In this section, we study the asynchronous implementation of accelerated RAPSA (ARAPSA). The main difference between the synchronous of implementation ARAPSA in Section 3 and the asynchronous version is in the update of the variable $\mathbf{x}_b^t$ corresponding to the block b. Consider the case that processor i finishes its previous task at time t, chooses the block $b = b_i^t$, and reads the variable $\mathbf{x}_b^t$. Then, it computes the stochastic gradient $\nabla f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ using the set of random variables $\mathbf{\Theta}_i^t$. Further, processor i computes the descent direction $\hat{\mathbf{B}}_b^t\ \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ using the last s sets of curvature information $\{\mathbf{v}_b^u, \hat{\mathbf{r}}_b^u\}_{u=\hat{t}_b-s+1}^{\hat{t}_b}$ corresponding to block b as shown in Algorithm 1. If we assume that the required time to compute the descent direction $\hat{\mathbf{B}}_b^t\ \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ is τ, processor i updates the variable $\mathbf{x}_b^{t+\tau}$ as

$$\mathbf{x}_b^{t+\tau+1} = \mathbf{x}_b^{t+\tau} - \gamma^{t+\tau}\, \hat{\mathbf{B}}_b^t\, \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t) \qquad b = b_i^t. \tag{11}$$

Note that the update in (11) is different from the synchronous version in (5) in the time index of the variable that is updated using the available information at time t. In other words, in the synchronous implementation the descent direction $\hat{\mathbf{B}}_b^t\ \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ is used to update the variable $\mathbf{x}_b^t$ with the same time index, while this descent direction is executed to update the variable $\mathbf{x}_b^{t+\tau}$ in asynchronous ARAPSA.

Note that the definitions of the variable variation $\mathbf{v}_b^t$ and the stochastic gradient variation $\hat{\mathbf{r}}_b^t$ are different in asynchronous setting and they are given by

$$\mathbf{v}_b^t \;=\; \mathbf{x}_b^{t+\tau+1} - \mathbf{x}_b^t, \qquad \hat{\mathbf{r}}_b^t \;=\; \nabla_{\mathbf{x}_b} f(\mathbf{x}^{t+\tau+1}, \mathbf{\Theta}_i^t) - \nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t). \tag{12}$$

This modification comes from the fact that the stochastic gradient $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ is already evaluated for the descent direction in (11). Thus, we define the stochastic gradient variation by computing the difference of the stochastic gradient $\nabla_{\mathbf{x}_b} f(\mathbf{x}^t, \mathbf{\Theta}_i^t)$ and the stochastic gradient associated with the same random set $\mathbf{\Theta}_i^t$ evaluated at the most recent iterate which is $\mathbf{x}_b^{t+\tau+1}$. Likewise, the variable variation is redefined as the difference $\mathbf{x}_b^{t+\tau+1} - \mathbf{x}_b^t$. The steps of asynchronous ARAPSA are summarized in Algorithm 5.

5. Convergence Analysis

We show in this section that the sequence of objective function values $F(\mathbf{x}^t)$ generated by RAPSA approaches the optimal objective function value $F(\mathbf{x}^*)$. We further show that the convergence guarantees for synchronous RAPSA generalize to the asynchronous setting. In establishing this result we define the set $\mathcal{S}^t$ corresponding to the components of the vector $\mathbf{x}$ associated with the blocks selected at step t defined by indexing set $\mathcal{I}^t \subset \{1, \dots, B\}$. Note that components of the set $\mathcal{S}^t$ are chosen uniformly at random from the set of blocks $\{\mathbf{x}_1, \dots, \mathbf{x}_B\}$. With this definition, due to convenience for analyzing the proposed methods, we rewrite the time evolution of the RAPSA iterates (Algorithm 1) as

$$\mathbf{x}_i^{t+1} \;=\; \mathbf{x}_i^t - \gamma^t \,\nabla_{\mathbf{x}_i} f(\mathbf{x}^t, \mathbf{\Theta}_i^t) \qquad \text{for all } \mathbf{x}_i \in \mathcal{S}^t, \tag{13}$$

while the rest of the blocks remain unchanged, i.e., $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t$ for $\mathbf{x}_i \notin \mathcal{S}^t$. Since the number of updated blocks is equal to the number of processors, the ratio of updated blocks is $r := |\mathcal{I}^t|/B = I/B$.

The key steps in our analysis are as follows. First, we establish that the random selection of block coordinates $\mathcal{S}^t$ in (13) may be characterized by a ratio of binomial random variables (see Lemma 2) which cancel out to a common constant factor $r = I/B \in (0, 1]$. Then, this reduces the stacked update in expectation over $\mathcal{S}^t$ to a factor of r multiplied by the usual stochastic gradient. The result of the analysis is conducted wielding this modified stochastic descent direction. In particular, using standard smoothness and convexity properties, we can construct a decrement relationship of the objective in conditional expectation. From this descent lemma, we can establish almost sure convergence through the appropriate definition of a supermartingale difference sequence under choice of diminishing step-sizes. Further, from this descent lemma, computing total expectations, we can then derive convergence rates under different step-size selections and glean how the ratio of updated blocks r figures into the algorithm performance.

To proceed with the analysis of RAPSA, we require the following assumptions.

Assumption 1 *The instantaneous objective functions $f(\mathbf{x}, \boldsymbol{\theta})$ are differentiable and the average function $F(\mathbf{x})$ is strongly convex with parameter $m > 0$.*

Assumption 2 *The average objective function gradients $\nabla F(\mathbf{x})$ are Lipschitz continuous with respect to the Euclidian norm with parameter M, i.e., for all $\mathbf{x}, \hat{\mathbf{x}} \in \mathbb{R}^p$, it holds that*

$$\|\nabla F(\mathbf{x}) - \nabla F(\hat{\mathbf{x}})\| \ \leq \ M \ \|\mathbf{x} - \hat{\mathbf{x}}\|. \tag{14}$$

Assumption 3 The variance of the stochastic gradient $\nabla f(\mathbf{x}, \boldsymbol{\theta})$ is uniformly bounded above for all $\mathbf{x}$, i.e., there exists a constant K such that for all variables $\mathbf{x}$, it holds

$$\mathbb{E}_{\boldsymbol{\theta}}\left[\|\nabla f(\mathbf{x}, \boldsymbol{\theta}) - \nabla F(\mathbf{x})\|^2\right] \leq K. \tag{15}$$

Notice that Assumption 1 only enforces strong convexity of the average function F, while the instantaneous functions f_i may not be even convex. Further, notice that since the instantaneous functions f_i are differentiable the average function F is also differentiable. The Lipschitz continuity of the average function gradients ∇F is customary in proving objective function convergence for descent algorithms. The restriction imposed by Assumption 3 is a standard condition in stochastic approximation literature (Robbins and Monro, 1951), its intent being to limit the variance of the stochastic gradients (Nemirovski et al., 2009).

5.1. Convergence of RAPSA

We turn our attention to the random parallel stochastic algorithm defined in (3)-(4) in Section 2, establishing performances guarantees in both the diminishing and constant algorithm step-size regimes. Our first result comes in the form of a expected descent lemma that relates the expected difference of subsequent iterates to the gradient of the average function.

Lemma 2 *Consider the random parallel stochastic algorithm defined in* (3)-(4)*. Recall the definitions of the set of updated blocks $\mathcal{I}^t$ which are randomly chosen from the total B blocks. Define $\mathcal{F}^t$ as a sigma algebra that measures the history of the system up until time t. Then, the expected value of the difference $\mathbf{x}^{t+1} - \mathbf{x}^t$ with respect to the random set $\mathcal{I}^t$ given $\mathcal{F}^t$ is*

$$\mathbb{E}_{\mathcal{I}^t}\left[\mathbf{x}^{t+1} - \mathbf{x}^t \mid \mathcal{F}^t\right] = -r\gamma^t \ \nabla f(\mathbf{x}^t, \boldsymbol{\Theta}^t). \tag{16}$$

Moreover, the expected value of the squared norm $\|\mathbf{x}^{t+1} - \mathbf{x}^t\|^2$ with respect to the random set $\mathcal{I}^t$ given $\mathcal{F}^t$ can be simplified as

$$\mathbb{E}_{\mathcal{I}^t}\left[\|\mathbf{x}^{t+1} - \mathbf{x}^t\|^2 \mid \mathcal{F}^t\right] = r(\gamma^t)^2 \ \left\|\nabla f(\mathbf{x}^t, \boldsymbol{\Theta}^t)\right\|^2. \tag{17}$$

Proof See Appendix A.1. ■

Notice that in the regular stochastic gradient descent method the difference of two consecutive iterates $\mathbf{x}^{t+1} - \mathbf{x}^t$ is equal to the stochastic gradient $\nabla f(\mathbf{x}^t, \boldsymbol{\Theta}^t)$ times the stepsize γ^t. Based on the first result in Lemma 2, the expected value of stochastic gradients with respect to the random set of blocks $\mathcal{I}^t$ is the same as the one for SGD except that it is multiplied by the fraction of updated blocks r. Expression in (17) shows the same relation for the expected value of the squared difference $\|\mathbf{x}^{t+1} - \mathbf{x}^t\|^2$. These relationships confirm that in expectation RAPSA behaves as SGD which allows us to establish the global convergence of RAPSA.

Proposition 3 *Consider the random parallel stochastic algorithm defined in* (3)-(4)*. If Assumptions 1-3 hold, then the objective function error sequence* $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ *satisfies*

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*) \mid \mathcal{F}^t\right] \leq \left(1 - 2mr\gamma^t\left(1 - \frac{M\gamma^t}{2}\right)\right)\left(F(\mathbf{x}^t) - F(\mathbf{x}^*)\right) + \frac{rM(\gamma^t)^2K}{2}. \tag{18}$$

Proof See Appendix A.2. ■

Proposition 3 leads to a supermartingale relationship for the sequence of objective function errors $F(\mathbf{x}^t) - F(\mathbf{x}^*)$. In the following theorem we show that if the sequence of stepsize satisfies standard stochastic approximation diminishing step-size rules (non-summable and squared summable), the sequence of objective function errors $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ converges to null almost surely. Considering the strong convexity assumption this result implies almost sure convergence of the sequence $\|\mathbf{x}^t - \mathbf{x}^*\|^2$ to null.

Theorem 4 *Consider the random parallel stochastic algorithm defined in* (3)-(4) *(Algorithm 1). If Assumptions 1-3 hold true and the sequence of stepsizes are non-summable* $\sum_{t=0}^{\infty}\gamma^t = \infty$ *and square summable* $\sum_{t=0}^{\infty}(\gamma^t)^2 < \infty$ *and for all* $t \geq 0$ *we have* $\gamma^t \leq 1/M$*, then sequence of the variables* $\mathbf{x}^t$ *generated by RAPSA converges almost surely to the optimal argument* $\mathbf{x}^*$*,*

$$\lim_{t\to\infty} \|\mathbf{x}^t - \mathbf{x}^*\|^2 \ = \ 0 \qquad a.s. \tag{19}$$

Moreover, if stepsize is defined as $\gamma^t := \gamma^0T^0/(t + T^0)$ *and the stepsize parameters are chosen such that* $mr\gamma^0T^0 > 1$*, then the expected average function error* $\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right]$ *converges to null at least with a sublinear convergence rate of order* $\mathcal{O}(1/t)$*,*

$$\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right] \ \leq \ \frac{C}{t + T^0}, \tag{20}$$

where the constant C *is defined as*

$$C = \max\left\{\frac{rMK(\gamma^0T^0)^2}{2(rm\gamma^0T^0 - 1)},\ T^0(F(\mathbf{x}^0) - F(\mathbf{x}^*))\right\}. \tag{21}$$

Proof See Appendix A.3. ■

The result in Theorem 4 shows that when the sequence of stepsize is diminishing as $\gamma^t = \gamma^0T^0/(t + T^0)$, the average objective function value $F(\mathbf{x}^t)$ sequence converges to the optimal objective value $F(\mathbf{x}^*)$ with probability 1. Further, the rate of convergence in expectation is at least of $\mathcal{O}(1/t)$.[1]

Notice that the only required condition for the stepsize parameters γ_0 and T_0 is to satisfy the inequalities $mr\gamma^0T^0 > 1$ and $\gamma^0 \leq 1/M$. Indeed, properly choosing the constants γ_0

1. The expectation on the left hand side of (20), and throughout the subsequent convergence rate analysis, is taken with respect to the full algorithm history $\mathcal{F}_0$, which all realizations of both $\mathbf{\Theta}_t$ and $\mathcal{I}_t$ for all $t \geq 0$.

and T_0 can reduce the value of C in (21). However, to optimally choose these constants prior knowledge of the optimal objective function value $F(\mathbf{x}^*)$ is required.

Diminishing stepsizes are useful when exact convergence is required, however, for the case that we are interested in a specific accuracy ϵ the more efficient choice is using a constant stepsize. In the following theorem we study the convergence properties of RAPSA for a constant stepsize $\gamma^t = \gamma$.

Theorem 5 *Consider the random parallel stochastic algorithm defined in* (3)-(4) *(Algorithm 1). If Assumptions 1-3 hold true and the stepsize is constant $\gamma^t = \gamma \leq 1/M$, then a subsequence of the variables $\mathbf{x}^t$ generated by RAPSA converges almost surely to a neighborhood of the optimal argument $\mathbf{x}^*$ as*

$$\liminf_{t\to\infty} \; F(\mathbf{x}^t) - F(\mathbf{x}^*) \;\leq\; \frac{\gamma MK}{2m} \qquad a.s. \tag{22}$$

Moreover, if the constant stepsize γ is chosen such that $mr\gamma < 1$ then the expected average function value error $\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right]$ converges linearly to an error bound as

$$\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right] \leq (1 - m\gamma r)^t \left(F(\mathbf{x}^0) - F(\mathbf{x}^*)\right) + \frac{\gamma MK}{2m}. \tag{23}$$

Proof See Appendix A.4. ■

Notice that according to the result in (23) there exits a trade-off between accuracy and speed of convergence. Decreasing the constant stepsize γ leads to a smaller error bound $(\gamma MK)/(2m)$ and a more accurate convergence, while the linear convergence constant $(1 - m\gamma r)$ increases and the convergence rate becomes slower. Further, note that the error of convergence $(\gamma MK)/(2m)$ is independent of the ratio of updated blocks r, while the constant of linear convergence $1 - m\gamma r$ depends on r. Therefore, updating a fraction of the blocks at each iteration decreases the speed of convergence for RAPSA relative to SGD that updates all of the blocks, however, both of the algorithms reach the same accuracy.

To achieve accuracy ϵ the sum of two terms in the right hand side of (23) should be smaller than ϵ. Let's consider ϕ as a positive constant that is strictly smaller than 1, i.e., $0 < \phi < 1$. Then, we want to have

$$\frac{\gamma MK}{2m} \leq \phi\epsilon, \quad (1 - m\gamma r)^t \left(F(\mathbf{x}^0) - F(\mathbf{x}^*)\right) \leq (1 - \phi)\epsilon. \tag{24}$$

Therefore, to satisfy the first condition in (24) we set the stepsize as $\gamma = 2m\phi\epsilon/MK$. Apply this substitution into the second inequality in (24) and consider the inequality $a + \ln(1-a) < 0$ for $0 < a < 1$, to obtain that

$$t \geq \frac{MK}{2m^2 r\phi\epsilon} \ln\left(\frac{F(\mathbf{x}^0) - F(\mathbf{x}^*)}{(1 - \phi)\epsilon}\right). \tag{25}$$

The lower bound in (25) shows the minimum number of required iterations for RAPSA to achieve accuracy ϵ.

Remark 6 *(Convergence Dependence on Processing Architecture)*

Suppose RAPSA is run with a ***diminishing step-size*** *of the form $\gamma^t = \gamma^0 T^0/(t + T^0)$ with $\gamma_0 = 1/M$ and $T_0 = 2M/(mr)$ so as to obtain the simplified expression for the constant C in (20): $C = \max\{\frac{2MK}{rm^2}, \frac{2M(F(\mathbf{x}^0)-F(\mathbf{x}^*))}{mr}\}$. In this case, we observe that C scales inversely with $r = I/B$, the proportion of blocks updated per iteration. This means that to improve C, one must make r* larger*. Then, for a fixed number of blocks B, one may make the constant C smaller by making $1/r = B/I$ smaller, which may be accomplished by increasing the number of processors I. Thus,* (20) *yields an explicit dependence between the learning constant C and the parallel computing architecture.*

For the ***constant stepsize*** *case, similar to the diminishing step-size exact convergence setting of* (20)*, when we fix $\gamma^t = \gamma < 1/(mr)$ as in* (23)*, the choice of r does not affect the* radius of convergence*. However, unlike the attenuating step-size case, it does effect the* rate of convergence*. Specifically, when we run the algorithm with a constant step-size, we obtain linear convergence to a neighborhood with rate $\rho = 1 - m\gamma r$. Note that ρ closer to null means that convergence happens more quickly, which may be obtained by increasing the term $m\gamma r$. Piggybacking on the analysis which presents the number of iterations needed to achieve a fixed accuracy ϵ in* (25)*, again we observe that the right-hand side is inversely proportional to r, and based on the definition of $r = I/B$, for a fixed B, we obtain that the number of iterations required to attain a specific accuracy* decreases *linearly with the number of processors I.*

These two observations show that by increasing the number of active processors we can achieve linear speed up in both constant and diminishing step-size scenarios.

5.2. Convergence of Asynchronous RAPSA

In this section, we study the convergence of Asynchronous RAPSA (Algorithm 4) developed in Section 4 and we characterize the effect of delay in the asynchronous implementation. To do so, the following condition on the delay τ is required.

Assumption 4 *The random variable τ which is the delay between reading and writing for processors does not exceed the constant Δ, i.e.,*

$$\tau \leq \Delta. \tag{26}$$

The condition in Assumption 4 implies that processors can finish their tasks in a time that is bounded by the constant Δ. This assumption is typical in the analysis of asynchronous algorithms.

To establish the convergence properties of asynchronous RAPSA recall the set $\mathcal{S}^t$ containing the blocks that are updated at step t with associated indices $\mathcal{I}^t \subset \{1, \dots, B\}$. Therefore, the update of asynchronous RAPSA can be written as

$$\mathbf{x}_i^{t+1} \;=\; \mathbf{x}_i^t - \gamma^t \, \nabla_{\mathbf{x}_i} f(\mathbf{x}^{t-\tau}, \mathbf{\Theta}_i^{t-\tau}) \qquad \text{for all } \mathbf{x}_i \in \mathcal{S}^t, \tag{27}$$

and the rest of the blocks remain unchanged, i.e., $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t$ for $\mathbf{x}_i \notin \mathcal{S}^t$.

Note that the random set $\mathcal{I}^t$ and the associated block set $\mathcal{S}^t$ are chosen at time $t - \tau$ in practice; however, for the sake of analysis we can assume that these sets are chosen at time

t. In other words, we can assume that at step $t-\tau$ processor i computes the full (for all blocks) stochastic gradient $\nabla f(\mathbf{x}^{t-\tau}, \mathbf{\Theta}_i^{t-\tau})$ and after finishing this task at time t, it chooses uniformly at random the block that it wants to update. Thus, the block $\mathbf{x}_i$ in (27) is chosen at step t. This new interpretation of the update of asynchronous RAPSA is only important for the convergence analysis of the algorithm and we use it in the proof of following lemma which is similar to the result in Lemma 2 for synchronous RAPSA.

Lemma 7 *Consider the asynchronous random parallel stochastic algorithm (Algorithm 4) defined in* (10). *Recall the definitions of the set of updated blocks $\mathcal{I}^t$ which are randomly chosen from the total B blocks. Define $\mathcal{F}^t$ as a sigma algebra that measures the history of the system up until time t. Then, the expected value of the difference $\mathbf{x}^{t+1}-\mathbf{x}^t$ with respect to the random set $\mathcal{I}^t$ given $\mathcal{F}^t$ is*

$$\mathbb{E}_{\mathcal{I}^t}\left[\mathbf{x}^{t+1}-\mathbf{x}^t \mid \mathcal{F}^t\right] = -\frac{\gamma^t}{B}\ \nabla f(\mathbf{x}^{t-\tau}, \mathbf{\Theta}^{t-\tau}). \tag{28}$$

Moreover, the expected value of the squared norm $\|\mathbf{x}^{t+1}-\mathbf{x}^t\|^2$ with respect to the random set $\mathcal{S}^t$ given $\mathcal{F}^t$ satisfies the identity

$$\mathbb{E}_{\mathcal{I}^t}\left[\|\mathbf{x}^{t+1}-\mathbf{x}^t\|^2 \mid \mathcal{F}^t\right] = \frac{(\gamma^t)^2}{B}\ \left\|\nabla f(\mathbf{x}^{t-\tau}, \mathbf{\Theta}^{t-\tau})\right\|^2. \tag{29}$$

Proof: See Appendex B.1. ■

The results in Lemma 7 is a natural extension of the results in Lemma 2 for the lock-free setting, since in the asynchronous scheme only one of the blocks is updated at each iteration and the ratio r can be simplified as $1/B$. This is the case, assuming that no two processors begin computation at the exact same time, and if this happens we break ties randomly. Thus, each processor operates on a distinct subset of coordinates at each step. Since this coordinate subset could be *any subset*, we assume that it is only one block, since we can reduce more complicated cases to this case by changing the total number of blocks B and the block size p_i. We use the result in Lemma 7 to characterize the decrement in the expected sub-optimality in the following proposition.

Proposition 8 *Consider the asynchronous random parallel stochastic algorithm defined in* (10) *(Algorithm 4). If Assumptions 1-3 hold and the sequence γ^t is decreasing, then for any arbitrary $\rho > 0$ we can show that the sequence of iterates generated by asynchronous RAPSA satisfies*

$$\begin{aligned}
&\mathbb{E}\left[F(\mathbf{x}^{t+1})-F(\mathbf{x}^*) \mid \mathcal{F}^{t-\tau}\right]\\
&\leq \left(1-2m\left(\frac{\gamma^t}{B}-\frac{\rho M\gamma^t}{2B}\right)\right)\mathbb{E}\left[F(\mathbf{x}^t)-F(\mathbf{x}^*) \mid \mathcal{F}^{t-\tau}\right]+\frac{\tau^2 M\gamma^t K(\gamma^{t-\tau})^2}{2\rho B^2}+\frac{M(\gamma^t)^2K}{2B}\\
&\quad+\frac{M^2(\gamma^t)^2}{B}(F(\mathbf{x}^{t-\tau})-F(\mathbf{x}^*))+\sum_{s=1}^{\tau}\frac{\tau M^2\gamma^t(\gamma^{t-s})^2}{\rho B^2}(F(\mathbf{x}^{t-\tau-s})-F(\mathbf{x}^*))
\end{aligned} \tag{30}$$

Proof: See Appendix B.2. ■

We proceed to use the result in Proposition 8 to prove that the sequence of iterates generated by asynchronous RAPSA converges to the optimal argument $\mathbf{x}^*$ defined by (2).

Theorem 9 *Consider the asynchronous RAPSA defined in* (10) *(Algorithm 4). Suppose Assumptions 1-4 hold and the stepsize is defined as* $\gamma^t := \gamma^0 T^0/(t+T^0)$ *and the stepsize parameters are chosen such that*

$$\gamma^0 \leq \min\left\{\frac{B}{2m}\ ,\ \frac{m}{64M^2}\ ,\ \sqrt{\frac{Bm}{128\Delta^2 M^3}}\right\},\quad T^0 \geq 4\Delta,\quad \gamma^0 T^0 > \frac{4B}{3m}, \tag{31}$$

then the expected average function error $\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right]$ *converges to null at least with a sublinear convergence rate of order* $\mathcal{O}(1/t)$*,*

$$\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right] \ \leq\ \frac{Q}{t+T^0}, \tag{32}$$

where the constant C is defined as

$$Q = \max\left\{(F(\mathbf{x}^t) - F(\mathbf{x}^*))T^0, \frac{2MKB(\gamma^0T^0)^2 + 16\Delta^2M^2K(\gamma^0)^3(T^0)^2}{B(3m\gamma^0T^0 - 4B)}\right\} \tag{33}$$

Proof: See Appendix B.3. ■

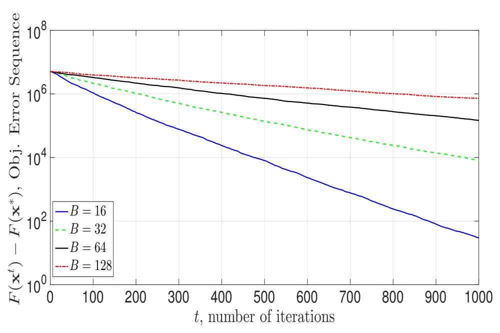
Theorem 9 establishes that the RAPSA algorithm when runs on a lock-free computing architecture, still yields convergence to the optimal argument $\mathbf{x}^*$ defined by (2). Moreover, the expected objective error sequence converges to null as $\mathcal{O}(1/t)$. These results, which correspond to the diminishing step-size regime, are comparable to the performance guarantees (Theorem 4) previously established for RAPSA on a synchronous computing cluster, meaning that the algorithm performance does not degrade significantly when implemented on an asynchronous system. This point is explored numerically in Section 6.

6. Numerical analysis

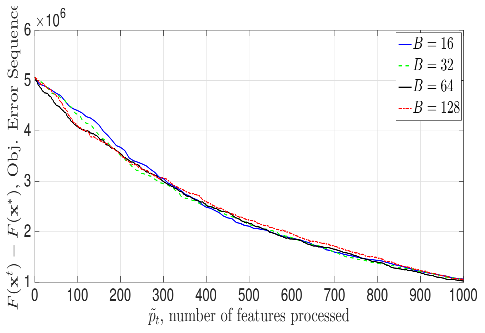
In this section we study the numerical performance of the doubly stochastic approximation algorithms developed in Sections 2-4 by first considering a linear regression problem. We then use RAPSA to develop a visual classifier to distinguish between distinct hand-written digits.

6.1. Linear Regression

We consider a setting in which observations $\mathbf{z}_n \in \mathbb{R}^q$ are collected which are noisy linear transformations $\mathbf{z}_n = \mathbf{H}_n\mathbf{x} + \mathbf{w}_n$ of a signal $\mathbf{x} \in \mathbb{R}^p$ which we would like to estimate, and $\mathbf{w} \sim \mathcal{N}(0, \sigma^2 I_q)$ is a Gaussian random variable. For a finite set of samples N, the optimal $\mathbf{x}^*$ is computed as the least squares estimate $\mathbf{x}^* := \text{argmin}_{\mathbf{x}\in\mathbb{R}^p}(1/N)\sum_{n=1}^N \|\mathbf{H}_n\mathbf{x} - \mathbf{z}_n\|^2$. We run RAPSA on LMMSE estimation problem instances where $q = 1$, $p = 1024$, and $N = 10^4$ samples are given. The observation matrices $\mathbf{H}_n \in \mathbb{R}^{q\times p}$, when stacked over all n (an $N \times p$ matrix), are generated from a matrix normal distribution whose mean is a tri-diagonal matrix. The main diagonal is 2, while the super and sub-diagonals are all set to $-1/2$. Moreover, the true signal has entries chosen uniformly at random from the fractions $\mathbf{x} \in \{1, \dots, p\}/p$. Additionally, the noise variance perturbing the observations is set to $\sigma^2 = 10^{-2}$. We assume that the number of processors $I = 16$ is fixed and each processor is in charge of 1 block. We consider different number of blocks $B = \{16, 32, 64, 128\}$. Note that when the number of blocks is B, there are $p/B = 1024/B$ coordinates in each block.



(a) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. iteration t



(b) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. feature $\tilde{p}_t$

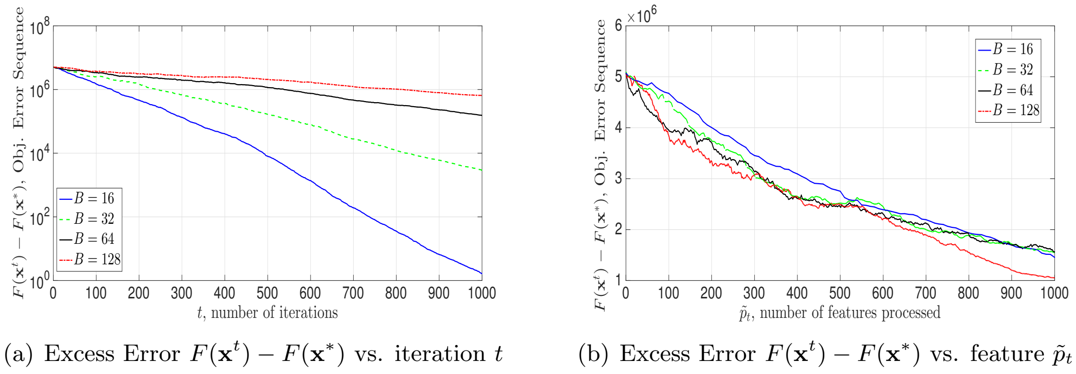
Figure 2: RAPSA on a linear regression (quadratic minimization) problem with signal dimension $p = 1024$ for $N = 10^3$ iterations with mini-batch size $L = 10$ for different number of blocks $B = \{16, 32, 64, 128\}$ initialized as $10^4 \times \mathbf{1}$. We use constant step-size $\gamma^t = \gamma = 10^{-2}$. Convergence is in terms of number of iterations is best when the number of blocks updated per iteration is equal to the number of processors ($B = 16$, corresponding to parallelized SGD), but comparable across the different cases in terms of number of features processed. This shows that there is no price paid in terms of convergence speed for reducing the computation complexity per iteration.

6.1.1. Synchronous Setting

We first consider the performance of RAPSA (Algorithm 1) when using a constant step-size $\gamma^t = \gamma = 10^{-2}$. The size of mini-batch is set as $L = 10$ in the subsequent experiments. To determine the advantages of incomplete randomized parallel processing, we vary the number of coordinates updated at each iteration. In the case that $B = 16$, $B = 32$, $B = 64$, and $B = 128$, in which case the number of updated coordinates per iteration are 1024, 512, 256, and 128, respectively. Notice that the case that $B = 16$ can be interpreted as parallel SGD, which is mathematically equivalent to Hogwild! (Recht et al., 2011), since all the coordinates are updated per iteration, while in other cases $B > 16$ only a subset of 1024 coordinates are updated.

Fig. 2(a) illustrates the convergence path of RAPSA's objective error sequence defined as $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ with $F(\mathbf{x}) = (1/N) \sum_{n=1}^{N} \|\mathbf{H}_n\mathbf{x} - \mathbf{z}_n\|^2$ as compared with the number of iterations t. In terms of iteration t, we observe that the algorithm performance is best when the number of processors equals the number of blocks, corresponding to parallelized stochastic gradient method. However, comparing algorithm performance over iteration t across varying numbers of blocks updates is unfair. If RAPSA is run on a problem for which $B = 32$, then at iteration t it has only processed *half* the data that parallel SGD, i.e., $B = 16$, has processed by the same iteration. Thus for completeness we also consider the algorithm performance in terms of number of features processed $\tilde{p}_t$ which is given by $\tilde{p}_t = ptI/B$.

In Fig. 2(b), we display the convergence of the excess mean square error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ in terms of number of features processed $\tilde{p}_t$. In doing so, we may clearly observe the advantages



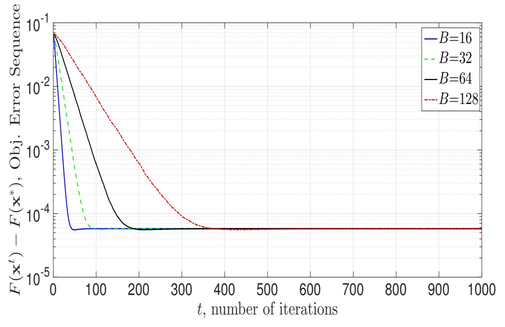
(a) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. iteration t (b) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. feature $\tilde{p}_t$

Figure 3: RAPSA on a linear regression problem with signal dimension $p = 1024$ for $N = 10^3$ iterations with mini-batch size $L = 10$ for different number of blocks $B = \{16, 32, 64, 128\}$ using initialization $\mathbf{x}_0 = 10^4 \times \mathbf{1}$. We use hybrid step-size $\gamma^t = \min(10^{-1.5}, 10^{-1.5}\tilde{T}_0/t)$ with annealing rate $\tilde{T}_0 = 400$. Convergence is faster with smaller B which corresponds to the proportion of blocks updated per iteration r closer to 1 in terms of number of iterations. Contrarily, in terms of number of features processed $B = 128$ is slightly better than the other choices of B. This shows that updating less features/coordinates per iterations can lead to faster convergence in terms of number of processed features.

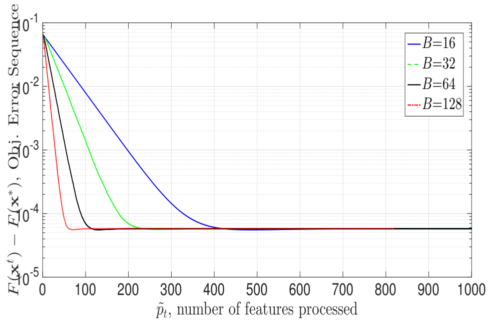
of updating fewer features/coordinates per iteration. Specifically, the different algorithms converge in a nearly identical manner, but RAPSA with $I << B$ may be implemented without any communication bottleneck in the dimension of the decision variable p (also the dimension of the feature space).

We observe a comparable trend when we run RAPSA with a hybrid step-size scheme $\gamma^t = \min(\epsilon, \epsilon\tilde{T}_0/t)$ which is a constant $\epsilon = 10^{-1.5}$ for the first $\tilde{T}_0 = 400$ iterations, after which it diminishes as $O(1/t)$. We again observe in Figure 3(a) that convergence is fastest in terms of excess mean square error versus iteration t when all blocks are updated at each step. However, for this step-size selection, we see that updating fewer blocks per step is *faster* in terms of number of features processed. This result shows that updating fewer coordinates per iteration yields convergence gains in terms of number of features processed. This advantage comes from the advantage of Gauss-Seidel style block selection schemes in block coordinate methods as compared with Jacobi schemes. In particular, it's well understood that for problems settings with specific conditioning, cyclic block updates are superior to parallel schemes, and one may respectively interpret RAPSA as compared to parallel SGD as executing variants of cyclic or parallel block selection schemes. We note that the magnitude of this gain is dependent on the condition number of the Hessian of the expected objective $F(\mathbf{x})$.

We now study the benefits of incorporating approximate second-order information about the objective $F(\mathbf{x})$ into the algorithm in the form of ARAPSA (Algorithm 3). We first run ARAPSA for the linear regression problem outlined above when using a constant step-size $\gamma^t = \gamma = 10^{-2}$ with fixed mini-batch size $L = 10$. Moreover, we again vary the number



(a) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. iteration t



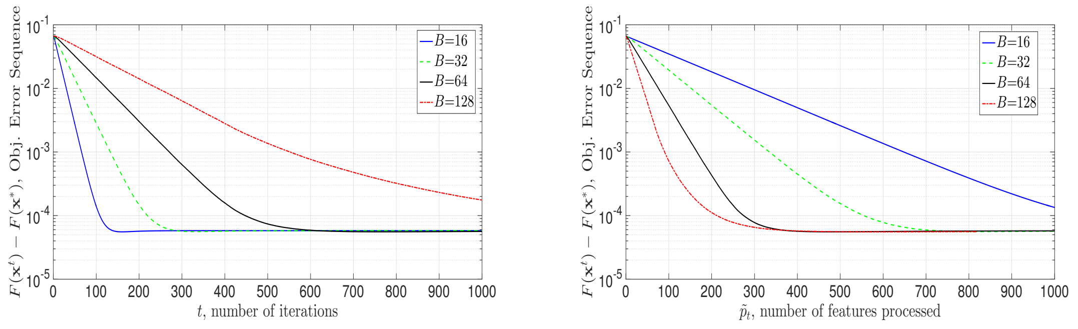
(b) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. feature $\tilde{p}_t$

Figure 4: ARAPSA on a linear regression problem with signal dimension $p = 1024$ for $N = 10^3$ iterations with mini-batch size $L = 10$ for different number of blocks $B = \{16, 32, 64, 128\}$. We use constant step-size $\gamma^t = \gamma = 10^{-1}$ using initialization $10^4 \times \mathbf{1}$. Convergence is comparable across the different cases in terms of number of iterations, but in terms of number of features processed $B = 128$ has the best performance and $B = 16$ (corresponding to parallelized oL-BFGS) converges slowest. We observe that using fewer coordinates per iterations leads to faster convergence in terms of number of processed elements of $\mathbf{x}$.

of blocks as $B = 16$, $B = 32$, $B = 64$, and $B = 128$, corresponding to updating all, half, one-quarter, and one-eighth of the elements of vector $\mathbf{x}$ per iteration, respectively.

Fig. 4(a) displays the convergence path of ARAPSA's excess mean-square error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ versus the number of iterations t. We observe that parallelized oL-BFGS ($I = B$) converges fastest in terms of iteration index t. On the contrary, in Figure 4(b), we may clearly observe that larger B, which corresponds to using *fewer* elements of $\mathbf{x}$ per step, converges faster in terms of number of features processed. The Gauss-Seidel effect is more substantial for ARAPSA as compared with RAPSA due to the fact that the argmin of the instantaneous objective computed in block coordinate descent is better approximated by its second-order Taylor-expansion (ARAPSA, Algorithm 3) as compared with its linearization (RAPSA, Algorithm 1).

We now consider the performance of ARAPSA when a hybrid algorithm step-size is used, i.e. $\gamma^t = \min(10^{-1.5}, 10^{-1.5}\tilde{T}_0/t)$ with attenuation threshold $\tilde{T}_0 = 400$. The results of this numerical experiment are given in Figure 5. We observe that the performance gains of ARAPSA as compared to parallelized oL-BFGS apparent in the constant step-size scheme are more substantial in the hybrid setting. That is, in Figure 5(a) we again see that parallelized oL-BFGS is best in terms of iteration index t – to achieve the benchmark $F(\mathbf{x}^t) - F(\mathbf{x}^*) \leq 10^{-4}$, the algorithm requires $t = 100$, $t = 221$, $t = 412$, and $t > 1000$ iterations for $B = 16$, $B = 32$, $B = 64$, and $B = 128$, respectively. However, in terms of $\tilde{p}_t$, the number of elements of $\mathbf{x}$ processed, to reach the benchmark $F(\mathbf{x}^t) - F(\mathbf{x}^*) \leq 0.1$, we require $\tilde{p}_t > 1000$, $\tilde{p}_t = 570$, $\tilde{p}_t = 281$, and $\tilde{p}_t = 203$, respectively, for $B = 16$, $B = 32$, $B = 64$, and $B = 128$.



(a) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. iteration t (b) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. feature $\tilde{p}_t$

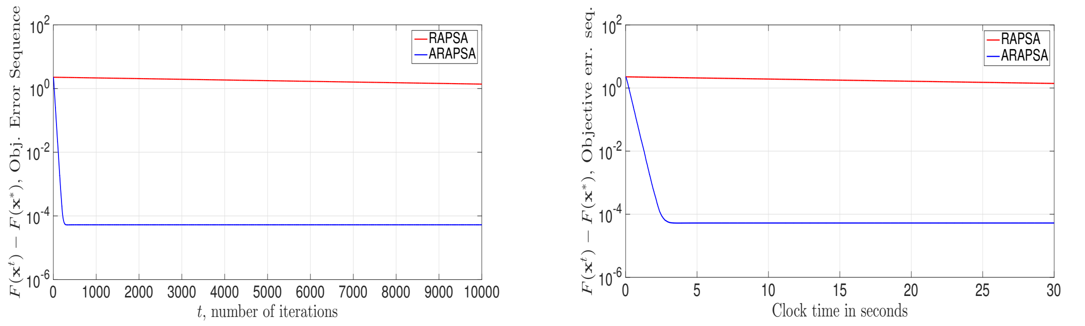
Figure 5: ARAPSA on a linear regression problem with signal dimension $p = 1024$ for $N = 10^4$ iterations with mini-batch size $L = 10$ for different number of blocks $B = \{16, 32, 64, 128\}$. We use hybrid step-size $\gamma^t = \min(10^{-1.5}, 10^{-1.5}\tilde{T}_0/t)$ with annealing rate $\tilde{T}_0 = 400$. Convergence is comparable across the different cases in terms of number of iterations, but in terms of number of features processed $B = 128$ has the best performance and $B = 16$ has the worst performance. This shows that updating less features/coordinates per iterations leads to faster convergence in terms of number of processed features.

We turn to numerically analyzing the performance of Accelerated RAPSA and RAPSA on the linear estimation problem for the case that parameter vectors $\mathbf{x} \in \mathbb{R}^p$ are $p = 500$ dimensional for $N = 10^4$ iterations in the constant step-size case $\gamma = 10^{-2}$. Both algorithms are initialized as $\mathbf{x}_0 = 10^3 \times \mathbf{1}$ with mini-batch size $L = 10$, and ARAPSA uses the curvature memory level $s = 10$. The number of processors is fixed again as $I = 16$, and the number of blocks is $B = 64$, meaning that $r = 1/4$ of the elements of $\mathbf{x}$ are operated on at each iteration.

The results of this numerical evaluation are given in Figure 6. We plot the objective error sequence versus iteration t in Figure 6(a). Observe that ARAPSA converges to within 10^{-4} of the optimum by $t = 300$ iterations in terms of $F(\mathbf{x}^t) - F(\mathbf{x}^*)$, whereas RAPSA, while descending slowly, approaches within 10 of the optimum by $t = 10^4$ iterations. The performance advantages of ARAPSA as compared to RAPSA are also apparent in Figure 6(b), which readjusts the results of Figure 6(a) to be in terms of *actual* elapsed time. We see that despite the higher complexity of ARAPSA per iteration, its empirical performance results in extremely fast convergence on linear estimation problems. That is, in about 3 seconds, the algorithm converges to within 10^{-4} of the optimal estimator in terms of objective function evaluation.

6.1.2. Asynchronous Setting

We turn to studying the empirical performance of the asynchronous variant of RAPSA (Algorithm 4) proposed in Section 4.1. The model we use for asynchronicity is modeled after a random delay phenomenon in physical communication systems in which each local



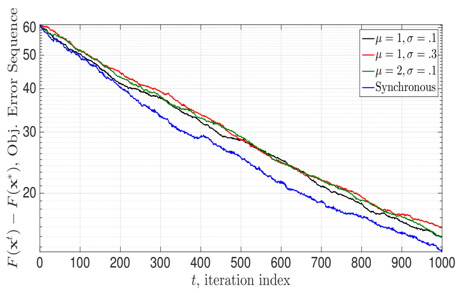
(a) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. iteration t (b) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. clock time (s)

Figure 6: A numerical comparison of RAPSA and ARAPSA on the linear estimation problem stated at the beginning of Section 6.1 for $N = 10^4$ iterations with signal dimension $p = 500$ with constant step-size $\gamma = 10^{-2}$ when there are $I = 16$ processors and $B = 64$ blocks, meaning that one quarter of the elements of $\mathbf{x}$ are updated per iteration. Observe that the rate of convergence for ARAPSA is empirically orders of magnitude higher than RAPSA.

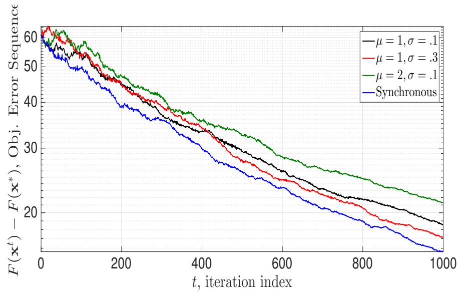
server has a distinct clock which is not locked to the others. Each processor's clock begins at time $t_0^i = t_0$ for all processors $i = 1, \ldots, I$ and selects subsequent times as $t_k = t_{k-1} + w_k^i$, where $w_k^i \sim \mathcal{N}(\mu, \sigma^2)$ is a normal random variable with mean μ and variance σ^2. The variance in this model effectively controls the amount of variability between the clocks of distinct processors.

We run Asynchronous RAPSA for the linear estimation problem when the parameter vector $\mathbf{x}$ is $p = 500$ dimensional for $N = 10^3$ iterations with no mini-batching $L = 1$ for both the case that the algorithm step-size is diminishing and constant step-size regimes for the case that the noise distribution perturbing the collected observations has variance $\sigma^2 = 10^{-2}$, and the observation matrix is as discussed at the outset of Section 6.1. Further, the algorithm is initialized as $\mathbf{x}_0 = 10^3\mathbf{1}$. We run the algorithm for a few different instantiations of asynchronicity, that is, $w_k^i \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu = 1$ or $\mu = 2$, and $\sigma = .1$ or $\sigma = .3$.

The results of this numerical experiment are given in Figure 7 for both the constant and diminishing step-size schemes. We see that the performance of the asynchronous parallel scheme is comparable across different levels of variability among the local clocks of each processor. In particular, in Figure 7(a) which corresponds to the case where the algorithm is run with constant step-size $\gamma = 10^{-2}$, we observe comparable performance in terms of the objective function error sequence $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ with iteration t – across the varying levels of asynchrony we have $F(\mathbf{x}^t) - F(\mathbf{x}^*) \leq 10$ by $t = 10^3$. This trend may also be observed in the diminishing step-size scheme $\gamma^t = 1/t$ which is given in Figure 7(b). That is, the distance to the optimal objective is nearly identical across differing levels of asynchronicity. In both cases, the synchronized algorithm performs better than its asynchronous counterpart.



(a) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. iteration t.



(b) Excess Error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. iteration t.

Figure 7: Asynchronous RAPSA (Algorithm 4) on the linear estimation problem in the constant ($\gamma = 10^4$, left) and diminishing ($\gamma_t = 10^6/(t + 250)$, right) step-size schemes with no mini-batching $L = 1$ for a binary training subset of size $N = 10^3$ with no regularization $\lambda = 0$ when the algorithm is initialized as $\mathbf{x}_0 = 10^3 \times \mathbf{1}$. Varying the asynchronicity distribution has little effect, but we find that convergence behavior is slower than its synchronized counterpart, as expected.

6.2. Hand-Written Digit Recognition

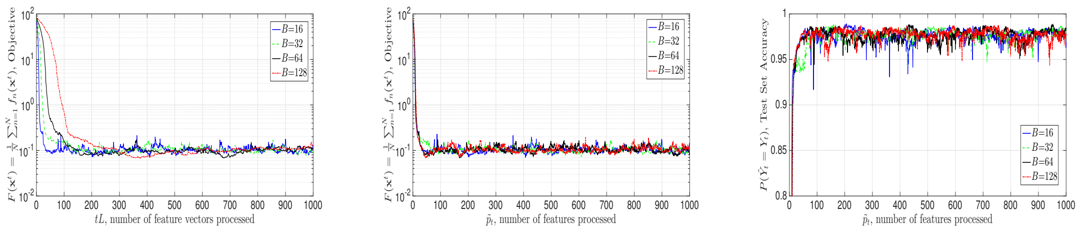
We now make use of RAPSA for visual classification of written digits. To do so, let $\mathbf{z} \in \mathbb{R}^p$ be a feature vector encoding pixel intensities (elements of the unit interval $[0, 1]$ with smaller values being closer to black) of an image and let $y \in \{-1, 1\}$ be an indicator variable of whether the image contains the digit 0 or 8, in which case the binary indicator is respectively $y = -1$ or $y = 1$. We model the task of learning a hand-written digit detector as a logistic regression problem, where one aims to train a classifier $\mathbf{x} \in \mathbb{R}^p$ to determine the relationship between feature vectors $\mathbf{z}_n \in \mathbb{R}^p$ and their associated labels $y_n \in \{-1, 1\}$ for $n = 1, \ldots, N$. The instantaneous function f_n in (1) for this setting is the λ-regularized negative log-likelihood of a generalized linear model of the odds ratio of whether the label is $y_n = 1$ or $y_n = -1$. The empirical risk minimization associated with training set $\mathcal{T} = \{(\mathbf{z}_n, y_n)\}_{n=1}^N$ is to find $\mathbf{x}^*$ as the maximum a posteriori estimate

$$\mathbf{x}^* := \underset{\mathbf{x} \in \mathbb{R}^p}{\operatorname{argmin}} \frac{\lambda}{2}\|\mathbf{x}\|^2 + \frac{1}{N}\sum_{n=1}^{N} \log(1 + \exp(-y_n \mathbf{x}^T \mathbf{z}_n)) \;, \tag{34}$$

where the regularization term $(\lambda/2)\|\mathbf{x}\|^2$ encodes a prior belief on the joint distribution of $(\mathbf{z}, y)$ and helps to avoid overfitting. We use the MNIST dataset, in which feature vectors $\mathbf{z}_n \in \mathbb{R}^p$ are $p = 28^2 = 784$ pixel images whose values are recorded as intensities, or elements of the unit interval $[0, 1]$. Considered here is the subset associated with digits 0 and 8, a training set $\mathcal{T} = \{\mathbf{z}_n, y_n\}_{n=1}^N$ with $N = 1.76 \times 10^4$ sample points.

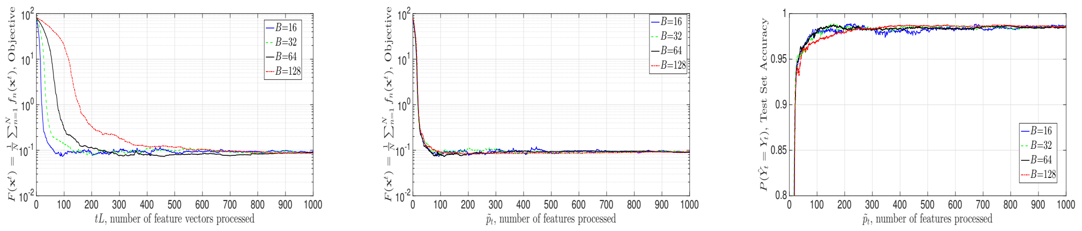
6.2.1. Synchronous Setting

We run RAPSA on this training subset for the cases that $B = 16$, $B = 32$, $B = 64$, and $B = 128$, which are associated with updating p, $p/2$, $p/4$, and $p/8$ features per iteration. We consider the use of RAPSA with both constant and hybrid step-size selections. In Figure



(a) Objective value vs. iteration t. (b) Objective value vs. feature $\tilde{p}_t$. (c) Test Accuracy vs. feature $\tilde{p}_t$

Figure 8: RAPSA on MNIST data with constant step-size $\gamma^t = \gamma = 10^{-.5}$ with no mini-batching $L = 1$. Algorithm performance is best in terms of number of iterations t when all blocks are used per step (parallelized SGD), but in terms of number of features processed, the methods perform comparably. Thus RAPSA performs as well as SGD while breaking the complexity bottleneck in p, the dimension of decision variable $\mathbf{x}$.

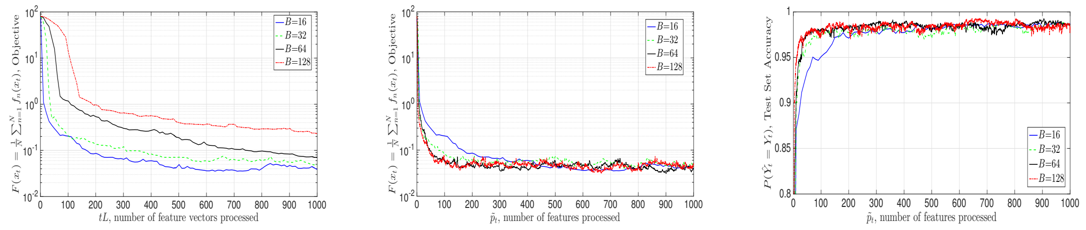


(a) Objective value vs. iteration t. (b) Objective value vs. feature $\tilde{p}_t$. (c) Test Accuracy vs. feature $\tilde{p}_t$

Figure 9: RAPSA on MNIST data with hybrid step-size $\gamma^t = \min(10^{-3/4}, 10^{-3/4}\tilde{T}_0/t)$, with $\tilde{T}_0 = 300$ and no mini-batching $L = 1$. As with the constant step-size selection, we observe that updating all blocks per iteration is best in terms of t, but in terms of elements of $\mathbf{x}$ updated, algorithm performance is nearly identical, meaning that no price is paid for breaking the complexity bottleneck in p.

8, we display the results when we select a constant learning rate $\gamma^t = \gamma = 10^{-.5} = 0.316$. In Figure 8(a) we plot the objective $F(\mathbf{x}^t)$ versus iteration t, and observe that algorithm performance improves with using more elements of $\mathbf{x}$ per iteration. That is, using all p coordinates of $\mathbf{x}$ achieves superior convergence with respect to iteration t. However, as previously noted, iteration index t is an unfair comparator for objective convergence since the four different setting process different number of features per iteration. In Figure 8(b), we instead consider $F(\mathbf{x}^t)$ versus the number of coordinates of $\mathbf{x}$, denoted as $\tilde{p}_t$, that algorithm performance is comparable across the different selections of B. This demonstrates that RAPSA breaks the computational bottleneck in p while suffering no reduction in convergence speed with respect to $\tilde{p}_t$.

We consider further the classification accuracy on a test subset of size $\tilde{N} = 5.88 \times 10^3$, the results of which are shown in Fig. 8(c). We see that the result for classification accuracy



(a) Objective value vs. iteration t. (b) Objective value vs. feature $\tilde{p}_t$. (c) Test Accuracy vs. feature $\tilde{p}_t$

Figure 10: ARAPSA on MNIST data with constant step-size $\gamma^t = \gamma = 10^{-2}$ and mini-batch size $L = 10$, curvature memory $\tau = 10$, and regularizer $\lambda = 7.5 \times 10^{-3}$. Algorithm performance is comparable across different numbers of decision variable coordinates updated per iteration t, but in terms of number of features processed, ARAPSA performance best when using the least information per update.

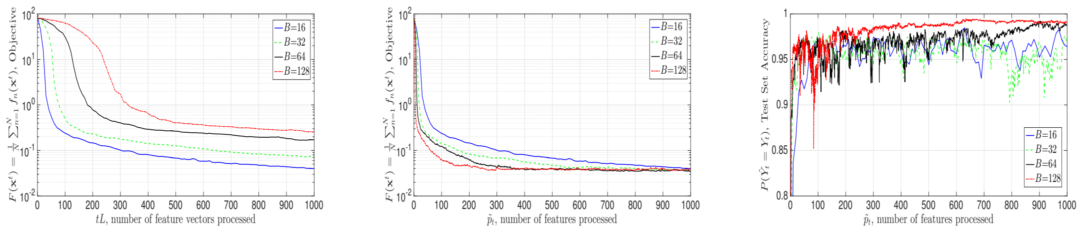
on a test set is consistent with the results for the convergence of the objective function value, and asymptotically reach approximately 98% across the different instances of RAPSA.

In Figure 9 we show the result of running RAPSA for this logistic regression problem with hybrid step-size $\gamma^t = \min(10^{-3/4}, 10^{-3/4}\tilde{T}_0/t)$, with $\tilde{T}_0 = 300$ and no mini-batching $L = 1$. In Fig. 9(a), which displays the objective $F(\mathbf{x}^t)$ versus iteration t, that using full stochastic gradients is better than only updating *some* of the coordinates in terms of the number of iterations t. In particular, to reach the objective benchmark $F(\mathbf{x}^t) \leq 10^{-1}$, we have to run RAPSA $t = 74$, $t = 156$, and $t = 217$, and $t = 631$ iterations, for the cases that $B = 16$, $B = 32$, $B = 64$, and $B = 128$. We illustrate the objective $F(\mathbf{x}^t)$ vs. feature $\tilde{p}_t$ in Fig. 9(b). Here we recover the advantages of randomized incomplete parallel processing: updating fewer blocks per iteration yields comparable algorithm performance.

We additionally display the algorithm's achieved test-set accuracy on a test subset of size $\tilde{N} = 5.88 \times 10^3$ in Fig. 9(c) under the hybrid step-size regime. We again see that after a burn-in period, the classifier achieves the highly accurate asymptotic error rate of between $1 - 2\%$ across the different instantiations of RAPSA. We note that the test set accuracy achieved by the hybrid scheme is superior to the constant step-size setting.

We now run Accelerated RAPSA (Algorithm 3) as stated in Section 3 for this problem setting for the entire MNIST binary training subset associated with digits 0 and 8, with mini-batch size $L = 10$ and the level of curvature information set as $\tau = 10$. We further select regularizer $\lambda = 1/\sqrt{N} = 7.5 \times 10^{-3}$, and consider both constant and hybrid step-size regimes. As before, we study the advantages of incomplete randomized parallel processing by varying the number of blocks $B \in \{16, 32, 64, 128\}$ on an architecture with a fixed number $|\mathcal{I}_t| = I = 16$ of processors. This setup is associated with using all p entries of vector $\mathbf{x}$ at each iteration as compared with 1/2, 1/4, and 1/8 of its entries.

Figures 10 the results of this algorithm run when a constant step-size $\gamma = 10^{-2}$ is used. Observe in Figure 10(a) that the algorithm achieves convergence across the differing numbers of blocks B in terms of iteration t, with faster learning rates achieved with smaller B. In particular, to reach the benchmark $F(\mathbf{x}^t) \leq 10^{-1}$, we require $t = 145$, $t = 311$, and



(a) Objective value vs. iteration t. (b) Objective value vs. feature $\tilde{p}_t$. (c) Test Accuracy vs. feature $\tilde{p}_t$

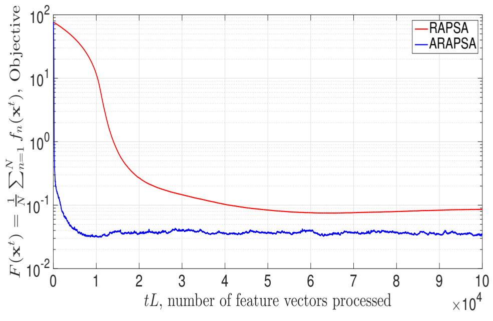
Figure 11: ARAPSA on MNIST data with hybrid step-size $\gamma^t = \min(10^{-1}, 10^{-1}\tilde{T}_0/t)$, with $\tilde{T}_0 = 500$, mini-batch size $L = 10$, curvature memory $\tau = 10$, and regularizer $\lambda = 7.5\times10^{-3}$. Algorithm performance is comparable across different numbers of decision variable coordinates updated per iteration t, but in terms of number of features processed, RAPSA performance best when using the least information per update.

$t = 701$ iterations for $B = 16$, $B = 32$, and $B = 64$, respectively, whereas the case $B = 128$ does not achieve this benchmark by $t = 10^3$. This trend is inverted, however, in Figure 10(b), which displays the objective $F(\mathbf{x}^t)$ with $\tilde{p}_t$ the number of coordinates of $\mathbf{x}$ on which the algorithm operates per step. Observe that using *fewer* entries of $\mathbf{x}$ per iteration is better in terms of number of features processed $\tilde{p}_t$. Furthermore, ARAPSA achieves comparable accuracy on a test set of images, approximately near 98% across different selections of B, as is displayed in Figure 10(c).

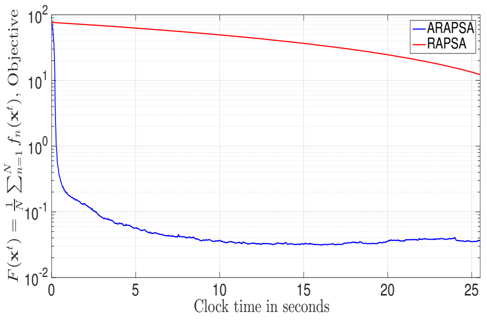
We now run Accelerated RAPSA when the learning rate is hand-tuned to optimize performance via a hybrid scheme $\gamma^t = \min(10^{-1}, 10^{-1}\tilde{T}_0/t)$, with attenuation threshold $\tilde{T}_0 = 500$. The results of this experiment are given in Figure 11. In particular, in Figure 11(a) we plot the objective $F(\mathbf{x}^t)$ with iteration t when the number of blocks B is varied. We see that parallelized oL-BFGS ($I = B$ so that $r = 1$) performs best in terms of t: to achieve the threshold condition $F(\mathbf{x}^t) \leq 10^{-1}$, we require $t = 278$, $t = 522$ iterations for $B = 16$ and $B = 32$, respectively, whereas the cases $B = 64$ and $B = 128$ do not achieve this benchmark by $t = 10^3$. However, the instance of ARAPSA with the fastest and most accurate convergence uses the *least* coordinates of $\mathbf{x}$ when we compare the objective with $\tilde{p}_t$, as may be observed in Figure 11(b). This trend is corroborated in Figure 11(c), where we observe that ARAPSA with $B = 128$ achieves 99% test-set accuracy the fastest, followed by $B = 64$, $B = 32$, and $B = 16$.

We now compare the performance of RAPSA and its accelerated variant on the MNIST digit recognition problem for a binary subset of the training data consisting of $N = 10^5$ samples. We run both algorithms on an $I = 16$ processor simulated architecture with $B = 64$ blocks, such that $r = 1/4$ of the elements of $\mathbf{x}$ are operated upon at each step. We consider the constant algorithm step-size scheme $\gamma = 10^{-2}$ with mini-batch size $L = 10$.

The results of this online training procedure are given in Figure (12), where we plot the objective optimality gap $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ versus the number of feature vectors processed tL (Figure 12(a)) and actual elapsed time (Figure 12(b)). We see ARAPSA achieves su-



(a) $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. iteration t.



(b) $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ vs. clock time (s).

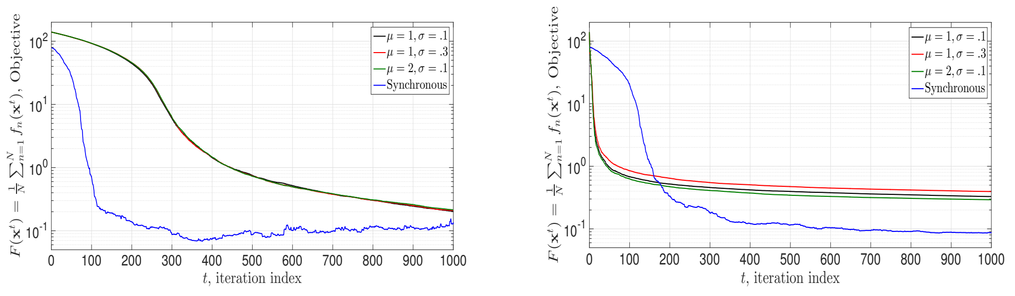
Figure 12: A comparison of RAPSA and ARAPSA on the MNIST digit recognition problem for a binary training subset of size $N = 10^5$ with mini-batch size $L = 10$ in the constant step-size scheme $\gamma = 10^{-2}$. The objective optimality gap $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ is shown with respect to the number of feature vectors processed tL (left) and actual elapsed time (right). While the performance difference between RAPSA and ARAPSA is not as large as in the linear estimation problem, we still observe that ARAPSA substantially accelerates the convergence of RAPSA for a standard machine learning problem.

perior convergence behavior with respect to RAPSA in terms of number of feature vectors processed: to achieve the benchmark $F(\mathbf{x}^t) - F(\mathbf{x}^*) \leq 10^{-1}$, ARAPSA requires fewer than $tL = 200$ feature vectors, whereas RAPSA requires $tL = 4 \times 10^4$ feature vectors. This relationship is corroborated in Figure 12(b), where we see that within a couple seconds ARAPSA converges to within 10^{-1}, whereas after *five* times as long, RAPSA does not achieve this benchmark.

6.2.2. Asynchronous Setting

We now evaluate the empirical performance of the asynchronous variant of RAPSA (Algorithm 4) proposed in Section 4.1 on the logistic regression formulation of the MNIST digit recognition problem. The model we use for asynchronicity is the one outlined in Section 6.1, that is, each local processor has a distinct local clock which is not required coincide with others, begins at time $t_0^i = t_0$ for all processors $i = 1, \ldots, I$, and then selects subsequent times as $t_k = t_{k-1} + w_k^i$. Here $w_k^i \sim \mathcal{N}(\mu, \sigma^2)$ is a normal random variable with mean μ and variance σ^2 which controls the amount of variability between the clocks of distinct processors. We run the algorithm with no regularization $\lambda = 0$ or mini-batching $L = 1$ and initialization $\mathbf{x}_0 = \mathbf{1}$.

The results of this numerical setup are given in Figure 13. We consider the expected risk $F(\mathbf{x}^t)$ in both both the constant ($\gamma = 10^{-2}$, Figure 13(a)) and diminishing ($\gamma^t = 1/t$, Figure 13(b)) algorithm step-size schemes. We see that the level of asynchronicity does not significantly impact the performance in either scheme, and that the convergence guarantees established in Theorem 9 hold true in practice. We again observe that the version of RAPSA with synchronized computations converges at a faster rate than Asynchronous RAPSA.



(a) Objective $F(\mathbf{x}^t)$ vs. iteration t. (b) Objective $F(\mathbf{x}^t)$ vs. iteration t.

Figure 13: Asynchronous RAPSA on MNIST data in the constant ($\gamma = 10^{-2}$, left) and diminishing ($\gamma^t = 1/t$, right) step-size schemes with no mini-batching $L = 1$ for a binary training subset of size $N = 10^3$ with no regularization $\lambda = 0$ when the algorithm is initialized as $\mathbf{x}_0 = \mathbf{1}$. The variability in local processor clocks does not significantly impact performance in both the diminishing and constant step-size settings; however, the synchronous algorithm converges at a faster rate.

7. Numerical Experiments on High-Performance Computing Cluster

In this Section, we report numerical experiments from a real high-performance computing (HPC) cluster. We wrote the code in C++ and compiled using Intel C++ compiler (version 16.0.2). Each node was equipped with two Intel Xeon Processor E7. The communication between nodes was achieved by MPI (we have used Intel's MPI implementation).

7.1. Implementation Details

Before we present our numerical results, let us describe our implementation of synchronous (Algorithm 3) and asynchronous (Algorithm 5) ARAPSA. In both implementations, we partition blocks $b \in \{1, 2, \ldots, B\}$ across computing nodes. Let us denote $\{\mathcal{P}_i\}_{i=1}^I$ such a partition. Then node $i \in \{1, 2, \ldots, I\}$ will "own" blocks $b. \in \mathcal{P}_i$. This means that the node i can choose only blocks from partition $\mathcal{P}_i$. We therefore can also store $\{(v_{b.}^u, r_{b.}^u)\}$ locally which will save a huge amount of storage and communication. Let us now briefly discuss the differences between synchronous and asynchronous implementation (for detailed comparisons and discussion about synchronous and asynchronous distributed algorithms see e.g. Richtárik and Takáč (2016); Mareček et al. (2014)).

Synchronous ARAPSA. Synchronous implementations are more straightforward and popular mainly because they are easy to implement and analyze and one can easily balance local computation with communication (see e.g. Jaggi et al. (2014); Smith et al. (2017); Yang (2013); Ma et al. (2017); Matsushima et al. (2017)). In our synchronous version of the algorithm, we have a dedicated (master) node which holds the "current" state of the optimization variable x^t. This vector is broadcasted to the worker nodes. Afterward, each worker node $i \in \{1, 2, \ldots, I\}$ selects a block $b_{j_i} \in \mathcal{P}_i$ and a mini-batch Θ_i^t and computes an update $d_{b_{j_i}}^t$. After each worker nodes finished its computation, a collective operation "gather" is used, to send the updates $\{d^t_{b_{j_i}}\}_{i=1}^I$ to the master node. Recall, that vectors $\{(v^u_{b.}, r^u_{b.})\}$ are stored locally and hence they do not need to be communicated. After master nodes gather all updates from workers, it applies them and forms vector x^{t+1} which is then again broadcaster to the workers (see Figure 14 left).

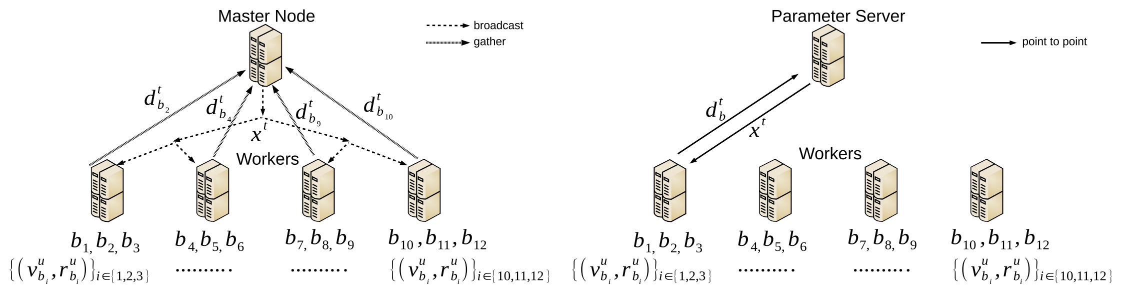


Figure 14: Synchronous implementation (left) or ARAPSA and asynchronous implementation (right). In both implementations we partition blocks $b \in \{1, 2, \ldots, B\}$ between workers and each worker is responsible for updating only its blocks. This approach will save on communication, as workers do not need to send (v^u_b, r^u_b) vectors to each other. In a synchronous implementation, a master node first broadcast the current state of the optimization variable x^t and afterward each worker picks a block from its partition and sends the update d^t_b to master node. Subsequently, master node applies all received updates and broadcasts a new state of optimization variable. In an asynchronous implementation, each worker is asking for a current state of x and sends updates d^t_b independently from each other.

Asynchronous ARAPSA. Asynchronous parallel and distributed optimization algorithms were studied deeply in late 80's (Tsitsiklis et al. (1986)), but they became popular more recently, when more and more researchers obtained access to large computing clusters and clouds (see e.g. Recht et al. (2011); Dean et al. (2012); Mareček et al. (2014); Meng et al. (2016)).

Our implementation closely follows an asynchronous Downpour SGD (Dean et al. (2012)) with the difference, that we do not partition data but rather we partition the blocks $b \in \{1, 2, \ldots, B\}$ (see e.g. Ma and Takáč (2016)). We dedicate one server to be "the parameter server," which holds the current state of the optimization variable x^t. Each other computing nodes are "workers" which in each iteration request the current state of the optimization variable from a parameter server, then they pick a random batch Θ_i, and each worker computes an update d^t_i. This update is subsequently sent to a parameter server, which applies it to x (see Figure 14 right). Note that workers do not synchronize between each others and work independently.

Table 1: Basic characteristics of the datasets.

Dataset	#Features	#Samples	Sparsity	Size [MBs]
RCV-Test	47,236	677,399	0.15%	575
URL	3,231,961	2,396,130	0.0036%	3,198
KDDA	20,216,830	8,407,752	0.00018%	3,593

7.2. Experiments

In this Section, we perform numerical experiments on datasets with 0.6M-8.5M samples and 47k-20M features (see Table 1 for the basic characteristics of the datasets used). All datasets are available for download at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

7.2.1. Evolution of Training Error

In Figures 15- 20 we show the evolution of training error as a function of iterations and wall clock time for various values of I (number of workers used) for both synchronous and asynchronous settings. In all Figures, the top row corresponds to the case when $B = 64$ and the bottom case corresponds to the case when $B = 256$.[2] We run all algorithms for 100 iterations only (to be precise for $100 \cdot I$ block updates). Let us now comment on interesting observations. First of all, for all datasets, we can observe that a smaller number of blocks leads to faster convergence. Also, a larger number of nodes (bigger I) leads to faster convergence. For RCV-Test there is no significant difference between synchronous and asynchronous version. This is mostly because the size of messages is small ($p = 47k$). However, we can observe that if more nodes are used, the speed of the asynchronous algorithm is worse. This, however, is not surprising and can be simply explained as follows: The computation takes typically longer than single communication. In synchronous version, in a single iteration, the master node will receive at most p numbers and will broadcast the current state of x^t which is exactly p numbers. However, in an asynchronous version of the algorithm, the parameter server also receives at most p numbers, however, has to send a fresh state of x^t I times. Moreover, when I becomes large, the communication performed by parameter server will become a bottleneck. This could be overcome if we use multiple parameter servers, each responsible for just a fraction of x.

We would like to also add that the results in Figures 15-20 highlight the speed-up obtained by parallel computing. For instance, the plots in Figure 15 show that for both cases of $B = 64$ and $B = 256$ by increasing the number of active processors I the training error reduces faster in terms of number of iterations. Indeed, as the process is in parallel, by increasing the number of active processors computation time per iteration does not increase, and we observe the same speed-up in terms of overall elapsed time as well.

Moreover, when we look at a fix number of processors, we observe better performance when B is smaller which leads to larger r. For instance, consider the plots in Figure 20 for Asynchronous RAPSA with $I = 2$. As we observe both in terms of time and iteration when

2. We have chosen $\gamma = 10^{-7}$ and size of mini-batch to be 10% of the whole dataset, so the local computation is not much smaller than the communication. We further partition data in such a way, that the computational time for each block would be roughly the same.

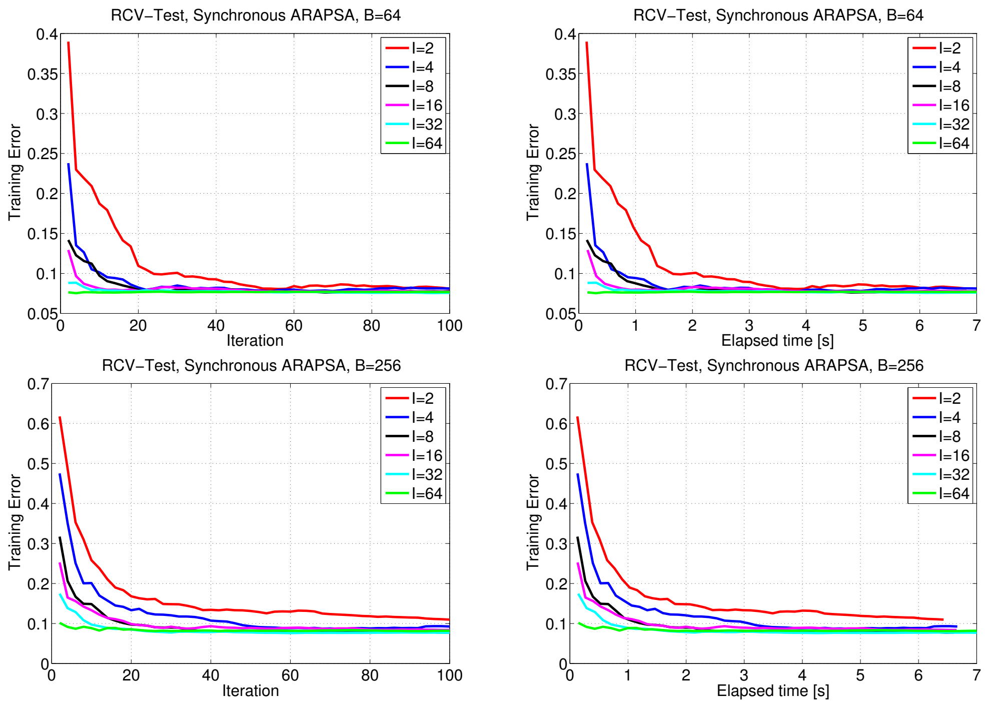


Figure 15: Evolution of training error for RCV-Test dataset for synchronous implementation for various values of B and I.

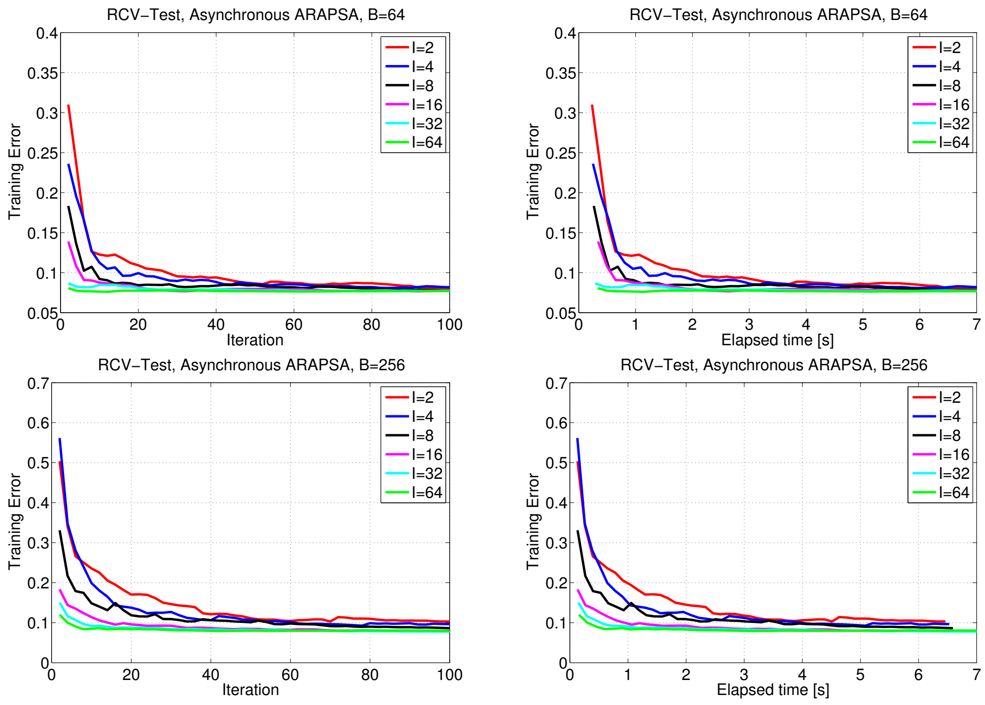


Figure 16: Evolution of training error for RCV-Test dataset for asynchronous implementation for various values of B and I.

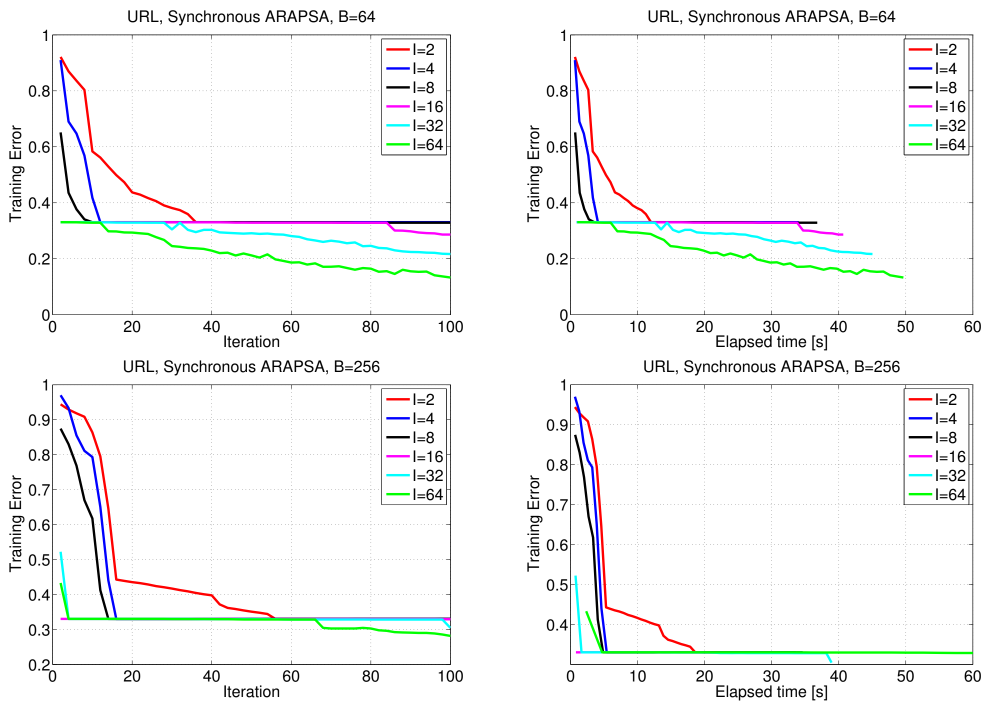


Figure 17: Evolution of training error for URL dataset for synchronous implementation for various values of B and I.

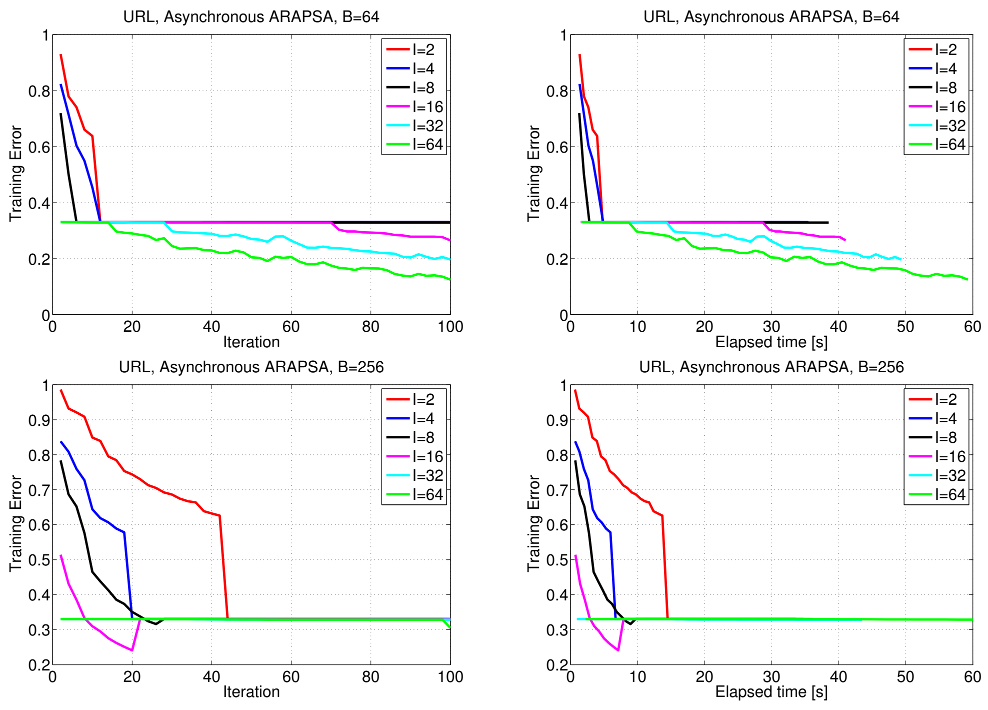


Figure 18: Evolution of training error for URL dataset for asynchronous implementation for various values of B and I.

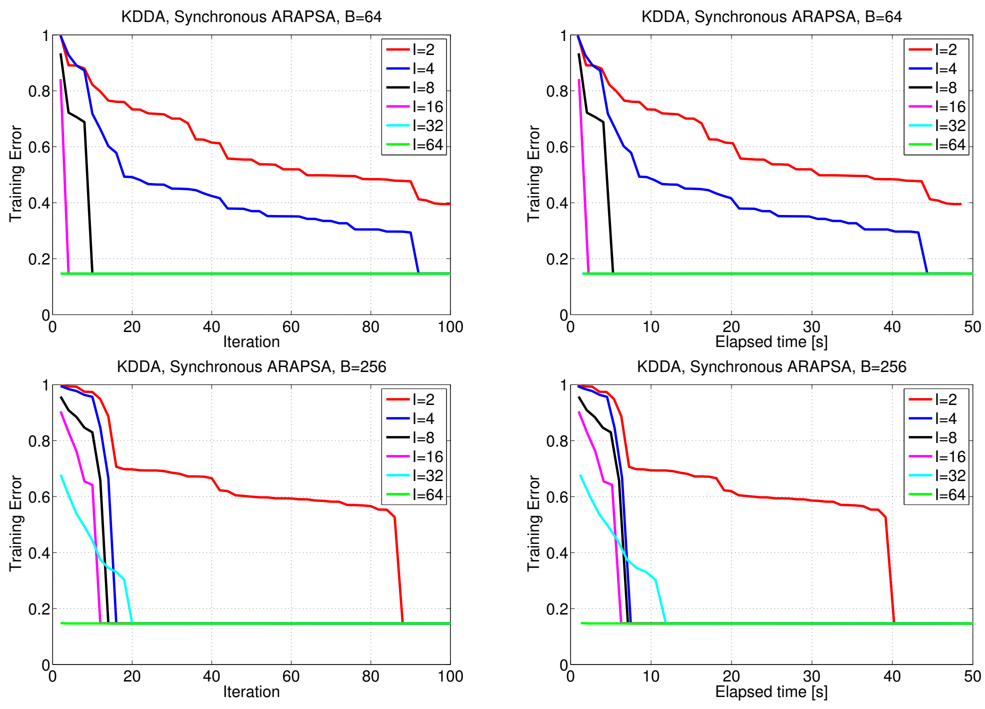


Figure 19: Evolution of training error for KDDA dataset for synchronous implementation for various values of B and I.

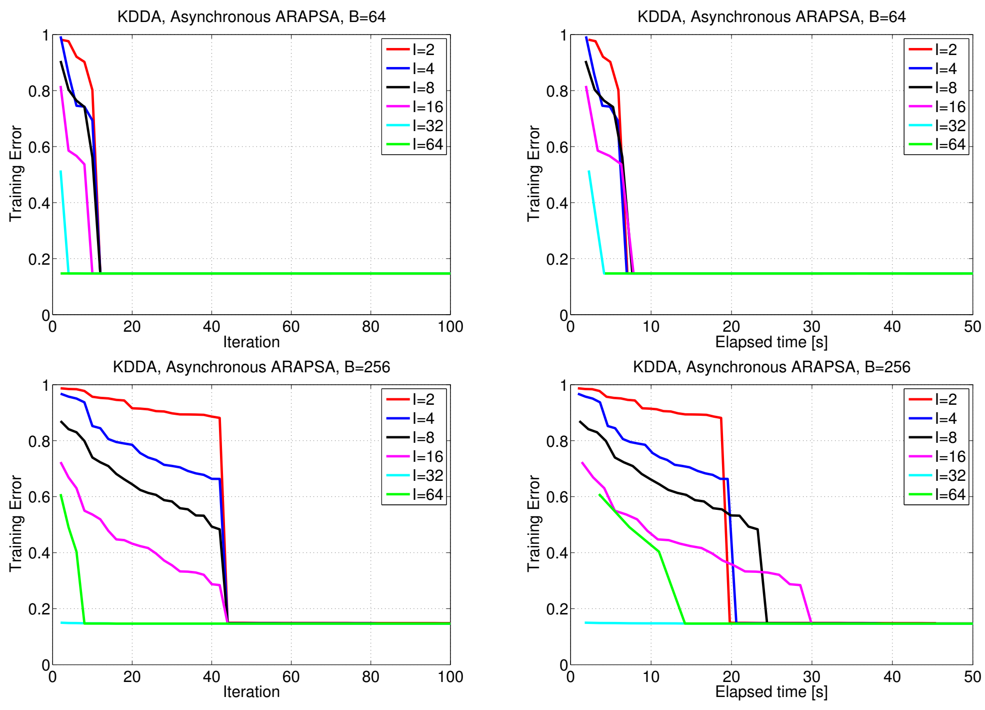


Figure 20: Evolution of training error for KDDA dataset for asynchronous implementation for various values of B and I.

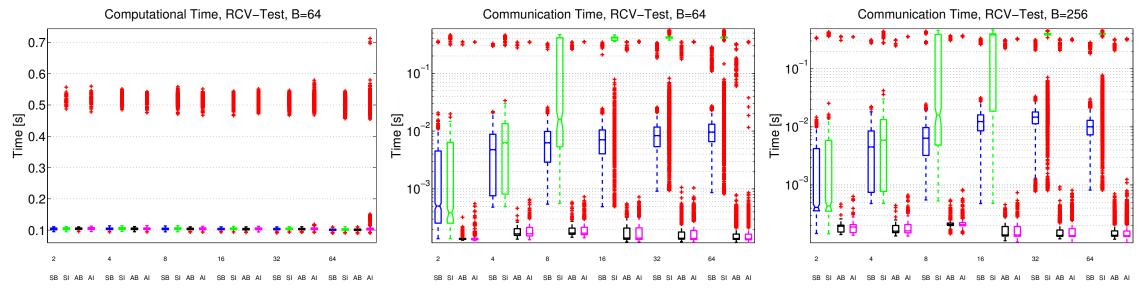


Figure 21: Distribution of local computation and communication for RCV-Test dataset. SB=Synchronous Balanced; SI=Synchronous Imbalanced, AB=Asynchronous Balanced; AI=Asynchronous Imbalanced.

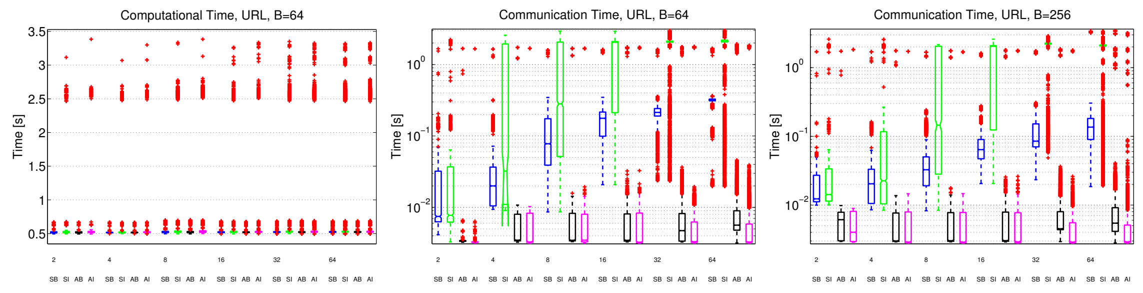


Figure 22: Distribution of local computation and communication for URL dataset.

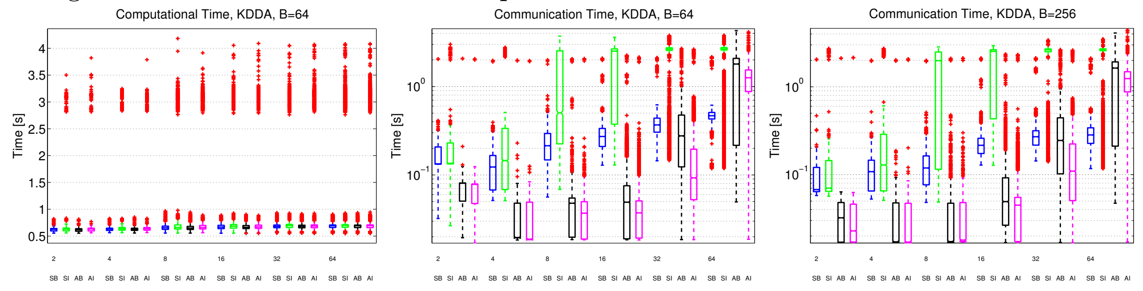


Figure 23: Distribution of local computation and communication for KDDA dataset.

we increase the number of blocks from $B = 64$ to $B = 256$, the convergence speed of the algorithm becomes slower. This observation matches our result that in parallel settings a larger ratio $r = I/B$ leads to faster convergence.

7.2.2. Effect of Imbalanced Blocks and Asynchronicity

In Figures 21, 22 and 23 we compare how the load-balancing of local computation affects the communication both for **S**ynchronous and **A**synchronous ARAPSA. We have two block-partitionings. One is **B**alanced, where the local computation for each block is very similar. The second type of partitioning is **I**mbalanced. In this partitioning, we have created two sizes of blocks. In each iteration, there is a 10% chance, that larger block is chosen. Let us remark, that the larger block requires approximately 5x more work than smaller blocks. In the first column on the plots, we can see the boxplots showing the computation time. Observe that most of the computation took 0.1sec, 0.5sec, and 0.6sec for RCV-Test, URL and KDDA dataset respectively. The middle and last column show the boxplots for communication time (as measured by workers). Note that in the synchronous implementation, even if some of the nodes finish their tasks, we have to wait for the slowest worker and this time is included in communication for all the workers. We can see that synchronous implementation are usually much slower than asynchronous implementations by a few orders of magnitude. This is however not the case for large value of I for KDDA dataset. Let us remind that in this case, the parameter server is the bottleneck as the size of messages it needs to send requires significantly more time than the duration of local computation.

8. Conclusions

We proposed the random parallel stochastic algorithm (RAPSA) proposed as a doubly stochastic approximation algorithm capable of optimization problems associated with learning problems in which both the number of predictive parameters and sample size are huge-scale. RAPSA is doubly stochastic since each processors utilizes a random set of functions to compute the stochastic gradient associated with a randomly chosen sets of variable coordinates. We showed the proposed algorithm converges to the optimal solution sublinearly when the step-size is diminishing. Moreover, linear convergence to a neighborhood of the optimal solution can be achieved using a constant step-size. We further introduced accelerated and asynchronous variants of RAPSA, and presented convergence guarantees for asynchronous RAPSA.

A detailed numerical comparison between RAPSA and parallel SGD for learning a linear estimator and a logistic regressor is provided. The numerical results showcase the advantage of RAPSA with respect to parallel SGD. Further empirical results illustrate the advantages of ARAPSA with respect to parallel oL-BFGS, and that implementing the algorithm on a lock-free parallel computing cluster does not substantially degrade empirical performance.

Acknowledgements

The research of Aryan Mokhtari was supported by NSF award CCF-2007668, and the research of Martin Takáč was partially supported by the U.S. National Science Foundation, under award numbers NSF:CCF-1618717 and NSF:CCF-1740796.

Appendix A. Proof of Results Leading to Theorems 4 and 5

A.1. Proof of Lemma 2

Recall that the components of vector $\mathbf{x}^{t+1}$ are equal to the components of $\mathbf{x}^t$ for the coordinates that are not updated at step t, i.e., $i \notin \mathcal{I}^t$. For the updated coordinates $i \in \mathcal{I}^t$ we know that $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t - \gamma^t \nabla_{\mathbf{x}_i^t} f(\mathbf{x}^t, \boldsymbol{\theta}^t)$. Therefore, $B - I$ blocks of the vector $\mathbf{x}^{t+1} - \mathbf{x}^t$ are 0 and the remaining I randomly chosen blocks are given by $-\gamma^t \nabla_{\mathbf{x}_i^t} f(\mathbf{x}^t, \boldsymbol{\theta}^t)$. Notice that there are $\binom{B}{I}$ different ways for picking I blocks out of the whole B blocks. Therefore, the probability of each combination of blocks is $1/\binom{B}{I}$. Further, each block appears in $\binom{B-1}{I-1}$ of the combinations. Therefore, the expected value can be written as

$$\mathbb{E}_{\mathcal{I}^t}\left[\mathbf{x}^{t+1}-\mathbf{x}^t \mid \mathcal{F}^t\right] = \frac{\binom{B-1}{I-1}}{\binom{B}{I}}\left(-\gamma^t\nabla f(\mathbf{x}^t,\boldsymbol{\Theta}^t)\right). \tag{35}$$

Observe that simplifying the ratio in the right hand sides of (35) leads to

$$\frac{\binom{B-1}{I-1}}{\binom{B}{I}} = \frac{\frac{(B-1)!}{(I-1)!\times(B-I)!}}{\frac{B!}{I!\times(B-I)!}} = \frac{I}{B} = r. \tag{36}$$

Substituting the simplification in (36) into (35) follows the claim in (16). To prove the claim in (17) we can use the same argument that we used in proving (16) to show that

$$\mathbb{E}_{\mathcal{I}^t}\left[\|\mathbf{x}_{t+1}-\mathbf{x}^t\|^2 \mid \mathcal{F}^t\right] = \frac{\binom{B-1}{I-1}}{\binom{B}{I}}(\gamma^t)^2\left\|\nabla f(\mathbf{x}^t,\boldsymbol{\Theta}^t)\right\|^2. \tag{37}$$

By substituting the simplification in (36) into (37) the claim in (17) follows.

A.2. Proof of Proposition 3

By considering the Taylor's expansion of $F(\mathbf{x}^{t+1})$ near the point $\mathbf{x}^t$ and observing the Lipschitz continuity of gradients ∇F with constant M we obtain that the average objective function $F(\mathbf{x}^{t+1})$ is bounded above by

$$F(\mathbf{x}^{t+1}) \leq F(\mathbf{x}^t) + \nabla F(\mathbf{x}^t)^T(\mathbf{x}^{t+1}-\mathbf{x}^t) + \frac{M}{2}\|\mathbf{x}^{t+1}-\mathbf{x}^t\|^2. \tag{38}$$

Compute the expectation of the both sides of (38) with respect to the random set $\mathcal{I}^t$ given the observed set of information $\mathcal{F}^t$. Substitute the terms $\mathbb{E}_{\mathcal{I}^t}\left[\mathbf{x}^{t+1}-\mathbf{x}^t \mid \mathcal{F}^t\right]$ and $\mathbb{E}_{\mathcal{I}^t}\left[\|\mathbf{x}^{t+1}-\mathbf{x}^t\|^2 \mid \mathcal{F}^t\right]$ with their simplifications in (16) and (17), respectively, to write

$$\mathbb{E}_{\mathcal{I}^t}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^t\right] \leq F(\mathbf{x}^t) - r\gamma^t\ \nabla F(\mathbf{x}^t)^T\nabla f(\mathbf{x}^t,\boldsymbol{\Theta}^t) + \frac{rM(\gamma^t)^2}{2}\ \left\|\nabla f(\mathbf{x}^t,\boldsymbol{\Theta}^t)\right\|^2. \tag{39}$$

Notice that the stochastic gradient $\nabla f(\mathbf{x}^t,\boldsymbol{\Theta}^t)$ is an unbiased estimate of the average function gradient $\nabla F(\mathbf{x}^t)$. Therefore, we obtain $\mathbb{E}_{\boldsymbol{\Theta}^t}\left[\nabla f(\mathbf{x}^t,\boldsymbol{\Theta}^t) \mid \mathcal{F}^t\right] = \nabla F(\mathbf{x}^t)$. Observing this relation and considering the assumption in (15), the expected value of (39) with respect to the set of realizations $\boldsymbol{\Theta}^t$ can be written as

$$\mathbb{E}_{\mathcal{I}^t,\boldsymbol{\Theta}^t}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^t\right] \leq F(\mathbf{x}^t) - r\gamma^t\ \left\|\nabla F(\mathbf{x}^t)\right\|^2 + \frac{rM(\gamma^t)^2}{2}\left\|\nabla F(\mathbf{x}^t)\right\|^2 + \frac{rM(\gamma^t)^2K}{2}. \tag{40}$$

Subtracting the optimal objective function value $F(\mathbf{x}^*)$ form the both sides of (40) implies that

$$\mathbb{E}_{\mathcal{I}^t,\boldsymbol{\Theta}^t}\left[F(\mathbf{x}^{t+1})-F(\mathbf{x}^*) \mid \mathcal{F}^t\right] \leq F(\mathbf{x}^t)-F(\mathbf{x}^*) - r\gamma^t\left(1-\frac{M\gamma^t}{2}\right)\left\|\nabla F(\mathbf{x}^t)\right\|^2 + \frac{rM(\gamma^t)^2K}{2}. \tag{41}$$

We proceed to find a lower bound for the gradient norm $\|\nabla F(\mathbf{x}^t)\|$ in terms of the objective value error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$. Assumption 1 states that the average objective function F is strongly convex with constant $m > 0$. Therefore, for any $\mathbf{y}, \mathbf{z} \in \mathbb{R}^p$ we can write

$$F(\mathbf{y}) \geq F(\mathbf{z}) + \nabla F(\mathbf{z})^T(\mathbf{y} - \mathbf{z}) + \frac{m}{2}\|\mathbf{y} - \mathbf{z}\|^2. \tag{42}$$

For fixed $\mathbf{z}$, the right hand side of (42) is a quadratic function of $\mathbf{y}$ whose minimum argument we can find by setting its gradient to zero. Doing this yields the minimizing argument $\hat{\mathbf{y}} = \mathbf{z} - (1/m)\nabla F(\mathbf{z})$ implying that for all $\mathbf{y}$ we must have

$$\begin{aligned} F(\mathbf{y}) \geq\ & F(\mathbf{w}) + \nabla F(\mathbf{z})^T(\hat{\mathbf{y}} - \mathbf{z}) + \frac{m}{2}\|\hat{\mathbf{y}} - \mathbf{z}\|^2 \\ =\ & F(\mathbf{z}) - \frac{1}{2m}\|\nabla F(\mathbf{z})\|^2. \end{aligned} \tag{43}$$

Observe that the bound in (43) holds true for all $\mathbf{y}$ and $\mathbf{z}$. Setting values $\mathbf{y} = \mathbf{x}^*$ and $\mathbf{z} = \mathbf{x}^t$ in (43) and rearranging the terms yields a lower bound for the squared gradient norm $\|\nabla F(\mathbf{x}^t)\|^2$ as

$$\|\nabla F(\mathbf{x}^t)\|^2 \geq 2m(F(\mathbf{x}^t) - F(\mathbf{x}^*)). \tag{44}$$

Substituting the lower bound in (44) by the norm of gradient square $\|\nabla F(\mathbf{x}^t)\|^2$ in (41) implies

$$\mathbb{E}_{\mathcal{I}^t,\boldsymbol{\Theta}^t}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*) \mid \mathcal{F}^t\right] \leq \left(1 - 2mr\gamma^t\left(1 - \frac{M\gamma^t}{2}\right)\right)(F(\mathbf{x}^t) - F(\mathbf{x}^*)) + \frac{rM(\gamma^t)^2K}{2}. \tag{45}$$

and the claim in (18) follows.

A.3. Proof of Theorem 4

We use the relationship in (18) to build a supermartingale sequence. To do so, define the stochastic process α^t as

$$\alpha^t := F(\mathbf{x}^t) - F(\mathbf{x}^*) + \frac{rMK}{2}\sum_{u=t}^{\infty}(\gamma^u)^2. \tag{46}$$

Note that α^t is well-defined because $\sum_{u=t}^{\infty}(\gamma^u)^2 \leq \sum_{u=0}^{\infty}(\gamma^u)^2 < \infty$ is summable. Further define the sequence β_t with values

$$\beta^t := 2m\gamma^t r\left(1 - \frac{M\gamma^t}{2}\right)(F(\mathbf{x}^t) - F(\mathbf{x}^*)). \tag{47}$$

Note that we further assume that for all $t \geq 0$ we have $\gamma^t \leq 1/M$ and therefore $\beta^t \geq 0$. The definitions of sequences α^t and β^t in (46) and (47), respectively, and the inequality in (18) imply that the expected value α^{t+1} given $\mathcal{F}^t$ can be written as

$$\mathbb{E}\left[\alpha^{t+1} \,\middle|\, \mathcal{F}^t\right] \leq \alpha^t - \beta^t. \tag{48}$$

Since the sequences α^t and β^t are nonnegative it follows from (48) that they satisfy the conditions of the supermartingale convergence theorem – see e.g. Theorem E7.4 of Solo and Kong (1994). Therefore, we obtain that: (i) The sequence α^t converges almost surely to a limit. (ii) The sum $\sum_{t=0}^{\infty} \beta^t < \infty$ is almost surely finite. The latter result yields

$$\sum_{t=0}^{\infty} 2m\gamma^t r \left(1 - \frac{M\gamma^t}{2}\right)(F(\mathbf{x}^t) - F(\mathbf{x}^*)) < \infty. \qquad \text{a.s.} \tag{49}$$

Further, we know that $1/2 \leq \left(1 - \frac{M\gamma^t}{2}\right) \leq 1$ and hence we can write

$$\sum_{t=0}^{\infty} \gamma^t (F(\mathbf{x}^t) - F(\mathbf{x}^*)) < \infty. \qquad \text{a.s.} \tag{50}$$

Since the sequence of step sizes γ^t is non-summable there exits a subsequence of sequence $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ which is converging to null. This observation is equivalent to almost sure convergence of $\liminf F(\mathbf{x}^t) - F(\mathbf{x}^*)$ to null,

$$\liminf_{t\to\infty} F(\mathbf{x}^t) - F(\mathbf{x}^*) = 0. \qquad \text{a.s.} \tag{51}$$

Based on the martingale convergence theorem for the sequences α^t and β^t in relation (48), the sequence α^t almost surely converges to a limit. Consider the definition of α^t in (46). Observe that the sum $\sum_{u=t}^{\infty} (\gamma^u)^2$ is deterministic and its limit is null. Therefore, the sequence of the objective function value error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ almost surely converges to a limit. This observation in association with the result in (51) implies that the whole sequence of $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ converges almost surely to null,

$$\lim_{t\to\infty} \; F(\mathbf{x}^t) - F(\mathbf{x}^*) = 0. \qquad \text{a.s.} \tag{52}$$

The last step is to prove almost sure convergence of the sequence $\|\mathbf{x}^t - \mathbf{x}^*\|^2$ to null, as a result of the limit in (52). To do so, we follow by proving a lower bound for the objective function value error $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ in terms of the squared norm error $\|\mathbf{x}^t - \mathbf{x}^*\|^2$. According to the strong convexity assumption, we can write the following inequality

$$F(\mathbf{x}^t) \geq F(\mathbf{x}^*) + \nabla F(\mathbf{x}^*)^T (\mathbf{x}^t - \mathbf{x}^*) + \frac{m}{2}\|\mathbf{x}^t - \mathbf{x}^*\|^2. \tag{53}$$

Observe that the gradient of the optimal point is the null vector, i.e., $\nabla F(\mathbf{x}^*) = \mathbf{0}$. This observation and rearranging the terms in (53) imply that

$$F(\mathbf{x}^t) - F(\mathbf{x}^*) \geq \frac{m}{2}\|\mathbf{x}^t - \mathbf{x}^*\|^2. \tag{54}$$

The upper bound in (54) for the squared norm $\|\mathbf{x}^t - \mathbf{x}^*\|^2$ in association with the fact that the sequence $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ almost surely converges to null, leads to the conclusion that the sequence $\|\mathbf{x}^t - \mathbf{x}^*\|^2$ almost surely converges to zero. Hence, the claim in (19) is valid.

The next step is to study the convergence rate of RAPSA in expectation. In this step we assume that the diminishing stepsize is defined as $\gamma^t = \gamma^0 T^0/(t+T^0)$. Recall the inequality in (18) which is

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*) \mid \mathcal{F}^t\right] \leq \left(1 - 2mr\gamma^t\left(1-\frac{M\gamma^t}{2}\right)\right)\left(F(\mathbf{x}^t) - F(\mathbf{x}^*)\right) + \frac{rM(\gamma^t)^2K}{2}. \tag{55}$$

Since we assume that for all $t \geq 0$ we have $\gamma^t \leq 1/M$, then we can replace $\left(1 - \frac{M\gamma^t}{2}\right)$ in (18) by its lower bound 1/2 to obtain

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*) \mid \mathcal{F}^t\right] \leq \left(1 - mr\gamma^t\right)\left(F(\mathbf{x}^t) - F(\mathbf{x}^*)\right) + \frac{rM(\gamma^t)^2K}{2}. \tag{56}$$

Substitute γ^t by $\gamma^0 T^0/(t+T^0)$ and compute the expected value of (18) given $\mathcal{F}^0$ to obtain

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*)\right] \leq \left(1 - \frac{mr\gamma^0 T^0}{(t+T^0)}\right)\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right] + \frac{rMK(\gamma^0 T^0)^2}{2(t+T^0)^2}. \tag{57}$$

We use the following lemma to show that the result in (57) implies sublinear convergence of the sequence of expected objective value error $\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right]$.

Lemma 10 *Let $c > 1$, $b > 0$ and $t^0 > 0$ be given constants and $u_t \geq 0$ be a nonnegative sequence that satisfies*

$$u^{t+1} \leq \left(1 - \frac{c}{t+t^0}\right)u^t + \frac{b}{(t+t^0)^2}\ , \tag{58}$$

for all times $t \geq 0$. The sequence u^t is then bounded as

$$u^t \leq \ \frac{Q}{t+t^0}, \tag{59}$$

for all times $t \geq 0$, where the constant Q is defined as $Q := \max\{b/(c-1),\ t^0u^0\}$.

Proof See Section 2 in (Nemirovski et al. (2009)). ■

Lemma 10 shows that if a sequence u^t satisfies the condition in (58) then the sequence u^t converges to null at least with the rate of $\mathcal{O}(1/t)$. By assigning values $t^0 = T^0$, $u^t = \mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right]$, $c = mr\gamma^0T^0$, and $b = rMK(\gamma^0T^0)^2/2$, the relation in (57) implies that the inequality in (58) is satisfied for the case that $mr\gamma^0T^0 > 1$. Therefore, the result in (59) holds and we can conclude that

$$\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right] \leq \frac{C}{t+T^0}, \tag{60}$$

where the constant C is defined as

$$C = \max\left\{\frac{rMK(\gamma^0T^0)^2}{2(rm\gamma^0T^0-1)},\ T^0(F(\mathbf{x}^0) - F(\mathbf{x}^*))\right\}. \tag{61}$$

A.4. Proof of Theorem 5

To prove the claim in (22) we use the relationship in (18) (Proposition 3) to construct a supermartingale. First note that since we assume that for all $t \geq 0$ we have $\gamma^t = \gamma \leq 1/M$, then we can replace $\left(1 - \frac{M\gamma^t}{2}\right)$ in (18) by its lower bound 1/2 to obtain

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*) \mid \mathcal{F}^t\right] \leq (1 - mr\gamma)\left(F(\mathbf{x}^t) - F(\mathbf{x}^*)\right) + \frac{rM\gamma^2 K}{2}. \tag{62}$$

Define the stochastic process α^t with values

$$\alpha^t := \left(F(\mathbf{x}^t) - F(\mathbf{x}^*)\right) \times \mathbf{1}\left\{\min_{u \leq t} F(\mathbf{x}^u) - F(\mathbf{x}^*) > \frac{\gamma MK}{2m}\right\} \tag{63}$$

The process α^t tracks the optimality gap $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ until the gap becomes smaller than $\gamma MK/4m$ for the first time at which point it becomes $\alpha^t = 0$. Notice that the stochastic process α^t is always non-negative, i.e., $\alpha^t \geq 0$. Likewise, we define the stochastic process β^t as

$$\beta^t := \gamma mr\left(F(\mathbf{x}^t) - F(\mathbf{x}^*) - \frac{\gamma MK}{2m}\right) \times \mathbf{1}\left\{\min_{u \leq t}\ F(\mathbf{x}^u) - F(\mathbf{x}^*) > \frac{\gamma MK}{2m}\right\}, \tag{64}$$

which follows $\gamma mr\left(F(\mathbf{x}^t) - F(\mathbf{x}^*) - \gamma MK/2m\right)$ until the time that the optimality gap $F(\mathbf{x}^t) - F(\mathbf{x}^*)$ becomes smaller than $\gamma MK/2m$ for the first time. After this moment the stochastic process β^t becomes null. According to the definition of β^t in (64), the stochastic process satisfies $\beta^t \geq 0$ for all $t \geq 0$. Based on the relationship (18) and the definitions of stochastic processes α^t and β^t in (63) and (64) we obtain that for all times $t \geq 0$

$$\mathbb{E}\left[\alpha^{t+1} \mid \mathcal{F}^t\right] \leq \alpha^t - \beta^t. \tag{65}$$

To check the validity of (65) we first consider the case that $\min_{u \leq t}\ F(\mathbf{x}^u) - F(\mathbf{x}^*) > \gamma MK/(2m)$ holds. In this scenario we can simplify the stochastic processes in (63) and (64) as $\alpha^t = F(\mathbf{x}^t) - F(\mathbf{x}^*)$ and $\beta^t = \gamma mr\left(F(\mathbf{x}^t) - F(\mathbf{x}^*) - \gamma MK/(2m)\right)$. Therefore, according to the inequality in (18) the result in (65) is valid. The second scenario that we check is $\min_{u \leq t}\ F(\mathbf{x}^u) - F(\mathbf{x}^*) \leq \gamma MK/(2m)$. Based on the definitions of stochastic processes α^t and β^t, both of these two sequences are equal to 0. Further, notice that when $\alpha^t = 0$, it follows that $\alpha^{t+1} = 0$. Hence, the relationship in (65) is true.

Given the relation in (65) and non-negativity of stochastic processes α^t and β^t we obtain that α^t is a supermartingale. The supermartingale convergence theorem yields: i) The sequence α^t converges to a limit almost surely. ii) The sum $\sum_{t=1}^{\infty} \beta^t$ is finite almost surely. The latter result implies that the sequence β^t is converging to null almost surely, i.e.,

$$\lim_{t \to \infty} \beta^t \ = \ 0 \quad \text{a.s.} \tag{66}$$

Based on the definition of β^t in (64), the limit in (66) is true if one of the following events holds: i) The indicator function is null after for large t. ii) The result that the limit $\lim_{t\to\infty}\left(F(\mathbf{x}^t) - F(\mathbf{x}^*) - \gamma MK/(2m)\right) = 0$ holds true. From any of these two events it is implied that

$$\liminf_{t \to \infty}\ F(\mathbf{x}^t) - F(\mathbf{x}^*) \ \leq \ \frac{\gamma MK}{2m} \quad \text{a.s.} \tag{67}$$

Therefore, the claim in (22) is valid. The result in (67) shows the objective function value sequence $F(\mathbf{x}^t)$ almost sure converges to a neighborhood of the optimal objective function value $F(\mathbf{x}^*)$.

We proceed to prove the result in (23). Compute the expected value of (62) given $\mathcal{F}^0$ to obtain

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*)\right] \leq (1 - m\gamma r)\,\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right] + \frac{rMK\gamma^2}{2}. \tag{68}$$

Notice that the expression in (68) provides an upper bound for the expected value of objective function error $\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*)\right]$ in terms of its previous value $\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right]$ and an error term. Rewriting the relation in (68) for step $t-1$ leads to

$$\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right] \leq (1 - m\gamma r)\,\mathbb{E}\left[F(\mathbf{x}^{t-1}) - F(\mathbf{x}^*)\right] + \frac{rMK\gamma^2}{2}. \tag{69}$$

Substituting the upper bound in (69) for the expectation $\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right]$ in (68) follows an upper bound for the expected error $\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*)\right]$ as

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*)\right] \leq (1 - m\gamma r)^2\,\mathbb{E}\left[F(\mathbf{x}^{t-1}) - F(\mathbf{x}^*)\right] + \frac{rMK\gamma^2}{2}\,(1 + (1 - mr\gamma)). \tag{70}$$

By recursively applying the steps in (69)-(70) we can bound the expected objective function error $\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*)\right]$ in terms of the initial objective function error $F(\mathbf{x}^0) - F(\mathbf{x}^*)$ and the accumulation of the errors as

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*)\right] \leq (1 - m\gamma r)^{t+1}\,(F(\mathbf{x}^0) - F(\mathbf{x}^*)) + \frac{rMK\gamma^2}{2}\sum_{u=0}^{t}(1 - mr\gamma)^u. \tag{71}$$

Substituting t by $t-1$ and simplifying the sum in the right hand side of (71) yields

$$\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*)\right] \leq (1 - m\gamma r)^t\,(F(\mathbf{x}^0) - F(\mathbf{x}^*)) + \frac{MK\gamma}{2m}\left[1 - (1 - mr\gamma)^t\right]. \tag{72}$$

Observing that the term $1 - (1 - mr\gamma)^t$ in the right hand side of (72) is strictly smaller than 1 for the stepsize $\gamma < 1/(mr)$, the claim in (23) follows.

Appendix B. Proofs Leading up to Theorem 9

B.1. Proof of Lemma 7

Proof: Recall that the components of vector $\mathbf{x}^{t+1}$ are equal to the components of $\mathbf{x}^t$ for the coordinates that are not updated at step t, i.e., $i \notin \mathcal{I}^t$. For the updated coordinates $i \in \mathcal{I}^t$ we know that $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t - \gamma^t \nabla_{\mathbf{x}_i^t} f(\mathbf{x}^{t-\tau}, \boldsymbol{\theta}^{t-\tau})$. Therefore, $B-1$ blocks of the vector $\mathbf{x}^{t+1} - \mathbf{x}^t$ are 0 and only one block is given by $-\gamma^t \nabla_{\mathbf{x}_i} f(\mathbf{x}^{t-\tau}, \boldsymbol{\theta}^{t-\tau})$. Since the corresponding processor picks its block uniformly at random from the B sets of blocks we obtain that the expected value of the difference $\mathbf{x}^{t+1} - \mathbf{x}^t$ with respect to the index of the block at time t is given by

$$\mathbb{E}_{\mathcal{I}^t}\left[\mathbf{x}^{t+1} - \mathbf{x}^t \mid \mathcal{F}^t\right] = \frac{1}{B}\left(-\gamma^t \nabla f(\mathbf{x}^{t-\tau}, \boldsymbol{\Theta}^{t-\tau})\right). \tag{73}$$

Substituting the simplification in (73) in place of (35) in the proof of Lemma 2 and simplifying the resulting expression yields the claim in (28). To prove the claim in (29) we can use the same argument that we used in proving (28) to show that

$$\mathbb{E}_{\mathcal{I}^t}\left[\|\mathbf{x}^{t+1}-\mathbf{x}^t\|^2 \mid \mathcal{F}^t\right] = \frac{(\gamma^t)^2}{B}\left\|\nabla f(\mathbf{x}^{t-\tau},\boldsymbol{\Theta}^{t-\tau})\right\|^2, \tag{74}$$

which completes the proof. ■

B.2. Proof of Proposition 8

The Lipschitz continuity of gradients with constant M allows us to write

$$F(\mathbf{x}^{t+1}) \leq F(\mathbf{x}^t) + \nabla F(\mathbf{x}^t)^T(\mathbf{x}^{t+1}-\mathbf{x}^t) + \frac{M}{2}\|\mathbf{x}^{t+1}-\mathbf{x}^t\|^2. \tag{75}$$

Compute the expectation of the both sides of (75) with respect to the random indexing set $\mathcal{I}^t \subset \{1,\dots,B\}$ associated with chosen blocks given the observed set of information $\mathcal{F}^t$. Substitute $\mathbb{E}_{\mathcal{I}^t}\left[\mathbf{x}^{t+1}-\mathbf{x}^t \mid \mathcal{F}^t\right]$ and $\mathbb{E}_{\mathcal{I}^t}\left[\|\mathbf{x}^{t+1}-\mathbf{x}^t\|^2 \mid \mathcal{F}^t\right]$ with their simplifications in (28) and (29), respectively, to write

$$\mathbb{E}_{\mathcal{I}^t}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^t\right] \leq F(\mathbf{x}^t) - \frac{\gamma^t}{B}\,\nabla F(\mathbf{x}^t)^T\nabla f(\mathbf{x}^{t-\tau},\boldsymbol{\Theta}^{t-\tau}) + \frac{M(\gamma^t)^2}{2B}\,\left\|\nabla f(\mathbf{x}^{t-\tau},\boldsymbol{\Theta}^{t-\tau})\right\|^2. \tag{76}$$

Notice that the stochastic gradient $\nabla f(\mathbf{x}^{t-\tau},\boldsymbol{\Theta}^{t-\tau})$ is an unbiased estimate of the average function gradient $\nabla F(\mathbf{x}^{t-\tau})$. Therefore, we obtain $\mathbb{E}\left[\nabla f(\mathbf{x}^{t-\tau},\boldsymbol{\Theta}^{t-\tau}) \mid \mathcal{F}^t\right] = \nabla F(\mathbf{x}^{t-\tau})$. Observing this relation and considering the assumption in (15), the expected value of (76) given the sigma algebra $\mathcal{F}^t$ can be written as

$$\mathbb{E}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^t\right] \leq F(\mathbf{x}^t) - \frac{\gamma^t}{B}\,\nabla F(\mathbf{x}^t)^T\nabla F(\mathbf{x}^{t-\tau}) + \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2 + \frac{M(\gamma^t)^2K}{2B}. \tag{77}$$

By adding and subtracting the term $(\gamma^t/B)\|\nabla F(\mathbf{x}^t)\|^2$ to the right hand side of (77) we obtain

$$\begin{aligned}\mathbb{E}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^t\right] \leq F(\mathbf{x}^t) &- \frac{\gamma^t}{B}\,\|\nabla F(\mathbf{x}^t)\|^2 + \frac{\gamma^t}{B}\left(\|\nabla F(\mathbf{x}^t)\|^2 - \nabla F(\mathbf{x}^t)^T\nabla F(\mathbf{x}^{t-\tau})\right) \\ &+ \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2 + \frac{M(\gamma^t)^2K}{2B}.\end{aligned} \tag{78}$$

Observe that the third term on the right-hand side of (78) is the directional error due to the presence of delays from asynchronicity. We proceed to find an upper bound for the expression $\|\nabla F(\mathbf{x}^t)\|^2 - \nabla F(\mathbf{x}^t)^T\nabla F(\mathbf{x}^{t-\tau})$, which means that the error due to delay may be mitigated. To do so, notice that we can write

$$\begin{aligned}\|\nabla F(\mathbf{x}^t)\|^2 - \nabla F(\mathbf{x}^t)^T\nabla F(\mathbf{x}^{t-\tau}) &= \nabla F(\mathbf{x}^t)^T(\nabla F(\mathbf{x}^t) - \nabla F(\mathbf{x}^{t-\tau})) \\ &\leq \|\nabla F(\mathbf{x}^t)\|\|\nabla F(\mathbf{x}^t) - \nabla F(\mathbf{x}^{t-\tau})\|,\end{aligned} \tag{79}$$

where for the inequality we have used the Cauchy–Schwarz inequality. Apply the fact that the gradient of the objective function is M-Lipschitz continuous, which implies that $\|\nabla F(\mathbf{x}^t) - \nabla F(\mathbf{x}^{t-\tau})\| \leq M\|\mathbf{x}^t - \mathbf{x}^{t-\tau}\|$. Substituting the upper bound $M\|\mathbf{x}^t - \mathbf{x}^{t-\tau}\|$ for $\|\nabla F(\mathbf{x}^t) - \nabla F(\mathbf{x}^{t-\tau})\|$ into (79) we obtain

$$\|\nabla F(\mathbf{x}^t)\|^2 - \nabla F(\mathbf{x}^t)^T \nabla F(\mathbf{x}^{t-\tau}) \leq M\|\nabla F(\mathbf{x}^t)\|\|\mathbf{x}^t - \mathbf{x}^{t-\tau}\|. \tag{80}$$

The difference norm $\|\mathbf{x}^t - \mathbf{x}^{t-\tau}\|$ is equivalent to $\|\sum_{s=t-\tau}^{t-1}(\mathbf{x}^{s+1} - \mathbf{x}^s)\|$ which can be bounded above by $\sum_{s=t-\tau}^{t-1}\|\mathbf{x}^{s+1} - \mathbf{x}^s\|$ by the triangle inequality. Therefore,

$$\|\nabla F(\mathbf{x}^t)\|^2 - \nabla F(\mathbf{x}^t)^T \nabla F(\mathbf{x}^{t-\tau}) \leq M\|\nabla F(\mathbf{x}^t)\| \sum_{s=t-\tau}^{t-1} \|\mathbf{x}^{s+1} - \mathbf{x}^s\|. \tag{81}$$

Substitute the upper bound in (81) for $\|\nabla F(\mathbf{x}^t)\|^2 - \nabla F(\mathbf{x}^t)^T \nabla F(\mathbf{x}^{t-\tau})$ into (78) to obtain

$$\begin{aligned}\mathbb{E}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^t\right] \leq F(\mathbf{x}^t) - \frac{\gamma^t}{B}\|\nabla F(\mathbf{x}^t)\|^2 + \frac{M\gamma^t}{B}\|\nabla F(\mathbf{x}^t)\| \sum_{s=t-\tau}^{t-1} \|\mathbf{x}^{s+1} - \mathbf{x}^s\| \\ + \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2 + \frac{M(\gamma^t)^2 K}{2B}.\end{aligned} \tag{82}$$

Note that for any positive scalars a, b, and ρ the inequality $ab \leq (\rho/2)a^2 + (1/2\rho)b^2$ holds. If we set $a := \|\nabla F(\mathbf{x}^t)\|$ and $b := \sum_{s=t-\tau}^{t-1}\|\mathbf{x}^{s+1} - \mathbf{x}^s\|$ we obtain that

$$\begin{aligned}\|\nabla F(\mathbf{x}^t)\| \sum_{s=t-\tau}^{t-1} \|\mathbf{x}^{s+1} - \mathbf{x}^s\| &\leq \frac{\rho}{2}\|\nabla F(\mathbf{x}^t)\|^2 + \frac{1}{2\rho}\left[\sum_{s=t-\tau}^{t-1} \|\mathbf{x}^{s+1} - \mathbf{x}^s\|\right]^2 \\ &\leq \frac{\rho}{2}\|\nabla F(\mathbf{x}^t)\|^2 + \frac{\tau}{2\rho}\sum_{s=t-\tau}^{t-1} \|\mathbf{x}^{s+1} - \mathbf{x}^s\|^2,\end{aligned} \tag{83}$$

Now substituting the upper bound in (83) into (82) yields

$$\begin{aligned}\mathbb{E}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^t\right] \leq F(\mathbf{x}^t) - \left(\frac{\gamma^t}{B} - \frac{\rho M\gamma^t}{2B}\right)\|\nabla F(\mathbf{x}^t)\|^2 + \frac{\tau M\gamma^t}{2\rho B}\sum_{s=t-\tau}^{t-1} \|\mathbf{x}^{s+1} - \mathbf{x}^s\|^2 \\ + \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2 + \frac{M(\gamma^t)^2 K}{2B}.\end{aligned} \tag{84}$$

Compute the expected value of both sides of (84) given the sigma-algebra $\mathcal{F}^{t-1}$ to obtain

$$\begin{aligned}&\mathbb{E}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^{t-1}\right] \\ &\leq \mathbb{E}\left[F(\mathbf{x}^t) \mid \mathcal{F}^{t-1}\right] - \frac{\gamma^t}{B}\left(1 - \frac{\rho M}{2}\right)\mathbb{E}\left[\|\nabla F(\mathbf{x}^t)\|^2 \mid \mathcal{F}^{t-1}\right] \\ &\quad + \frac{\tau M\gamma^t}{2\rho B}\mathbb{E}\left[\sum_{s=t-\tau}^{t-1} \|\mathbf{x}^{s+1} - \mathbf{x}^s\|^2 \mid \mathcal{F}^{t-1}\right] + \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2 + \frac{M(\gamma^t)^2 K}{2B}\ ,\end{aligned} \tag{85}$$

Note that the last term of the sum $\mathbb{E}\left[\sum_{s=t-\tau}^{t-1}\|\mathbf{x}^{s+1}-\mathbf{x}^{s}\|^2 \mid \mathcal{F}^{t-1}\right]$ can be bounded above using the expression in (74) as

$$\mathbb{E}\left[\|\mathbf{x}^{t}-\mathbf{x}^{t-1}\|^2 \mid \mathcal{F}^{t-1}\right] = \frac{(\gamma^{t-1})^2}{B}\left(\|\nabla F(\mathbf{x}^{t-1-\tau})\|^2 + K\right) \tag{86}$$

Hence, we can show that

$$\begin{aligned}
&\mathbb{E}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^{t-1}\right]\\
&\leq \mathbb{E}\left[F(\mathbf{x}^{t}) \mid \mathcal{F}^{t-1}\right] - \frac{\gamma^t}{B}\left(1-\frac{\rho M}{2}\right)\mathbb{E}\left[\|\nabla F(\mathbf{x}^{t})\|^2 \mid \mathcal{F}^{t-1}\right] + \frac{\tau M \gamma^t}{2\rho B}\sum_{s=t-\tau}^{t-2}\|\mathbf{x}^{s+1}-\mathbf{x}^{s}\|^2\\
&\quad + \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2 + \frac{M(\gamma^t)^2K}{2B} + \frac{\tau M\gamma^t(\gamma^{t-1})^2}{2\rho B^2}\|\nabla F(\mathbf{x}^{t-\tau-1})\|^2 + \frac{\tau K M\gamma^t(\gamma^{t-1})^2}{2\rho B^2}
\end{aligned} \tag{87}$$

Follow the same argument for the remaining term of the sum to obtain

$$\begin{aligned}
&\mathbb{E}\left[F(\mathbf{x}^{t+1}) \mid \mathcal{F}^{t-\tau}\right]\\
&\leq \mathbb{E}\left[F(\mathbf{x}^{t}) \mid \mathcal{F}^{t-\tau}\right] - \frac{\gamma^t}{B}\left(1-\frac{\rho M}{2}\right)\mathbb{E}\left[\|\nabla F(\mathbf{x}^{t})\|^2 \mid \mathcal{F}^{t-\tau}\right] + \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2\\
&\quad + \frac{M(\gamma^t)^2K}{2B} + \sum_{s=1}^{\tau}\frac{\tau M\gamma^t(\gamma^{t-s})^2}{2\rho B^2}\|\nabla F(\mathbf{x}^{t-\tau-s})\|^2 + \sum_{s=1}^{\tau}\frac{\tau K M\gamma^t(\gamma^{t-s})^2}{2\rho B^2}
\end{aligned} \tag{88}$$

Notice that the sequence of stepsizes γ^t is decreasing, thus the sum $\sum_{s=t-\tau}^{t-1}(\gamma^s)^2$ can be bounded above by $\tau(\gamma^{t-\tau})^2$. Applying this substutition and subtracting the optimal objective function value $F(\mathbf{x}^*)$ from both sides of the implied expression lead to

$$\begin{aligned}
&\mathbb{E}\left[F(\mathbf{x}^{t+1})-F(\mathbf{x}^*) \mid \mathcal{F}^{t-\tau}\right]\\
&\leq \mathbb{E}\left[F(\mathbf{x}^{t})-F(\mathbf{x}^*) \mid \mathcal{F}^{t-\tau}\right] - \left(\frac{\gamma^t}{B}-\frac{\rho M\gamma^t}{2B}\right)\mathbb{E}\left[\|\nabla F(\mathbf{x}^{t})\|^2 \mid \mathcal{F}^{t-\tau}\right] + \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2\\
&\quad + \sum_{s=1}^{\tau}\frac{\tau M\gamma^t(\gamma^{t-s})^2}{2\rho B^2}\|\nabla F(\mathbf{x}^{t-\tau-s})\|^2 + \frac{\tau^2 M\gamma^t K(\gamma^{t-\tau})^2}{2\rho B^2} + \frac{M(\gamma^t)^2K}{2B}.
\end{aligned} \tag{89}$$

We make use of the fact that the average function $F(\mathbf{x})$ is m-strongly convex in applying the relation $\|\nabla F(\mathbf{x}^t)\|^2 \geq 2m(F(\mathbf{x}^t)-F(\mathbf{x}^*))$ to the expression (91). Therefore,

$$\begin{aligned}
&\mathbb{E}\left[F(\mathbf{x}^{t+1})-F(\mathbf{x}^*) \mid \mathcal{F}^{t-\tau}\right]\\
&\leq \left(1-2m\left(\frac{\gamma^t}{B}-\frac{\rho M\gamma^t}{2B}\right)\right)\mathbb{E}\left[F(\mathbf{x}^{t})-F(\mathbf{x}^*) \mid \mathcal{F}^{t-\tau}\right] + \frac{M(\gamma^t)^2}{2B}\|\nabla F(\mathbf{x}^{t-\tau})\|^2\\
&\quad + \sum_{s=1}^{\tau}\frac{\tau M\gamma^t(\gamma^{t-s})^2}{2\rho B^2}\|\nabla F(\mathbf{x}^{t-\tau-s})\|^2 + \frac{\tau^2 M\gamma^t K(\gamma^{t-\tau})^2}{2\rho B^2} + \frac{M(\gamma^t)^2K}{2B}.
\end{aligned} \tag{90}$$

Moreover, using M-smoothness of the function $F(\mathbf{x})$ know that $\|\nabla F(\mathbf{x}^s)\|^2 \leq 2M(F(\mathbf{x}^s) - F(\mathbf{x}^*))$. Therefore,

$$\begin{aligned}
&\mathbb{E}\left[F(\mathbf{x}^{t+1}) - F(\mathbf{x}^*) \mid \mathcal{F}^{t-\tau}\right] \\
&\leq \left(1 - 2m\left(\frac{\gamma^t}{B} - \frac{\rho M\gamma^t}{2B}\right)\right)\mathbb{E}\left[F(\mathbf{x}^t) - F(\mathbf{x}^*) \mid \mathcal{F}^{t-\tau}\right] + \frac{\tau^2 M\gamma^t K(\gamma^{t-\tau})^2}{2\rho B^2} + \frac{M(\gamma^t)^2 K}{2B} \\
&\quad + \frac{M^2(\gamma^t)^2}{B}(F(\mathbf{x}^{t-\tau}) - F(\mathbf{x}^*)) + \sum_{s=1}^{\tau}\frac{\tau M^2\gamma^t(\gamma^{t-s})^2}{\rho B^2}(F(\mathbf{x}^{t-\tau-s}) - F(\mathbf{x}^*)) \qquad (91)
\end{aligned}$$

Hence, the result in Proposition 8 follows.

B.3. Proof of Theorem 9

Proof: We use the result in Proposition 8 to prove the claim. Begin by defining the non-negative stochastic processes α^t, β^t, and ζ^t for $t \geq 0$ as

$$\begin{aligned}
&\alpha^t := F(\mathbf{x}^t) - F(\mathbf{x}^*), \qquad \beta^t := \frac{2m\gamma^t}{B}\left[1 - \frac{\rho M}{2}\right](F(\mathbf{x}^t) - F(\mathbf{x}^*)), \\
&\zeta^t := \frac{MK(\gamma^t)^2}{2B} + \frac{\tau^2 MK\gamma^t(\gamma^{t-\tau})^2}{2\rho B^2}. \qquad (92)
\end{aligned}$$

According to the definitions in (92) and the inequality in (30) we can write

$$\begin{aligned}
\mathbb{E}\left[\alpha^{t+1} \mid \mathcal{F}^{t-\tau}\right] &\leq \mathbb{E}\left[\alpha^t \mid \mathcal{F}^{t-\tau}\right] - \frac{2m\gamma^t}{B}\left(1 - \frac{\rho M}{2}\right)\mathbb{E}\left[\alpha^t \mid \mathcal{F}^{t-\tau}\right] + \frac{M^2(\gamma^t)^2}{B}\alpha^{t-\tau} \\
&\quad + \sum_{s=1}^{\tau}\frac{\tau M^2\gamma^t(\gamma^{t-s})^2}{\rho B^2}\alpha^{t-\tau-s} + \zeta^t. \qquad (93)
\end{aligned}$$

Computing the expected value of both sides of (93) with respect to the initial sigma algebra $\mathbb{E}\left[\cdot \mid \mathcal{F}^0\right] = \mathbb{E}\left[\cdot\right]$ yields

$$\begin{aligned}
\mathbb{E}\left[\alpha^{t+1}\right] &\leq \mathbb{E}\left[\alpha^t\right] - \frac{2m\gamma^t}{B}\left(1 - \frac{\rho M}{2}\right)\mathbb{E}\left[\alpha^t\right] + \frac{M^2(\gamma^t)^2}{B}\mathbb{E}\left[\alpha^{t-\tau}\right] \\
&\quad + \sum_{s=1}^{\tau}\frac{\tau M^2\gamma^t(\gamma^{t-s})^2}{\rho B^2}\mathbb{E}\left[\alpha^{t-\tau-s}\right] + \zeta^t. \qquad (94)
\end{aligned}$$

Set $\rho = 1/M$ and use the fact that $\gamma^t \leq \gamma^{t-s}$ for $s \geq 1$ to obtain

$$\mathbb{E}\left[\alpha^{t+1}\right] \leq \mathbb{E}\left[\alpha^t\right] - \frac{m\gamma^t}{B}\mathbb{E}\left[\alpha^t\right] + \frac{M^2(\gamma^t)^2}{B}\mathbb{E}\left[\alpha^{t-\tau}\right] + \sum_{s=1}^{\tau}\frac{\tau M^3(\gamma^{t-s})^3}{B^2}\mathbb{E}\left[\alpha^{t-\tau-s}\right] + \zeta^t. \qquad (95)$$

By setting $\gamma^t = \gamma^0 T^0/(t + T^0)$ in (95) and defining $\phi^t := \mathbb{E}\left[\alpha^t\right]$ we obtain

$$\begin{aligned}
\phi^{t+1} &\leq \left(1 - \frac{m\gamma^0 T^0}{B(t+T^0)}\right)\phi^t + \frac{M^2(\gamma^0 T^0)^2}{B(t+T^0)^2}\phi^{t-\tau} + \sum_{s=1}^{\tau}\frac{\tau M^3(\gamma^0 T^0)^3}{B^2(t+T^0-s)^3}\phi^{t-\tau-s} \\
&\quad + \frac{MK(\gamma^0 T^0)^2}{2B(t+T^0)^2} + \frac{\tau^2 M^2 K(\gamma^0 T^0)^3}{2B^2(t+T^0-\tau)^3}. \qquad (96)
\end{aligned}$$

Next we use induction to prove that

$$\phi^t \leq \frac{Q}{t+T^0}, \tag{97}$$

where Q is defined as

$$Q = \max\left\{\phi^0 T^0, \frac{2MKB(\gamma^0T^0)^2 + 8\Delta^2M^2K(\gamma^0)^3(T^0)^2}{B(3m\gamma^0T^0-4B)}\right\} \tag{98}$$

The base of induction for $t=0$ indeed holds as $\phi^0 \leq \frac{Q}{T^0} \leq \frac{\phi^0T^0}{T^0} = \phi^0$. Next, assume that the inequality in (97) holds for all $t=k$ and we aim to show that it also holds for $t=k+1$. Since (97) holds for all $t \leq k$ then we have

$$\phi^t \leq \frac{Q}{t+T^0}, \tag{99}$$

for all $t \leq k$. Next by setting $t=k$ in (96) and using the upper bound in (99) we obtain that

$$\begin{aligned}\phi^{k+1} \leq& \left(1-\frac{m\gamma^0T^0}{B(k+T^0)}\right)\frac{Q}{k+T^0} + \frac{M^2(\gamma^0T^0)^2}{B(k+T^0)^2}\frac{Q}{k-\tau+T^0}\\ &+\sum_{s=1}^{\tau}\frac{\tau M^3(\gamma^0T^0)^3}{B^2(k+T^0-s)^3}\frac{Q}{k-\tau-s+T^0} + \frac{MK(\gamma^0T^0)^2}{2B(k+T^0)^2} + \frac{\tau^2M^2K(\gamma^0T^0)^3}{2B^2(k+T^0-\tau)^3}.\end{aligned} \tag{100}$$

Note that we have also assumed that $\left(1-\frac{m\gamma^0T^0}{B(k+T^0)}\right) > 0$ which holds since $\frac{m\gamma^0}{B} < \frac{1}{2}$. In the sum, replace s by its upper bound τ to obtain

$$\begin{aligned}\phi^{k+1} \leq& \left(1-\frac{m\gamma^0T^0}{B(k+T^0)}\right)\frac{Q}{k+T^0} + \frac{M^2(\gamma^0T^0)^2}{B(k+T^0)^2}\frac{Q}{k-\tau+T^0} + \frac{\tau^2M^3(\gamma^0T^0)^3}{B^2(k+T^0-\tau)^3}\frac{Q}{k-2\tau+T^0}\\ &+\frac{MK(\gamma^0T^0)^2}{2B(k+T^0)^2} + \frac{\tau^2M^2K(\gamma^0T^0)^3}{2B^2(k+T^0-\tau)^3}\\ \leq& \left(1-\frac{m\gamma^0T^0}{B(k+T^0)}\right)\frac{Q}{k+T^0} + \frac{M^2Q(\gamma^0T^0)^2}{B(k-\tau+T^0)^3} + \frac{\tau^2M^3Q(\gamma^0T^0)^3}{B^2(k+T^0-2\tau)^4}\\ &+\frac{MK(\gamma^0T^0)^2}{2B(k+T^0)^2} + \frac{\tau^2M^2K(\gamma^0T^0)^3}{2B^2(k+T^0-\tau)^3}.\end{aligned} \tag{101}$$

Next, replace τ by its upper bound Δ to obtain

$$\begin{aligned}\phi^{k+1} \leq& \left(1-\frac{m\gamma^0T^0}{B(k+T^0)}\right)\frac{Q}{k+T^0} + \frac{M^2Q(\gamma^0T^0)^2}{B(k-\Delta+T^0)^3} + \frac{\Delta^2M^3Q(\gamma^0T^0)^3}{B^2(k+T^0-2\Delta)^4}\\ &+\frac{MK(\gamma^0T^0)^2}{2B(k+T^0)^2} + \frac{\Delta^2M^2K(\gamma^0T^0)^3}{2B^2(k+T^0-\Delta)^3}.\end{aligned} \tag{102}$$

Further, since we assume that $T^0 \geq 4\Delta$, we have

$$\frac{1}{(k+T^0-2\Delta)^4} \leq \frac{1}{(k+T^0/2)^4} = \frac{16}{(2k+T^0)^4} \leq \frac{16}{(k+T^0)^4} \tag{103}$$

and also

$$\frac{1}{(k+T^0-\Delta)^3} \leq \frac{1}{(k+3T^0/4)^3} \leq \frac{1}{(k+T^0/2)^3} = \frac{8}{(2k+T^0)^4} \leq \frac{8}{(k+T^0)^3} \tag{104}$$

Applying these substations into (102) implies that

$$\phi^{k+1} \leq \left(1-\frac{m\gamma^0T^0}{B(k+T^0)}\right)\frac{Q}{k+T^0} + \frac{8M^2Q(\gamma^0T^0)^2}{B(k+T^0)^3} + \frac{16\Delta^2M^3Q(\gamma^0T^0)^3}{B^2(k+T^0)^4} + \frac{MK(\gamma^0T^0)^2}{2B(k+T^0)^2} + \frac{4\Delta^2M^2K(\gamma^0T^0)^3}{B^2(k+T^0)^3}. \tag{105}$$

Further, we can show that if $\gamma^0 \leq \frac{m}{64M^2}$ then for all $k \geq 0$ we have

$$\frac{8M^2Q(\gamma^0T^0)^2}{B(k+T^0)^3} \leq \frac{1}{8}\frac{mQ\gamma^0T^0}{B(k+T^0)^2} \tag{106}$$

Further, if $\gamma^0 \leq \sqrt{\frac{Bm}{128\Delta^2M^3}}$ then for all $k \geq 0$ we have

$$\frac{16\Delta^2M^3Q(\gamma^0T^0)^3}{B^2(k+T^0)^4} \leq \frac{1}{8}\frac{mQ\gamma^0T^0}{B(k+T^0)^2} \tag{107}$$

Apply these substitutions into (108) to obtain

$$\phi^{k+1} \leq \left(1-\frac{3m\gamma^0T^0}{4B(k+T^0)}\right)\frac{Q}{k+T^0} + \frac{MK(\gamma^0T^0)^2}{2B(k+T^0)^2} + \frac{4\Delta^2M^2K(\gamma^0T^0)^3}{B^2(k+T^0)^3}. \tag{108}$$

Moreover, since $\frac{(T^0)^3}{(k+T^0)^3} \leq \frac{(T^0)^2}{(k+T^0)^2}$ we can simplify the last term and regroup the terms to obtain

$$\phi^{k+1} \leq \left(1-\frac{3m\gamma^0T^0}{4B(k+T^0)}\right)\frac{Q}{k+T^0} + \frac{MKB(\gamma^0T^0)^2 + 8\Delta^2M^2K(\gamma^0)^3(T^0)^2}{2B^2(k+T^0)^2}. \tag{109}$$

To simplify notation define a and b as

$$a := \frac{3m\gamma^0T^0}{4B}, \qquad b := \frac{MKB(\gamma^0T^0)^2 + 8\Delta^2M^2K(\gamma^0)^3(T^0)^2}{2B^2} \tag{110}$$

which allows us to write

$$\phi^{k+1} \leq \left(1-\frac{a}{k+T^0}\right)\frac{Q}{k+T^0} + \frac{b}{(k+T^0)^2}. \tag{111}$$

Considering the definitions of a, b, and Q we can show that $\frac{b}{a-1} \leq Q$. Hence, we have $b \leq (a-1)Q$ which implies that

$$\begin{aligned}\phi^{k+1} &\leq \left(1-\frac{a}{k+T^0}\right)\frac{Q}{k+T^0} + \frac{(a-1)Q}{(k+T^0)^2} \\ &= \frac{Q}{k+T^0} - \frac{Q}{(k+T^0)^2} \\ &= Q\left(\frac{k+T^0-1}{(k+T^0)^2}\right) \\ &\leq Q\left(\frac{1}{k+1+T^0}\right)\end{aligned} \tag{112}$$

where the last inequality holds since $(k+T^0+1)(k+T^0-1) \leq (k+T^0)^2 - 1 < (k+T^0)^2$. Hence, the step of induction is complete and the inequality in (99) also holds for $t = k+1$. The proof is complete. ■

References

Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-Newton stochastic gradient descent. *The Journal of Machine Learning Research*, 10:1737–1754, 2009.

Charles G Broyden, John E Dennis, and Jorge J Moré. On the local and superlinear convergence of quasi-Newton methods. *IMA Journal of Applied Mathematics*, 12(3): 223–245, 1973.

Richard H Byrd, Jorge Nocedal, and Ya-Xiang Yuan. Global convergence of a class of quasi-Newton methods on convex problems. *SIAM Journal on Numerical Analysis*, 24 (5):1171–1190, 1987.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

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

John E Dennis and Jorge J Moré. A characterization of superlinear convergence and its application to quasi-Newton methods. *Mathematics of computation*, 28(126):549–560, 1974.

Francisco Facchinei, Gesualdo Scutari, and Simone Sagratella. Parallel selective algorithms for nonconvex big data optimization. *Signal Processing, IEEE Transactions on*, 63(7): 1874–1889, 2015.

Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

Dong-Hui Li and Masao Fukushima. A modified BFGS method and its global convergence in nonconvex minimization. *Journal of Computational and Applied Mathematics*, 129(1): 15–35, 2001.

Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *The Journal of Machine Learning Research*, 16(1):285–322, 2015.

Zhaosong Lu and Lin Xiao. On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, pages 1–28, 2013.

Zhaosong Lu and Lin Xiao. On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, 152(1-2):615–642, 2015.

Zhi-Quan Luo and Paul Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72 (1):7–35, 1992.

Chenxin Ma and Martin Takáč. Distributed inexact damped newton method: Data partitioning and load-balancing. In *AAAI 2017 Workshop on Distributed Machine Learning*, 2016.

Chenxin Ma, Jakub Konečný, Martin Jaggi, Virginia Smith, Michael I Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *Optimization Methods and Software*, pages 1–36, 2017.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research*, 11:19–60, 2010.

Jakub Mareček, Peter Richtárik, and Martin Takáč. Distributed block coordinate descent for minimizing partially separable functions. *Numerical Analysis and Optimization 2014, Springer Proceedings in Mathematics and Statistics*, 2014.

Shin Matsushima, Hyokun Yun, Xinhua Zhang, and SVN Vishwanathan. Distributed stochastic optimization of regularized risk via saddle-point problem. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 460–476, 2017.

Qi Meng, Wei Chen, Jingcheng Yu, Taifeng Wang, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous accelerated stochastic gradient descent. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1853–1859. AAAI Press, 2016.

Aryan Mokhtari and Alejandro Ribeiro. RES: Regularized stochastic BFGS algorithm. *IEEE Transactions on Signal Processing*, 62(23):6089–6104, 2014.

Aryan Mokhtari and Alejandro Ribeiro. Global convergence of online limited memory BFGS. *The Journal of Machine Learning Research*, 16(1):3151–3181, 2015.

Aryan Mokhtari, Mark Eisen, and Alejandro Ribeiro. IQN: An incremental quasi-Newton method with local superlinear convergence rate. *SIAM Journal on Optimization*, 28(2): 1670–1698, 2018a.

Aryan Mokhtari, Mert Gurbuzbalaban, and Alejandro Ribeiro. Surpassing gradient descent provably: A cyclic incremental method with linear convergence rate. *SIAM Journal on Optimization*, 28(2):1420–1447, 2018b.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, pages 1–52, 2015.

Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. *Journal of Machine Learning Research*, 17:1–25, 2016.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

Geoffrey Sampson, Robin Haigh, and Eric Atwell. Natural language analysis by stochastic optimization: A progress report on project april. *J. Exp. Theor. Artif. Intell.*, 1(4): 271–287, October 1990. ISSN 0952-813X.

Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin. Feature clustering for accelerating parallel coordinate descent. In *Advances in Neural Information Processing Systems*, pages 28–36, 2012.

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-Newton method for online convex optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 436–443, 2007.

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

Virginia Smith, Simone Forte, Chenxin Ma, Martin Takáč, Michael I Jordan, and Martin Jaggi. CoCoA: A general framework for communication-efficient distributed optimization. *The Journal of Machine Learning Research*, 18(1):8590–8638, 2017.

Victor Solo and Xuan Kong. *Adaptive signal processing algorithms: stability and performance*. Prentice-Hall, Inc., 1994.

Murat Taşan, Gabriel Musso, Tong Hao, Marc Vidal, Calum A MacRae, and Frederick P Roth. selecting causal genes from genome-wide association studies via functionally coherent subnetworks. *Nature methods*, 2014.

Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

Lin Xiao, Adams Wei Yu, Qihang Lin, and Weizhu Chen. Dscovr: Randomized primal-dual block coordinate algorithms for asynchronous distributed optimization. *Journal of Machine Learning Research*, 20(43):1–58, 2019.

Yangyang Xu and Wotao Yin. Block stochastic gradient iteration for convex and nonconvex optimization. *SIAM Journal on Optimization*, 25(3):1686–1716, 2015.

Yangyang Xu and Wotao Yin. A globally convergent algorithm for nonconvex optimization based on block coordinate update. *Journal of Scientific Computing*, 72(2):700–734, 2017.

Tianbao Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 629–637, 2013.

Yang Yang, Gesualdo Scutari, and Daniel P Palomar. Parallel stochastic decomposition algorithms for multi-agent systems. In *Signal Processing Advances in Wireless Communications (SPAWC), 2013 IEEE 14th Workshop on*, pages 180–184. IEEE, 2013.